For office use only	Team Control Number	For office use only
T1 _____	**78009**	F1 _____
T2 _____		F2 _____
T3 _____	Problem Chosen	F3 _____
T4 _____	**D**	F4 _____

2018

MCM/ICM

Summary Sheet

Analysis on the Diffusion of Electric Vehicles

Summary

The development of electric vehicles plays an important role in environmental protection and energy saving. Based on the ideal of heat bug model in computer science, this paper combines the existing method of Agent based modeling and GIS technology to create a heat bug model that is dedicated to analyze the diffusion mechanism of electric vehicles.

This paper follows the logic of analyzing the problem and creating model to solve the problem. Firstly the all-electric problem is divided into two parts: the optimization of geographical distribution and that of quantity. Then we assume that the diffusion of electric vehicles and that of charging stations have relevance to some extends. Considering the influence of geography, economy, population and other factors, Monte Carlo method is also used to create a heat bug model that can be used to study the diffusion mechanism of electric vehicles. On this basis, the Annealing Algorithm is used to optimize the existing model and form the diffusion of electric vehicles model.

In this paper, we use the model to analyze the situation of America, South Korea and Australia. The stability and applicability of the model are tested in the practical analysis. And the questions such as the layout and quantity of the optimal charging stations have been answered. Then the advantages and disadvantages of the model are analyzed. Also we find some ways to improve. The model is stable and applicable. It can be further used in the analysis of different countries. It is also an innovation in theory and can provide reference for governments and enterprises.

Contents

Analysis on the Diffusion of Electric Vehicles

1. Introduction

1.1 Background

Promoting new energy vehicles (NEVs) are of great significance to energy conservation, emission reduction and environmental governance, which is regarded as a turning point of emerging industries[1]. In this context, several enterprises strengthen research and development capability actively and have successfully launched new energy automotive products like Tesla.

However, the prominent problems during the promotion of NEVs are the charging stations and the inadequacy of charging facilities. There is a “chicken and egg” problem exists in the diffusion of new energy vehicles and the construction of its charging stations[2][3] : one the one hand, if the potential buyers perceive that there are very few stations or facilities for charging, they will doubt the convenience and become unwilling to own a new energy car; on the other hand, if the suppliers find there are few consumers, they will lack motivation to invest huge amounts of money to build the stations and facilities.

1.2 Literature Review

Scholars put forward that using technological learning method to explore the diffusion mechanism of new energy vehicles[4], which provides an idea for researching the diffusion of NEVs in terms of quantity and distribution. They also explore ways to study the diffusion of NEVs. For example, R. Axelrod proposed that simulation is a way to test whether a theory is scientific. Arthur further put forward the complex adaptive system[5].

Complex adaptive systems refer to the elements of systems that co-evolve with their environment[5].The system consists of multiple individuals that interact with one another and then adapt to each other. Agent based modeling (ABM) is a computer simulation method that can construct the operating environment to simulate the behavior and interaction of a large number of different decision-making bodies. It is considered as an important method to simulate the systematic evolution process of interaction between heterogeneous agents in complex adaptive systems[3].

The use of complex adaptive systems and the construction of ABM can be used to study the diffusion of NEVs. For example, Malte Schwoon used ABM to make scenario analysis for the diffusion of hydrogen station and simulated the diffusion scenario of German hydrogen fuel

cell vehicle technology[6].

1.3 Restatement of the Problem

Combining the realistic and theoretical backgrounds, we analyze the problem of "Out of Gas and Driving on E". We see it as a simple-to-complex process to creating an electric vehicles diffusion model. The model extends from the current and growing Tesla charging stations in the United States to the development of electric vehicles in South Korea. Then we need to examine the replacement of geographical conditions and test its applicability in countries with different population densities and economic developments. Also the key elements of different growth networks and establish a classification system should be discussed. After that, the influence of technical learning factors on the diffusion of electric vehicles needs to be considered.

The problem can be split into four parts:

Part 1: The spatial distribution of electric vehicles and its development trend.

Part 2: The quantitative distribution of electric vehicles and its development trend.

Part 3: The timeline and plans for achieving driving on E.

Part 4: Technology and other factors' influence.

1.4 Planned Approach

1.4.1 Simulating the Diffusion of Electric Vehicles

The relationship between the use of electric vehicles and our life is close. The discussion on the development of a country's electric vehicles industry also has different development in their different administrative regions. We hope to find a way to display the diffusion, which including the spatial distribution and the increase in the number. ArcGIS can be a good tool.

1.4.2 Building a Heat Bug Model

In our opinion, the diffusion of electric vehicles, that is, the number and distribution of electric vehicles can be partly reflected by the number and distribution of charging station. And we think that the popularization of charging station is a complex adaptive system. Each charging station is an individual in the system, which will interact and influence each other but will adapt to each other and achieve a stable state finally. For computer simulation, we plan to use Swarm to build a heat bug model. The heat bug in our model is a charging station and every charging station has a certain range of radiation as well as a most suitable site selection point. And the accumulation of charging station will form a strong service radiation.

1.4.3 The Consideration of Different Affecting Factors

The geographical environment, economic development, population size and technological development will have an impact on the diffusion of electric vehicles. Also we think they will affect the progress of the popularization of electric vehicles to a certain extent. We will take the aspect into consideration.

2. Assumptions and Justifications

In order to simplify the problem, we regard the switch to all-electric as two major parts: environment and heat bug. Then we make the following assumptions.

2.1 Environment

- The number and distribution of charging station can partly reflect those of electric vehicles.
- The popularization of charging station is a complex adaptive system. Each charging station is an individual in the system.
- The system environment has a special property: heat. After a certain period of time the heat will dissipate and disappear.

2.2 Heat bug

- Each charging station is a heat bug.
- Each heat bug emits a certain amount of heat and has its own ideal temperature for survival.
- Heat bug is selfish, which selfishly looking for its best temperature and moving toward points closer to the ideal temperature for survival.
- A single heat bug cannot get enough heat to survive, so it tends to seek for their partners and pile up in order to get enough heat.
- A heat bug can only move to four directions: up, down, left and right.
- Individual optimization will promote overall optimization.

3. Notations

The variables used in the paper are listed as follow:

Symbol Table

Symbol	Definition
F(U)	The level of unhappiness of heat bugs
$T_{working\ areas}$	Preference temperatures of working areas heat bugs
$T_{living\ areas}$	Preference temperatures of living areas heat bugs
$T_{hot\ spots}$	Preference temperatures of hot spots heat bugs
M	Any country
K_i	Weights of each factor
P_1	Total population
P_2	Land area
P_3	Per capita GDP
P_4	The number of personal passenger vehicles
$R_{DarkSize\ Rate}$	The optimal area that the model does not cover
$S_{Areas\ without\ Heat}$	Areas without Heat
$S_{Total\ Area}$	Total Area

4. The diffusion of Electric Vehicles Model

Based on the idea of heat bug model in computer science, we consider the factors of geography, economy and population. And we use the Monte Carlo Algorithm to create our own basic heat model. Then we find the shortcomings and unidentified problems of our existing model in the actual problem analysis and add the method of Annealing Algorithm for optimization. And we also discuss the setting of different charging stations by using Analytic Hierarchy Process.

4.1 Basic Heat Bug Model

In the heat bug model, each heat bug is a charging station and the heat bugs live in the two-dimensional space. We use this to simulate the diffusion of electric vehicles. Because the situation that heat bugs consider their own optimal states can promote overall optimality, we hope to use the model to find a balance state between the individual and the whole group. Also the heat bug model can be a tool to find the optimal results of our further models.

4.1.1 The Characters of Heat Bugs

- ✓ Preference Temperature

  It reflects the number of their partners and the distance from themselves they hope.
- ✓ Initial Heat

  Each heat bug located in a cell, and cause there is a certain temperature in the cell, which will affect other bugs.
- ✓ Heat Diffusion Rate and Evaporation Rate

A heat bug can radiate heat to its surrounding. As the distance increases, the heat radiated by the heat bug will decrease.

4.1.2 Assumptions of the Model

According to the characters of heat bugs, we assume there are there different kinds of heat bugs： hot spots, living areas and work areas in our model. And have the following assumptions:

- **Hot Spots are divided based on road. Heat bugs which belong to hot spots have the minimum requirements to Preference Temperature.** We think the freedom degree of the road direction is small, as a result there just need to be a station in front.
- **Living areas are divided based on demographic factors and have the second lowest preference temperature.** They have the feature that drivers can charge at home and they can also charge on the road.
- **Working areas are divided based on the level of Gross Domestic Product (GDP), which has the most intense preference for temperature.** This group requires many heats around them and then they can charge by the way when going to work, which satisfy them.
- **Each heat bug exists in a single cell and the heat bug will move to a cell which is closer to its ideal temperature.** We divide the two-dimensional space into some cells.

4.1.3 Parameter Setting and Model Statement

To determine the reasonable size of each cell, we choose ArcGIS to assist our work. We normalize the coordinate system of the US map with the WGS1984 coordinate system. Then we express its latitude and longitude range of the United States (about 130°W~70°W, 30°N~50°N) to determine the aspect ratio as follow：

$$\text{Height} : \text{Width} = 570 : 2000$$

After that, we use the latitude and longitude distance calculation formula and get each cell size is 2.9×2.9 square kilometers.

According to the battery life of the electric cars that Tesla announced and considering the capability to continue to drive after low battery alarm (the electric cars like model 3 can continue to travel about 21km after low battery alarm).We set the max output heat as 7 cells and the step of evaporation rate is 1 cell.

Based on these, we use the formula to calculate the level of unhappiness of heat bug:

$$\text{Unhappiness} = \text{F(U)} = \left(\frac{\text{Ideal Temp} - \text{Real Temp}}{\text{Ideal Temp}}\right)^2$$

As for the preference temperature, we consider the three kinds of heat bugs have different preference to temperature:

$$T_{\text{working areas}} > T_{\text{living areas}} > T_{\text{hot spots}}$$

Based on their above relationships, computer iterations are used to make the level of overall unhappiness drop to an acceptable and stable state. When the heat bug selfishly considers its own optimal situation, the overall unhappiness will constantly decline. By using the formula of Unhappiness, we can get the balance state between individual and the whole group through computer iterative algorithm. Then we can achieve the overall optimal result from the individual optimal direction.

4.2 The Optimal Number and Distribution of Charging Station

In combination with the heat bug model, we consider the distribution of heat bugs in specific countries and regions. We choose the United States and as the country to first analysis and combined with the Monte Carlo method to find the optimal layout of the charging stations.

4.2.1 Dividing the Area

We analyze the similarities of GDP and population between different administrative regions to divide the United States into different regions. We counted the GDP and population of each state in the United States and calculated the percentage of population in each area as well as the percentage of total GDP in each area[7]. The similar and adjacent states are divided into a region. Then we get the following six regions.

Table 1. Division of the United States

New Regions	Population (person)	Proportion (%)	GDP (million dollars)	Proportion (%)
New England	131,917,511	41%	131917511	43%
Plains	15,180,575	5%	15180575	5%
Southeast	68,975,010	21%	68975010	17%
Southwest	41,339,800	13%	41339800	12%
Rocky Mountain	12,055,738	4%	12055738	3%
Far West	54,083,211	17%	54083211	19%
Total	323,551,845	100%	323551845	100%

Source: U.S. Federal Statistics(https://fedstats.sites.usa.gov/)

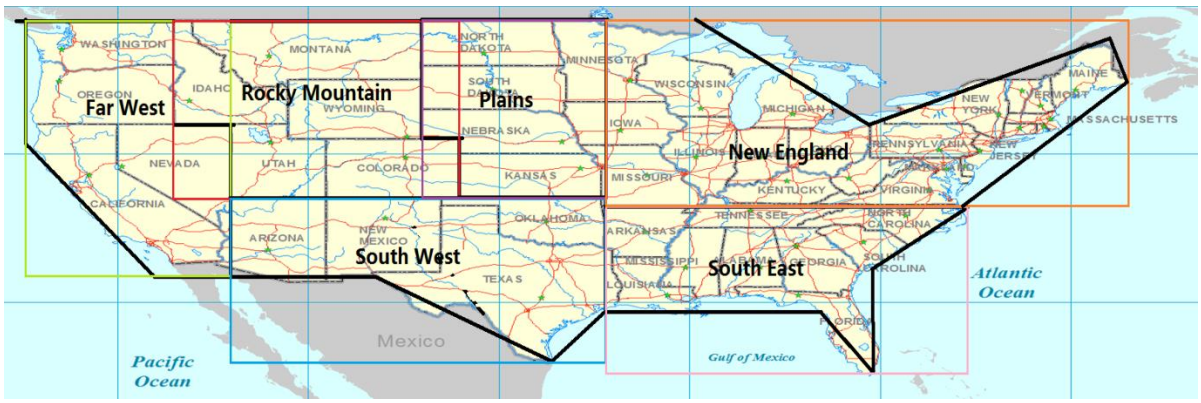


Fig. 1. The United States map segmentation

4.2.2 Setting the Initial Rule

The initial rule is used to determine the initial value and covers the geographical conditions of the United States. We form the terrain shape of the United Stated by using the fuzzy algorithm to linearize the edges. According to our given rules, heat bugs can be accurately placed on the territory of the United States. We set the defined rule as a parent. The parent is represented by the coordinate points on the map, which is a rectangle defined by the starting point, length and width.

The country is divided into six regions, each region has a parent. The points in the parent area are not necessarily in the region but the points in the region must be in the parent area.

4.2.3 The Number of Heat Bugs Should Be Replaced

We search the official website of Tesla for statistics on the number of charging stations during 2013 to 2017 and calculate its growth rate. We also grab the database of the sales and ownership of electric vehicles in U.S. Because we divide the heat bugs into three kinds: hot spots, working areas and living areas. We regard destination charging as living areas and regard super charging as working areas. And we consider the number of highways and make a secondary distribution. Then we draw the figure of how many heat bugs we should put.

Table 2. The number of heat bugs in each area

New regions	Destination Charging	Super Charging	Road（300）
New England	1,274	389	
Plains	147	43	
Southeast	666	157	
Southwest	399	108	
Rocky Mountain	116	31	
Far West	522	171	
Secondary Distribution	2891.08	831.96	300

4.2.4 Heat Bug Model Based on Monte Carlo Algorithm

According to the partitions that are previously set and the rules determined by GDP and the population, the heat bugs are placed randomly, which is also the initial state of Monte Carlo. We use the following constraint formulas:

✓ New England (its parent is 1017,20—1799,257)

$$\begin{cases} x < 1411, y \geq 0.5155x - 574.376 \\ x > 1411, y \geq -0.3139x + 595.8972 \\ y \geq 2.1481x - 3764.3704 \\ \mathrm{y} \leq -0.6667\mathrm{x} + 1296.6667 \end{cases}$$

✓ Plains (its parent is 741,20-1017,246)

$$\mathrm{x} < 798, \mathrm{y} \leq 168$$

✓ South East (its parent is 1016,257-1558,471)

$$\begin{cases} x < 1339, y \leq 391 \\ \mathrm{x} > 1339, \mathrm{y} \leq 1.0256\mathrm{x} - 982.3333 \\ x > 1417, y \leq 339 \\ x > 1417, y \leq -0.5745x + 1153.0213 \end{cases}$$

✓ South West (its parent is 455,247-1016,456)

$$\begin{cases} x < 671, y \leq 348 \\ \mathrm{x} > 671, \mathrm{y} \leq 0.4113\mathrm{x} + 70.5926 \\ \mathrm{y} \leq -0.8101\mathrm{x} + 1215.0886 \end{cases}$$

✓ Rocky Mountain (its parent is 370,20-798,246)

$$\begin{cases} x < 454, y \geq 152 \\ \mathrm{x} > 741, \mathrm{y} \leq 168 \end{cases}$$

✓ Far West (its parent is 149,20-455,347)

$$\begin{cases} y \leq 0.886x + 44.8705 \\ \mathrm{x} > 369, \mathrm{y} \geq 152 \end{cases}$$

4.3 Model Improvement: The Use of Simulated Annealing Algorithms

After analyzing the diffusion mechanism of electric vehicles in the United States based on our basic heat bug model, we discuss the situation in Korea and Australia. We find that as the number of iterations increasing, the phenomenon of moving together will occur (the heat bugs and its partners move together, the overall status has not changed), which may affect the result of optimization. As a result, we use simulated annealing algorithm. That is, we add

stochastic process based on the basic heat bug model. It will improve the phenomenon that the heat bug and its partners move in parallel, so that the overall status can be greatly changed and the result will be more optimal.

4.3.1 Simulated Annealing Algorithm Process

Step 1: Set the initial status.

Determining the initial annealing temperature T_0 and the temperature decay rate $Step$.

Step 2: Solve the solution and determine the feasible solution.

We use the Unhappiness formula to generate four solutions X' (on behalf of the heat bug can move to up, down, left and right directions). And the test of whether the four solutions can be considered as feasible solutions is needed. Defining the formula:

$$\Delta F(U)_{situation} = F(U)_{new} - F(U)_{old}$$

And use $Min[1, e^{\frac{-\Delta F(U)_{situation}}{TK_n}}] > Random(0,1)$ to judge whether the new solutions are feasible. Only the feasible solutions can make contribution value to heat bugs moving.

Step 3: Heat bugs moving.

If the test in Step 2 is completed, the heat bug moves to (offset X, offset Y).

4.3.2 Initial Annealing Temperature

We performed a number of experiments. When the number of iterations N=15, the value of unhappiness decreased to a steady states. Since it may be a pseudo-optimal, we combine the annealing algorithm to increase the number of iterations to 25 times. Based on the preliminary experiments, we what to achieve optimal state at 20 iterations. After 20 iterations even the smallest non-optimal solution cannot be accepted. That is to say, the tolerance of wrong solutions is reduced to less than 1%.

According to the Unhappiness formula:

$$Unhappiness = F(U) = (\frac{Ideal\ \ Temp - Real\ \ Temp}{Ideal\ \ Temp})^2$$

For easy calculation, we use $F(\sqrt{U}) = \sqrt{Unhappiness}$ to judge whether that is the best.

The increased value $\Delta T = F(\sqrt{U})_{new} - F(\sqrt{U})_{old}$

And $\min(|\Delta T|) = \frac{1}{Ideal\ \ Temp}$

When$\Delta T > 0$, the minimum is $\frac{1}{Ideal\ \ Temp}$

Thene$^{-\frac{1}{Ideal\ \ Temp \times T_N}} \leq 1\%$, and we assume$T_N = \frac{T_{N-1}}{Step}$, we get $\frac{(Step)^N}{Ideal\ \ Temp \times T_0} \geq \ln 100$

When N=20,$T_0 \leq \frac{(\text{Step})^{20}}{\text{Ideal Temp}\times\ln 100}$, $\text{Ideal Temp} = 2^4$

Take Step=2, we get $T_0 = 2^{14}$. It is the initial annealing temperature.

4.4 The Use of Analytic Hierarchy Process

Since Tesla is a leading player in the electric car industry, we use the number of Tesla charging stations to approximate the number of charging stations in M. As Tesla is relatively mature in the United States, the number of destination charging and super charging stations required by M is calculated by reference to the number of charge stations in the United States after all-electric. In addition, we also consider the total population (p_1), land area (p_2), per capita GDP (p_3) and the number of personal passenger vehicles (p_4) and use AHP (Analytic Hierarchy Process).

Weight calculation results are as follows:

$$\begin{pmatrix} 1 & 3 & 1/5 & 1/3 \\ 1/3 & 1 & 1/7 & 1/5 \\ 5 & 7 & 1 & 3 \\ 3 & 5 & 1/3 & 1 \end{pmatrix}$$

And we get the weights K_i

$$K_1 = 12.2\% \ , \quad K_2 = 5.7\% \ , \quad K_3 = 55.8\%, \quad K_4 = 26.3\%$$

After the United States all-electric, roughly 260,000 destination charging stations and 80,000 super charging stations will need to be built. According to the following formulas to estimate the number of charging stations (million):

Destination charging stations:

$$\sum (K_i \times P_{iM}/P_{iUS}) \times 26$$

Super charging stations:

$$\sum (K_i \times P_{iM}/P_{iUS}) \times 8$$

5. The Results of the complete switch to all-electric

We analyze the situations of different countries: America, Korea and Australia to investigate the diffusion mechanism of electric vehicles in these countries and test the

applicability of our model. We find some rules and make some conclusions.

5.1 The Situation of America

5.1.1 Whether Tesla is on Track to Allow the Complete Switch in America

To solve the problem of "Is Tesla on track to allow a complete switch in America", we discuss from two parts: the rationality of distribution and the rationality of quantity[8].

As for the rationality of distribution, we use our heat bug model to analyze. After several times of iterations, the value of unhappiness reduces to a relatively stable state.

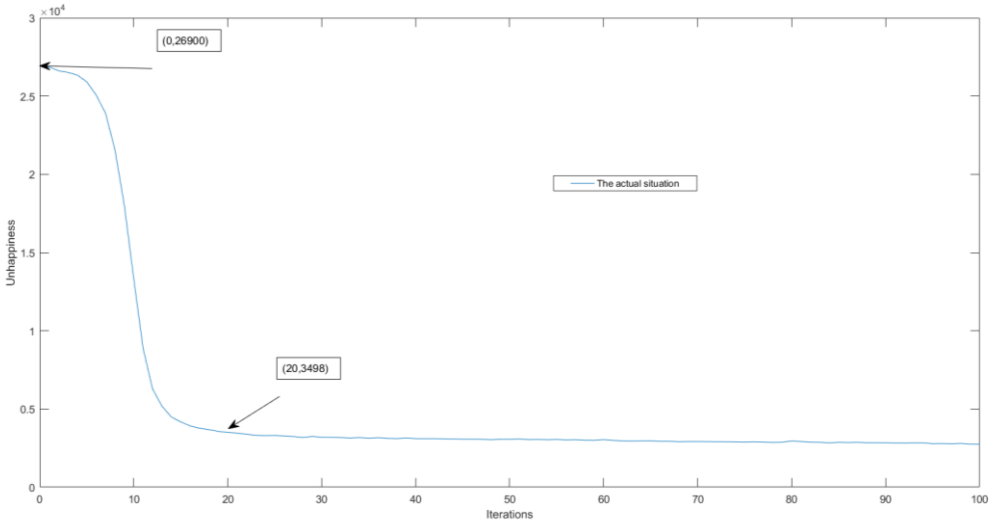


Fig. 2. The value of unhappiness(American)

We combine the American map to make a simulation.

Fig. 3. Initial distribution(American)

Fig. 4. Distribution after 100 iterations(American)

We analyze the actual distribution of Tesla

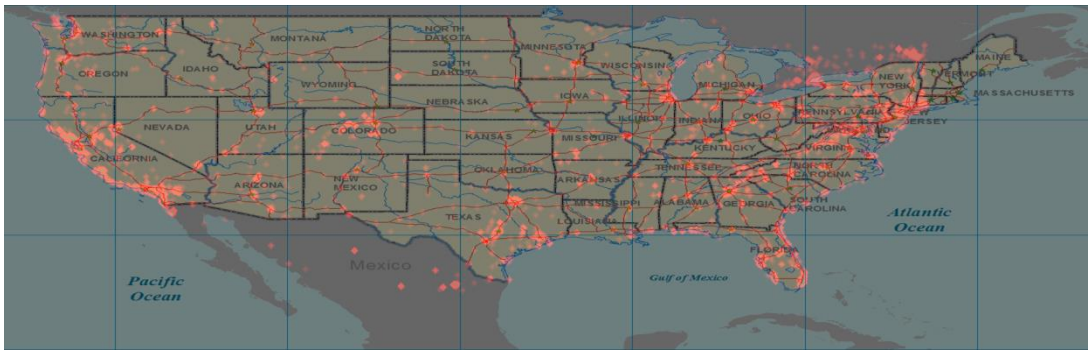


Fig. 5. The actual distribution of Tesla(American)

Comparing the simulation results with the actual distribution of Tesla charging stations, we find that they are nearly the same and we think the charging stations are reasonable in distribution.

As for the rationality of quantity, according to the relationship between the growth rate of electric vehicles and Tesla charging stations, we can determine whether the number of Tesla charging stations can match the demand for charging stations of electric vehicles. According to the searched data, the calculation result is as follows:

Table 3. The growth rate of electric vehicles and Tesla charging stations

the growth rate	electric vehicles	Supercharging	destination charging	total charging stations
2014	100%	110%	110%	110%
2015	50%	70%	70%	70%
2016	40%	110%	70%	80%
2017	40%	60%	40%	40%

As the growth rate of the number of charging stations is greater than the growth rate of electric vehicles, the number of Tesla charging stations can meet the demand.

Above these we think Tesla is on the right track.

5.1.2 The Demand and Distribution of Charging Stations

According to the searched data of 2017, in US, the market share of electric vehicles is

1.17%, there are 3,125 destination charging stations and 899 super charging stations belonging to Tesla. Assuming that the current number of Tesla charging stations in the United States is reasonable, if everyone switched to all-electric personal passenger vehicles in the US, people will require 267094 (3125 / 1.17%) of destination charging stations and 76838 (899 / 1.17%) of super-charging stations, 310,000 of total charging stations.

Select Washington, Iowa and Montana as the representatives of urban, suburban, and rural areas. Then calculate the density of charging stations in the three continents after all-electric according to the proportion of their population and GDP in the United States. We chose the density of charging stations as the standard to divide the United States roughly. After calculation, the division criteria are as follows:

Charging station density is more than 7 / km^2 for the urban, less than 2 / km^2 for the rural areas, and between the two is the suburban.

5.2 The Situation of Korea

5.2.1 The Optimal Distribution and Quantity of Charging Station

We divided the map of South Korea into four regions according to our model, and use the model to simulate the optimization results of charging stations.

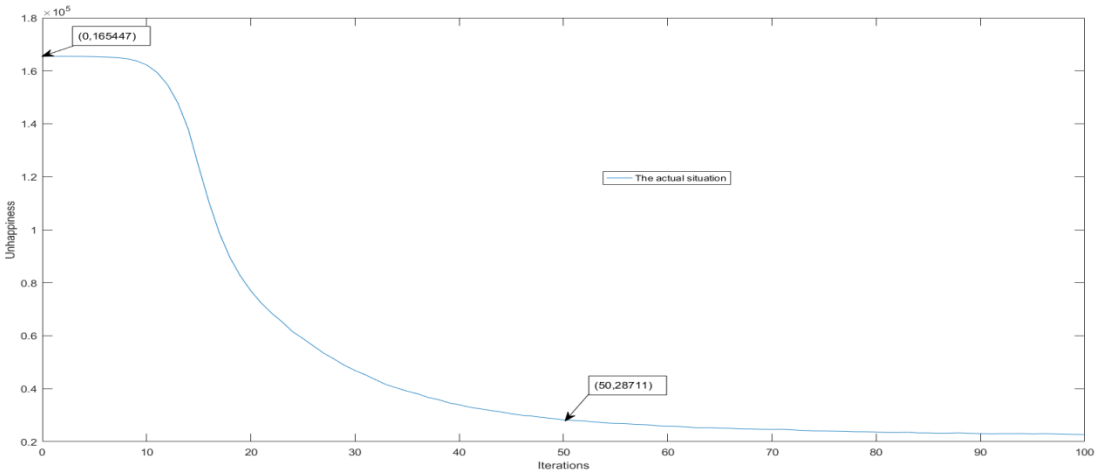
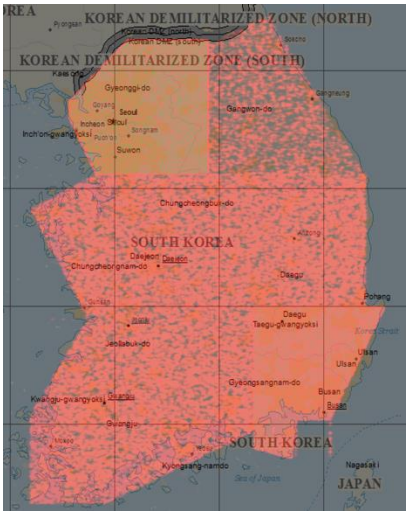


Fig. 6. The value of unhappiness(South Korea)



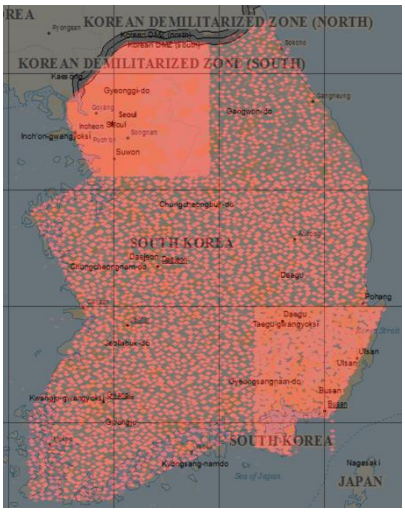


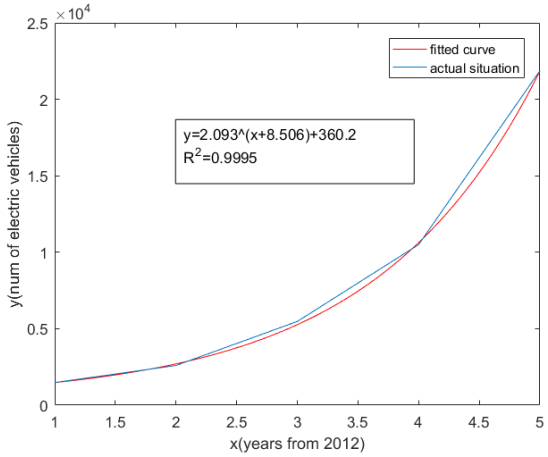
Fig. 7. Initial distribution(South Korea) **Fig. 8. Distribution after 100 iterations(South Korea)**

Also we use the formula $\sum(K_i \times \frac{P_{iM}}{P_{iUS}}) \times 26$ and $\sum(K_i \times \frac{P_{iM}}{P_{iUS}}) \times 8$, then we the results that 80 thousand destination charging stations and 20 thousand super charging stations are needed[9].

5.2.2 Plan for Setting Charging Stations

The country should build city-based chargers and rural chargers at the same time, but pay more attention to the city. Since living areas and working areas are relatively densely distributed in the city, electric vehicles are used more in urban areas and cities have a greater demand for charging stations. However, the establishment of rural charging stations cannot be completely set aside. Rural areas are distant from cities. If there is no charging station in rural areas, people may worry about the power problem when they drive to the countryside. While people consider whether to buy electric cars, charging station coverage is a key factor. It will affect people's desire to buy electric vehicles.

We also consider the timeline of installing, and make some calculations. We use the data of Korean electric vehicles and personal passenger vehicles, the growth curves that fit the growth trend are as follows:



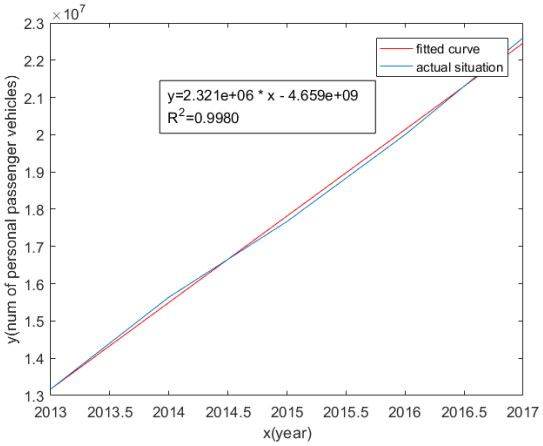


Fig. 9. The growth curves of electric vehicles **Fig. 10. The growth curves of personal passenger vehicles**

According to the ratio of the two curves' function analytic formulas, predict the market share of South Korea's future electric vehicles as follows[8]:

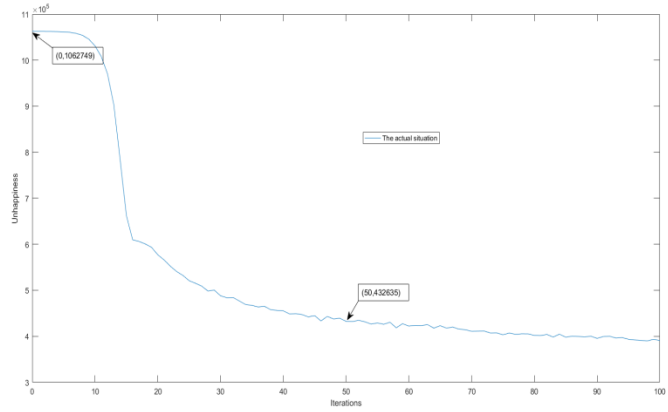
Table 4. The market share of South Korea's future electric vehicles

Year	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027
market share	0.18%	0.35%	0.67%	1.30%	2.54%	4.97%	9.77%	19.29%	38.22%	75.93%

According to market share, it is recommended that South Korea turn to all-electric by 2026. And in our model, we need to put some heat bugs to analyze its diffusion mechanism. The heat bugs represent charging stations. We think that a certain number of charging stations need to be set in advance to promote the further development of electric vehicles.

5.3 The Situation of Australia

We use our model to repeat the analysis of the United States and South Korea we did before[10]. When the Unhappiness of the heat bugs is reduced to a steady state, we get the optimal distribution of charging stations in Australia.



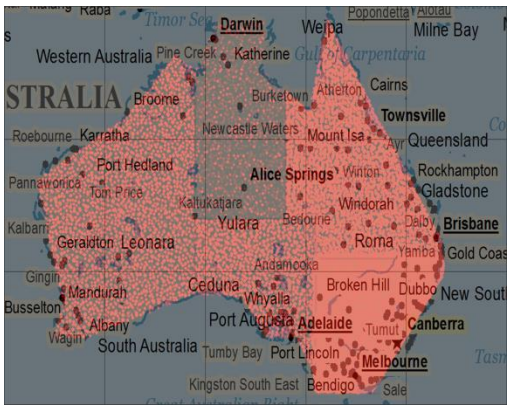


Fig. 11. The value of unhappiness(Australia) **Fig. 12. Distribution after 100 iterations(Australia)**

Comparing the results of our analysis of America, South Korea and Australia, we find that

after many times of iterations, the value of Unhappiness will drop to a stable state. The stability of the model is also reflected in the results. Therefore, we think this model has a applicability and can be used for analyzing different countries.

5.4 The Impact of Technical Factors

We think technologies will definitely impact our analyses of the increasing use of electric vehicles. While people are more inclined to drive by themselves in long-distance travel, car-share and ride-share services can improve car utilization since travel distance is relatively fixed. Some studies show that car-share and ride-share can increase car utilization by 30%. However, the improvement of car utilization will also shorten the retirement cycle of the car, reducing the impact of increased utilization on holdings. Self-driving cars will also increase car utilization, making up for the price disadvantage of electric vehicles. Thus, fuel cars are losing competitiveness in the automotive market due to high fuel and maintenance costs. Rapid battery-swap stations for electric cars will make the electric cars more convenient to use and accelerate the growth of electric vehicle ownership. Hyper loop is mainly for long-distance traffic, while electric vehicles focus on daily travel or relatively short distances. As a result, hyper loop may have an effect on electric vehicle ownership.

6. Further Discussion

6.1 The Present Method for Improvement

After using our heat bug model based on Monte Carlo Algorithm, we have found the phenomenon of moving together will occur: the heat bugs and its partners move together, the overall status has not changed. Then we use simulated annealing algorithm and tend to make an optimization.

We use the size of the heat areas that are not covered by heat bugs to compare the effects of using annealing algorithms and unused.

$$R_{\text{DarkSize Rate}} = S_{\text{Areas without Heat}} / S_{\text{Total Area}}$$

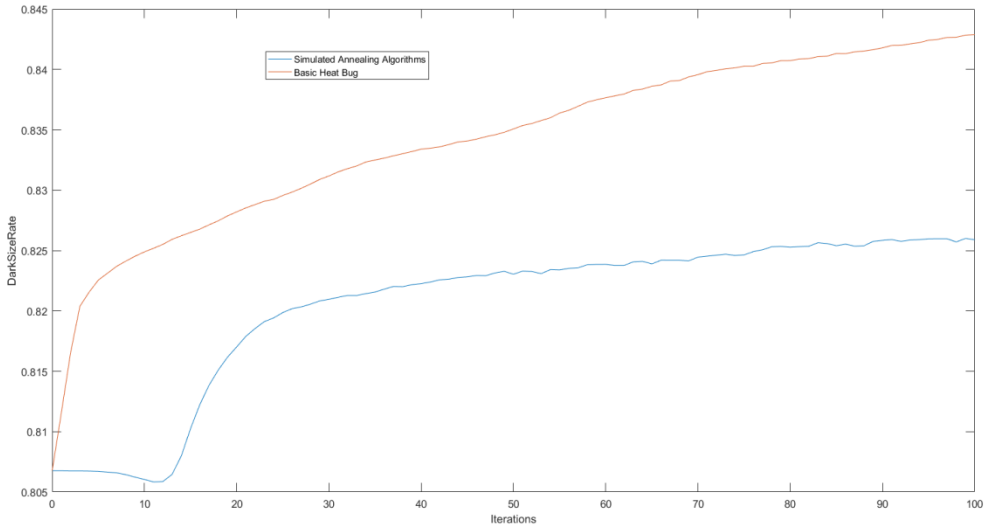


Fig. 13. Dark Size Rate

We find that the uncovered rate of the model after using the Annealing Algorithm is always lower than that of the unused model and as the number of iterations increases, the difference between the two increases further. It shows that the model using the Annealing Algorithm considers the optimal solutions more comprehensive. The model is more optimal.

6.2 Aspects Can Be Further Optimized

- **Initial Annealing Temperature and The Dropping Ratio of Annealing Temperature**

The number of pre-experiments can be increased to improve the performance of the model and the number of iterations can be increased to increase the annealing temperature. Then the dropping function of annealing temperature can be changed to slow down the dropping ratio of annealing temperature.

- **The Choice of Preference Temperature**

The choice of preference temperature is determined by the demands of charging stations. As a result, more experiments and field trips should be conducted to further determine the need to make the preference temperature of different types of heat bugs more accurate.

7. Strengths and Weakness

7.1 Strengths

- **Applicability:** In the analysis of the three countries, the results of the model are stable and have applicability. It can also be further used in the analysis of other countries.
- **Theoretical Innovation:** The factors of geography, economy and population are

considered. And we use the Monte Carlo Algorithm to create our heat bug model to study to diffusion of electric vehicles. Then we use the Annealing Algorithm to make a preliminary optimization. It is also an innovation of the traditional ABM.

- **Practical Significance:** The model simulates the diffusion mechanism of electric vehicles and presents the diffusion effect intuitively, which can provide a reference for governments and enterprises.

7.2 Weaknesses

- The sizes of divided new areas are quite large and the regional division of flat. Making more detailed constraints subdivision blocks is also needed. And we should combine with the characteristics of the blocks to make a multi-level analysis.
- The model only uses the hot spots to examine the impact of highways. A large amount of data and field work are needed to refine the factor of highway in the model. Then we can limit the direction of movement.

2/12/2018

INTERNATIONAL ENERGY SUMMIT

TRANSPORTATION TOWARDS ALL-ELECTRIC CARS

As technology advances, electric vehicles continue to develop in recent years and more consumers chose electric vehicles as transport. Electric vehicles are also increasingly becoming an important tool for dealing with environmental pollution and energy crisis. However, due to technical limitations and hardware bottlenecks, the market share of electric vehicles is still very low. Today, we will focus on how to migrate personal transportation towards all-electric cars and set a gas vehicle-ban date.

THE KEY FACTORS YOU SHOULD CONSIDER

- Determination of the location of the charging stations: Due to the classic "chicken or the egg" quandary, the government needs to establish a certain number of charging stations at the right place first. These charging stations is conducive to electric vehicles sales growth and trigger the proliferation of electric vehicles. You can adopt "The diffusion of Electric Vehicles Model" (in our essay) to specify the location of the charging station.

- Government subsidies: As electric vehicles are generally more expensive than fuel-powered vehicles and their replacement is fast, the cost of consumers buying electric vehicles is relatively high. The government should implement electric car subsidy policies and tax incentives to attract consumers.

- Technology investment：The battery life of electric vehicles is a distinct disadvantage compared to fuel-powered vehicles. Electric car battery research and development requires a lot of investment. And the company have a high risk of investing in the industry. The government should increase investment in electric vehicles and strive to achieve technological breakthroughs.

CONCLUSION

Full electrification is the development trend of the automotive industry, however, there is still a long way to go for the complete ban on the sale of fuel-powered vehicles. We can see many factors that hinder the development of electric vehicles. The leaders of all countries should act together to promote the early realization of this goal.

THANKS FOR ATTENDING

LET'S GET START!

Reference

[1] Hongqi Wang, Yinghua Wang. Evolution Mechanism of Innovative Ecosystem of New Energy Vehicles - A Case Study Based on BYD New Energy Vehicles [J]. China Soft Science, 2016(4): 81-94.
[2] Craig Stephan, Matthew Mahalik, Thomas Veselka, Guenter Conzelmann. Modeling the transition to a hydrogenbased personal transportation system [R]. Frontiers in Transportation. 2007.
[3] Shijian Xiang. Simulation Research on the Diffusion of New Energy Vehicles from the Perspective of Complex Adaptive System [D]. East China University of Science and Technology, 2012.
[4] Gunnar Thesena, Oluf Langhelleb. Awareness, acceptability and attitudes towards hydrogen vehicles and filling stations: A Greater Stavanger case study and comparisons with London [J]. International Journal of Hydrogen Engergy. 2008, 33: 5859-5867.
[5] Arthur WB. Complexity and the economy [J]. Science. 1999, 284(5411): 107-109.
[6] Malte Schwoon. A tool to optimize the initial distribution of hydrogen filling stations [J]. Transportation Research Part D. 2007, 12: 70-82.
[7] U.S. Federal Statistics: https://fedstats.sites.usa.gov/
[8] The Electric Vehicle World Sales Database: http://www.ev-volumes.com/
[9] Korea National Bureau of Statistics : http://kostat.go.kr/portal/korea/
[10] Australian Bureau of Statistics: http://www.abs.gov.au/

Appendix

Program Language：c#

A.Model Realization and Demonstration

1.HeatWorld.CS

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Windows.Forms;

namespace HeatbugModel
{
    public partial class HeatWorld : Form
    {
        public List<Heatbug> Bugs = new List<Heatbug>();
        public Bitmap imgHeatWorld;
        int size;
        Graphics g;
        public HeatWorld()
        {
            InitializeComponent();
        }

        private void HeatWorld_Load(object sender, EventArgs e)
        {
            resetData();
        }
        public void resetData()
        {
            Heatbug.pointHeat = new int[HeatbugParm.worldWidth,
HeatbugParm.worldHeight];
            pbHeatWorld.Width = HeatbugParm.worldWidth * HeatbugParm.Scale;
            pbHeatWorld.Height = HeatbugParm.worldHeight * HeatbugParm.Scale;
            size = HeatbugParm.worldWidth * HeatbugParm.worldHeight;
            this.Width = pbHeatWorld.Width + 18;
            this.Height = pbHeatWorld.Height + 45;
```

```
            imgHeatWorld = new Bitmap(pbHeatWorld.Width, pbHeatWorld.Height);
            g = Graphics.FromImage(imgHeatWorld);
            drawHeatWorld();
        }
        public void addRandomBug(int num)
        {
            Bugs.addRandomBugs(num);
            drawHeatWorld();
        }
        public void addFileBug(FileStream fs)
        {
            Bugs.addFileBugs(fs);
            drawHeatWorld();
        }
        public void nextFrame(bool isdraw = true)
        {
            moveBugs();
            drawHeatWorld(isdraw);
        }
        public void drawHeatWorld(bool isdraw = true)
        {
            refreshHeat();
            if (isdraw)
            {
                g.Clear(Color.Black);
                g.DrawHeat(Heatbug.pointHeat, HeatbugParm.Scale);
                g.DrawBugs(Bugs, HeatbugParm.Scale);
                pbHeatWorld.Image = imgHeatWorld;
            }
        }
        private void moveBugs()
        {
            if (Heatbug.AnnealingTemperature != 1)
            {
                Heatbug.AnnealingTemperature = Heatbug.AnnealingTemperature >> 1;
            }
            Heatbug.rdForAnnealing = new Random(DateTime.Now.Millisecond);
            foreach (var bug in Bugs)
            {
                bug.Move();
            }
        }
        public void refreshHeat()
        {
```

```
            Heatbug.refreshHeat(Bugs);
        }
        public double getAllUnhappiness()
        {
            return Heatbug.getAllUnhappiness(Bugs);
        }
        public double getDarkSizeRate()
        {
            double allSize = Heatbug.pointHeat.GetLength(0) *
Heatbug.pointHeat.GetLength(1);
            double darknum = 0;
            for (int x = 0; x < Heatbug.pointHeat.GetLength(0); x++)
            {
                for (int y = 0; y < Heatbug.pointHeat.GetLength(1); y++)
                {
                    if (Heatbug.pointHeat[x, y] == 0)
                        darknum++;
                }
            }
            return darknum / allSize;
        }
    }
    public static class ExpandGraphics
    {
        public static void DrawBug(this Graphics g, Heatbug Bug,int scale)
        {
            Point point = new Point(Bug.Position.X * scale, Bug.Position.Y * scale);
            g.FillRectangle(Brushes.Green, new Rectangle(point, new Size(scale, scale)));
        }
        public static void DrawBugs(this Graphics g, IEnumerable<Heatbug> Bugs, int
scale)
        {
            foreach (var Bug in Bugs)
            {
                g.DrawBug(Bug, scale);
            }
        }
        public static void DrawHeat(this Graphics g, int[,] pointHeat, int scale)
        {
            for (int x = 0; x < pointHeat.GetLength(0); x++)
            {
                for (int y = 0; y < pointHeat.GetLength(1); y++)
                {
                    Point point = new Point(x * scale, y * scale);
```

```
                        int red = 10 * pointHeat[x, y];
                        red = (red > 256) ? 255 : red;
                        g.FillRectangle(new SolidBrush((Color.FromArgb(red, 0, 0))), new
Rectangle(point, new Size(scale, scale)));
                    }
                }
            }
        }
}
```

2. HeatbugModel.CS

```
namespace HeatbugModel
{
    static class HeatbugParm
    {
        public static int displayFrequency = 1;
        public static int worldWidth = 80;
        public static int worldHeight = 80;
        public static int Scale = 9;
        public static double idealTemp = 0;
        public static void addRandomBugs(this List<Heatbug> bugs, int num)
        {
            int[] allPoint = getRandomArray(worldWidth * worldHeight, num);
            int nowIndex = allPoint.Length - 1;
            for (int i = 0; i < num; i++)
            {
                int now = allPoint[nowIndex];
                nowIndex--;
                int x = (now % worldWidth);
                int y = (now / worldWidth);
                bugs.Add(new Heatbug(x, y, idealTemp));
            }
        }
        public static void addFileBugs(this List<Heatbug> bugs, FileStream fs)
        {
            StreamReader sr = new StreamReader(fs);
            string line;
            while (!string.IsNullOrEmpty(line = sr.ReadLine()))
            {
                string[]subline = line.Split(',');
                int x = Convert.ToInt32(subline[0]);
                int y = Convert.ToInt32(subline[1]);
                bugs.Add(new Heatbug(x, y, idealTemp));
```

```
            }
        }
        private static int[] getRandomArray(int Length, int RandomNum)
        {
            int[] Array = new int[Length];
            for (int i = 0; i < Array.Length; i++)
            {
                Array[i] = i;
            }
            int minPointIndex = Array.Length - 1 - RandomNum;
            Random m = new Random(DateTime.Now.Millisecond);
            int nowRandom, temp;
            for (int i = Array.Length - 1; i > minPointIndex; i--)
            {
                nowRandom = m.Next(0, i + 1);
                temp = Array[i];
                Array[i] = Array[nowRandom];
                Array[nowRandom] = temp;
            }
            return Array;
        }
    }
}
```

2. Heatbug.CS

```
namespace HeatbugModel
{
    public class Heatbug
    {
        public double idealTemp = 1;
        public Point Position;
        public static Random rdForAnnealing;
        public static int AnnealingTemperature = 32768;
        public static int[,] pointHeat;
        public int maxOutputHeat = 7;
        public int stepOutputHeat = 1;
        public int minOutputHeat = 0;
        public double Unhappiness
        {
            get
            {
                return getUnhappiness();
            }
```

```
        }
        public int maxStep
        {
            get
            {
                return (maxOutputHeat - minOutputHeat) / stepOutputHeat;
            }
        }
        public int HeatTo(int x,int y)
        {
            int distance = Math.Abs(this.Position.X - x) + Math.Abs(this.Position.Y - y);
            int heat = maxOutputHeat - distance * stepOutputHeat;
            return (heat < minOutputHeat) ? 0 : heat;
        }
        public Heatbug(int x, int y, double idealtemp = 1)
        {
            Position = new Point(x, y);
            idealTemp = idealtemp;
        }
        public Heatbug(Point position,double idealtemp = 1)
        {
            Position = position;
            idealTemp = idealtemp;
        }
        public double getUnhappiness()
        {
            return getUnhappiness(this.Position.X, this.Position.Y);
        }
        public double getUnhappiness(int x, int y)
        {
            double result = (HeatbugParm.idealTemp - pointHeat[x, y] + this.HeatTo(x,
y)) / idealTemp;
            return result * result;
        }
        public double getUnhappiness(Point point)
        {
            return getUnhappiness(point.X, point.Y);
        }
        public void Move()
        {
            Func<double, double, bool> couldNewUnhappinessAcceptable =
(newUnhappiness, oldUnhappiness) =>
            {
                if (newUnhappiness < oldUnhappiness)
```

```
                {
                    return true;
                }
                else
                {
                    double random = rdForAnnealing.NextDouble();
                    double diffUnhappiness = newUnhappiness - oldUnhappiness;
                    diffUnhappiness *= -1;
                    return Math.Exp(diffUnhappiness / AnnealingTemperature) >
random;
                }
            };
             int offsetX = 0; int offsetY = 0; double nowUnhappiness = getUnhappiness();
             if (this.Position.X - 1 >= 0)
             {
                 double newUnhappiness = getUnhappiness(new Point(this.Position.X - 1,
this.Position.Y));
                 if (couldNewUnhappinessAcceptable(newUnhappiness,
nowUnhappiness))
                 {
                     nowUnhappiness = newUnhappiness;
                     offsetX -= 1;
                 }
             }
             if (this.Position.X + 1 < HeatbugParm.worldWidth)
             {
                 double newUnhappiness = getUnhappiness(new Point(this.Position.X + 1,
this.Position.Y));
                 if (couldNewUnhappinessAcceptable(newUnhappiness,
nowUnhappiness))
                 {
                     offsetX += 1;
                 }
             }
             if (this.Position.Y - 1 >= 0)
             {
                 double newUnhappiness = getUnhappiness(new Point(this.Position.X,
this.Position.Y - 1));
                 if (couldNewUnhappinessAcceptable(newUnhappiness,
nowUnhappiness))
                 {
                     nowUnhappiness = newUnhappiness;
                     offsetY -= 1;
                 }
```

```
                }
                if (this.Position.Y + 1 < HeatbugParm.worldHeight)
                {
                    double newUnhappiness = getUnhappiness(new Point(this.Position.X,
this.Position.Y + 1));
                    if (couldNewUnhappinessAcceptable(newUnhappiness,
nowUnhappiness))
                    {
                        nowUnhappiness = newUnhappiness;
                        offsetY += 1;
                    }
                }
                this.Position.Offset(offsetX, offsetY);
            }
            public static implicit operator Point(Heatbug bug)
            {
                return bug.Position;
            }
            public static void refreshHeat(IEnumerable<Heatbug> Bugs)
            {
                pointHeat = new int[HeatbugParm.worldWidth, HeatbugParm.worldHeight];
                foreach (var bug in Bugs)
                {
                    getHeater(bug.Position, bug);
                }
            }
            public static double getAllUnhappiness(IEnumerable<Heatbug> Bugs)
            {
                double result = 0;
                foreach (var bug in Bugs)
                {
                    result += bug.Unhappiness;
                }
                return result;
            }
            private static void getHeater(Point now, Heatbug bug)
            {
                int distance = Math.Abs(now.X - bug.Position.X) + Math.Abs(now.Y -
bug.Position.Y);
                if (distance >= bug.maxStep)
                    return;
                if (now.X < 0 || now.Y < 0 || now.X >= Heatbug.pointHeat.GetLength(0) ||
now.Y >= Heatbug.pointHeat.GetLength(1))
                    return;
```

```
                Heatbug.pointHeat[now.X, now.Y] += 10 - distance * bug.stepOutputHeat;
                if (bug.Position.Y == now.Y)
                {
                    if (bug.Position.X == now.X)
                    {
                        getHeater(new Point(now.X - 1, now.Y), bug);
                        getHeater(new Point(now.X + 1, now.Y), bug);
                    }
                    if (bug.Position.X > now.X)
                    {
                        getHeater(new Point(now.X - 1, now.Y), bug);
                    }
                    if (bug.Position.X < now.X)
                    {
                        getHeater(new Point(now.X + 1, now.Y), bug);
                    }
                }
                if (bug.Position.Y >= now.Y)
                {
                    getHeater(new Point(now.X, now.Y - 1), bug);
                }
                if (bug.Position.Y <= now.Y)
                {
                    getHeater(new Point(now.X, now.Y + 1), bug);
                }
            }
        }
}
```

B. National Parameters and the Algorithm of Generating Random Points

1.SouthKoreaConst.CS

```
namespace HeatbugModel
{
    class SouthKoreaConst
    {
        /* LongitudeAndLatitude */
        int beginLongitude = 125;
        int endLongitude = 130;
        int beginLatitude = 59;
        int endLatitude = 54;
        /* img */
        int imgWidth = 1104;
```

```
int imgHeight = 2000;
int distancePerPX = 278;
/* destination */
int destinationnumNorthWest = 40299;
int destinationnumNorthEast = 2416;
int destinationnumMidland = 23419;
int destinationnumSouthEast = 13867;
/* super */
int supernumNorthWest = 10075;
int supernumNorthEast = 604;
int supernumMidland = 5855;
int supernumSouthEast = 3467;
/* beginx */
int beginxNorthWest = 348;
int beginxNorthEast = 637;
int beginxMidland = 264;
int beginxSouthEast = 737;
/* endx */
int endxNorthWest = 637;
int endxNorthEast = 967;
int endxMidland = 971;
int endxSouthEast = 1010;
/* beginy */
int beginyNorthWest = 297;
int beginyNorthEast = 129;
int beginyMidland = 749;
int beginySouthEast = 1201;
/* endy */
int endyNorthWest = 749;
int endyNorthEast = 749;
int endyMidland = 1868;
int endySouthEast = 1714;
/* ruler */
public Func<int, int, bool> isNorthWest;
public Func<int, int, bool> isNorthEast;
public Func<int, int, bool> isMidland;
public Func<int, int, bool> isSouthEast;
public SouthKoreaConst()
{
    isNorthWest = (x, y) =>
    {
        if (y < -1.6885 * x + 1090.6066)
            return false;
        if (y > 3.3818 * x - 613.8727)
```

```
                return false;
            return true;
        };
        isNorthEast = (x, y) =>
        {
            if (y < -0.8125 * x + 832.5625)
                return false;
            if (y < 2.6634 * x - 1826.4752)
                return false;
            return true;
        };
        isMidland = (x, y) =>
        {
            if (x < -1.1301 * y + 1249.4309)
                return false;
            if (x < 0.3261 * y - 20.3989 && y < 1243)
                return false;
            if (x < -0.3746 * y + 850.644 && y > 1243)
                return false;
            if (x > 737 && y > 1201)
                return false;
            if (y > -0.7816 * x + 2197.0728)
                return false;
            return true;
        };
        isSouthEast = (x, y) =>
        {
            if (y > 0.4211 * x + 1371.6842)
                return false;
            if (y > -13.1 * x + 12364.3 && x < 823)
                return false;
            if (x > 823 && x < 896 && y > 1583)
                return false;
            if (x > 896 && y > -3.3509 * x + 4585.386)
                return false;
            return true;
        };
    }
    public string getPoints()
    {
        string result = getSuperPoint() + getDestinationPoint();
        return result;
    }
    public string getSuperPoint()
```

```
        {
            string result = "";
            result += RandomPoint(beginxMidland, beginyMidland, endxMidland,
endyMidland, supernumMidland, isMidland);
            result += RandomPoint(beginxNorthEast, beginyNorthEast, endxNorthEast,
endyNorthEast, supernumNorthEast, isNorthEast);
            result += RandomPoint(beginxNorthWest, beginyNorthWest, endxNorthWest,
endyNorthWest, supernumNorthWest, isNorthWest);
            result += RandomPoint(beginxSouthEast, beginySouthEast, endxSouthEast,
endySouthEast, supernumSouthEast, isSouthEast);
            return result;
        }
        public string getDestinationPoint()
        {
            string result = "";
            result += RandomPoint(beginxMidland, beginyMidland, endxMidland,
endyMidland, destinationnumMidland, isMidland);
            result += RandomPoint(beginxNorthEast, beginyNorthEast, endxNorthEast,
endyNorthEast, destinationnumNorthEast, isNorthEast);
            result += RandomPoint(beginxNorthWest, beginyNorthWest, endxNorthWest,
endyNorthWest, destinationnumNorthWest, isNorthWest);
            result += RandomPoint(beginxSouthEast, beginySouthEast, endxSouthEast,
endySouthEast, destinationnumSouthEast, isSouthEast);
            return result;
        }
        private static string RandomPoint(int beginx, int beginy, int endx, int endy, int num,
Func<int, int, bool> ruler)
        {
            int disx = endx - beginx;
            int disy = endy - beginy;
            int[] Array = new int[disx * disy];
            for (int i = 0; i < Array.Length; i++)
            {
                Array[i] = i;
            }
            int minPointIndex = 0;
            Random m = new Random(DateTime.Now.Millisecond);
            int nowRandom, temp;
            for (int i = Array.Length - 1; i > minPointIndex; i--)
            {
                nowRandom = m.Next(0, i + 1);
                temp = Array[i];
                Array[i] = Array[nowRandom];
                Array[nowRandom] = temp;
```

```
                }
                string result = "";
                int index = 0;
                for (int i = 0; i < num; i++)
                {
                    int xnow = Array[index] % disx + beginx;
                    int ynow = Array[index] / disx + beginy;
                    index++;
                    while (!ruler(xnow, ynow))
                    {
                        xnow = Array[index] % disx + beginx;
                        ynow = Array[index] / disx + beginy;
                        index++;
                    }
                    result += xnow.ToString() + "," + ynow.ToString();
                    result += "\r\n";
                }
                return result;
            }
        }
}
```

2.USAConst.CS

```
    class USAConst
    {
        /* LongitudeAndLatitude */
        int beginLongitude = -130;
        int endLongitude = -60;
        int beginLatitude = 50;
        int endLatitude = 30;
        /* img */
        int imgWidth = 2000;
        int imgHeight = 570;
        int distancePerPX = 2900;
        /* destination */
        int destinationnumFarWest = 483;
        int destinationnumRockyMountain = 107;
        int destinationnumPlains = 136;
        int destinationnumNewEngland = 1179;
        int destinationnumSouthWest = 369;
        int destinationnumSouthEast = 616;
        /* super */
        int supernumFarWest = 158;
```

```
int supernumRockyMountain = 29;
int supernumPlains = 40;
int supernumNewEngland = 360;
int supernumSouthWest = 100;
int supernumSouthEast = 145;
/* beginx */
int beginxFarwest = 149;
int beginxRockyMountain = 370;
int beginxPlains = 741;
int beginxNewEngland = 1017;
int beginxSouthWest = 455;
int beginxSouthEast = 1016;
/* endx */
int endxFarwest = 455;
int endxRockyMountain = 798;
int endxPlains = 1017;
int endxNewEngland = 1799;
int endxSouthWest = 1016;
int endxSouthEast = 1558;
/* beginy */
int beginyFarwest = 20;
int beginyRockyMountain = 20;
int beginyPlains = 20;
int beginyNewEngland = 20;
int beginySouthWest = 247;
int beginySouthEast = 257;
/* endy */
int endyFarwest = 347;
int endyRockyMountain = 246;
int endyPlains = 246;
int endyNewEngland = 257;
int endySouthWest = 456;
int endySouthEast = 471;
/* ruler */
public Func<int, int, bool> isFarWest;
public Func<int, int, bool> isRockyMountain;
public Func<int, int, bool> isPlains;
public Func<int, int, bool> isNewEngland;
public Func<int, int, bool> isSouthWest;
public Func<int, int, bool> isSouthEast;
public USAConst()
{
    isFarWest = (x, y) =>
    {
```

```
        if (x > 369 && y < 152)
            return false;
        if (y > 0.886 * x + 44.8705)
            return false;
        return true;
};
isRockyMountain = (x, y) =>
{
        if (x < 454 && y > 152)
            return false;
        if (x > 741 && y < 168)
            return false;
        return true;
};
isPlains = (x, y) =>
{
        if (x < 798 && y > 168)
            return false;
        return true;
};
isNewEngland = (x, y) =>
{
        if (x < 1411 && y < 0.5155 * x - 574.376)
            return false;
        if (x > 1411 && y < -0.3139 * x + 595.8972)
            return false;
        if (y < 2.1481 * x - 3764.3704)
            return false;
        if (y > -0.6667 * x + 1296.6667)
            return false;
        return true;
};
isSouthWest = (x, y) =>
{
        if (x < 671 && y > 348)
            return false;
        if (x > 671 && y > 0.4113 * x + 70.5926)
            return false;
        if (y > -0.8101 * x + 1215.0886)
            return false;
        return true;
};
isSouthEast = (x, y) =>
{
```

```
                    if (x < 1339 && y > 391)
                        return false;
                    if (x > 1339 && y > 1.0256 * x - 982.3333)
                        return false;
                    if (x > 1417 && y > 339)
                        return false;
                    if (x > 1417 && y > -0.5745 * x + 1153.0213)
                        return false;
                    return true;
                };
            }
            public string getPoints()
            {
                string result = getSuperPoint() + getDestinationPoint();
                return result;
            }
            public string getSuperPoint()
            {
                string result = "";
                result += RandomPoint(beginxFarwest, beginyFarwest, endxFarwest,
endyFarwest, supernumFarWest, isFarWest);
                result += RandomPoint(beginxNewEngland, beginyNewEngland,
endxNewEngland, endyNewEngland, supernumNewEngland, isNewEngland);
                result += RandomPoint(beginxPlains, beginyPlains, endxPlains, endyPlains,
supernumPlains, isPlains);
                result += RandomPoint(beginxRockyMountain, beginyRockyMountain,
endxRockyMountain, endyRockyMountain, supernumRockyMountain, isRockyMountain);
                result += RandomPoint(beginxSouthEast, beginySouthEast, endxSouthEast,
endySouthEast, supernumSouthEast, isSouthEast);
                result += RandomPoint(beginxSouthWest, beginySouthWest, endxSouthWest,
endySouthWest, supernumSouthWest, isSouthWest);
                return result;
            }
            public string getDestinationPoint()
            {
                string result = "";
                result += RandomPoint(beginxFarwest, beginyFarwest, endxFarwest,
endyFarwest, destinationnumFarWest, isFarWest);
                result += RandomPoint(beginxNewEngland, beginyNewEngland,
endxNewEngland, endyNewEngland, destinationnumNewEngland, isNewEngland);
                result += RandomPoint(beginxPlains, beginyPlains, endxPlains, endyPlains,
destinationnumPlains, isPlains);
                result += RandomPoint(beginxRockyMountain, beginyRockyMountain,
endxRockyMountain, endyRockyMountain, destinationnumRockyMountain,
```

```
isRockyMountain);
                result += RandomPoint(beginxSouthEast, beginySouthEast, endxSouthEast,
endySouthEast, destinationnumSouthEast, isSouthEast);
                result += RandomPoint(beginxSouthWest, beginySouthWest, endxSouthWest,
endySouthWest, destinationnumSouthWest, isSouthWest);
                return result;
            }
            private static string RandomPoint(int beginx, int beginy, int endx, int endy, int num,
Func<int, int, bool> ruler)
            {
                int disx = endx - beginx;
                int disy = endy - beginy;
                int[] Array = new int[disx * disy];
                for (int i = 0; i < Array.Length; i++)
                {
                    Array[i] = i;
                }
                int minPointIndex = 0;
                Random m = new Random(DateTime.Now.Millisecond);
                int nowRandom, temp;
                for (int i = Array.Length - 1; i > minPointIndex; i--)
                {
                    nowRandom = m.Next(0, i + 1);
                    temp = Array[i];
                    Array[i] = Array[nowRandom];
                    Array[nowRandom] = temp;
                }
                string result = "";
                int index = 0;
                for (int i = 0; i < num; i++)
                {
                    int xnow = Array[index] % disx + beginx;
                    int ynow = Array[index] / disx + beginy;
                    index++;
                    while (!ruler(xnow, ynow))
                    {
                        xnow = Array[index] % disx + beginx;
                        ynow = Array[index] / disx + beginy;
                        index++;
                    }
                    result += xnow.ToString() + "," + ynow.ToString();
                    result += "\r\n";
                }
                return result;
```

```
            }
        }
```

3.AustraliaConst.CS

```
namespace HeatbugModel
{
    class AustraliaConst
    {
        /* LongitudeAndLatitude */
        int beginLongitude = 110;
        int endLongitude = 160;
        int beginLatitude = -10;
        int endLatitude = -40;
        /* img */
        int imgWidth = 2000;
        int imgHeight = 1333;
        int distancePerPX = 2500;
        /* destination */
        int destinationnumWest = 16695;
        int destinationnumNorth = 1587;
        int destinationnumNorthEast = 30724;
        int destinationnumSouthEast = 90140;
        int destinationnumSouth = 10855;
        /* super */
        int supernumWest = 6241;
        int supernumNorth = 578;
        int supernumNorthEast = 7693;
        int supernumSouthEast = 23016;
        int supernumSouth = 2472;
        /* beginx */
        int beginxWest = 127;
        int beginxNorth = 757;
        int beginxNorthEast = 1121;
        int beginxSouthEast = 1243;
        int beginxSouth = 757;
        /* endx */
        int endxWest = 757;
        int endxNorth = 1121;
        int endxNorthEast = 1741;
        int endxSouthEast = 1741;
        int endxSouth = 1243;
        /* beginy */
        int beginyWest = 165;
```

```
int beginyNorth = 49;
int beginyNorthEast = 35;
int beginySouthEast = 843;
int beginySouth = 713;
/* endy */
int endyWest = 1111;
int endyNorth = 713;
int endyNorthEast = 841;
int endySouthEast = 1283;
int endySouth = 1255;
/* ruler */
public Func<int, int, bool> isWest;
public Func<int, int, bool> isNorth;
public Func<int, int, bool> isNorthEast;
public Func<int, int, bool> isSouthEast;
public Func<int, int, bool> isSouth;
public AustraliaConst()
{
    isWest = (x, y) =>
    {
        if (x < 485 && y < -0.5922 * x + 656.2067)
            return false;
        if (x >= 485 && x < 679 && y < -1.0515 * x + 879)
            return false;
        if (x >= 679 && y < 0.6 * x - 242.4)
            return false;
        if (y > 581 && y < 1005 && x < 0.2547 * y - 20.9906)
            return false;
        if (y >= 1005 && x < -0.4151 * y + 652.1698)
            return false;
        if (y > -0.3426 * x + 1222.3426)
            return false;
        return true;
    };
    isNorth = (x, y) =>
    {
        if (y < -1.907 * x + 1660.3953)
            return false;
        if (y < 211 && x > -0.1235 * y + 1 SQWQEFDSV049.0494)
            return false;
        if (y >= 211 && y < 281 && x > 1.3714 * y + 733.6286)
            return false;
        return true;
    };
```

```
            isNorthEast = (x, y) =>
            {
                if (y < 347 && x < 1213 && x > 1.3714 * y + 733.6286)
                    return false;
                if (x >= 1213 && x < 1283 &&    y < -4.4286 * x + 5718.8571)
                    return false;
                if (x >= 1283 && y < 1.6295 * x - 2053.6027)
                    return false;
                if (x < 1239 && y > 709)
                    return false;
                return true;
            };
            isSouthEast = (x, y) =>
            {
                if (y > -1.4029 * x + 3335.4101)
                    return false;
                return true;
            };
            isSouth = (x, y) =>
            {
                if (x < 945 && y > 0.0532 * x + 922.734)
                    return false;
                if (x >= 945 && x < 1037 && y > 1.4565 * x - 403.413)
                    return false;
                if (x >= 1037 && x < 1161 && y > 1107)
                    return false;
                if (x >= 1161 && y > 1.8049 * x - 988.4634)
                    return false;
                return true;
            };
        }
        public string getPoints()
        {
            string result = getSuperPoint() + getDestinationPoint();
            return result;
        }
        public string getSuperPoint()
        {
            string result = "";
            result += RandomPoint(beginxWest, beginyWest, endxWest, endyWest,
supernumWest, isWest);
            result += RandomPoint(beginxNorth, beginyNorth, endxNorth, endyNorth,
supernumNorth, isNorth);
            result += RandomPoint(beginxNorthEast, beginyNorthEast, endxNorthEast,
```

```
endyNorthEast, supernumNorthEast, isNorthEast);
                result += RandomPoint(beginxSouthEast, beginySouthEast, endxSouthEast,
endySouthEast, supernumSouthEast, isSouthEast);
                result += RandomPoint(beginxSouth, beginySouth, endxSouth, endySouth,
supernumSouth, isSouth);
                return result;
            }
            public string getDestinationPoint()
            {
                string result = "";
                result += RandomPoint(beginxWest, beginyWest, endxWest, endyWest,
destinationnumWest, isWest);
                result += RandomPoint(beginxNorth, beginyNorth, endxNorth, endyNorth,
destinationnumNorth, isNorth);
                result += RandomPoint(beginxNorthEast, beginyNorthEast, endxNorthEast,
endyNorthEast, destinationnumNorthEast, isNorthEast);
                result += RandomPoint(beginxSouthEast, beginySouthEast, endxSouthEast,
endySouthEast, destinationnumSouthEast, isSouthEast);
                result += RandomPoint(beginxSouth, beginySouth, endxSouth, endySouth,
destinationnumSouth, isSouth);
                return result;
            }
            private static string RandomPoint(int beginx, int beginy, int endx, int endy, int num,
Func<int, int, bool> ruler)
            {
                int disx = endx - beginx;
                int disy = endy - beginy;
                int[] Array = new int[disx * disy];
                for (int i = 0; i < Array.Length; i++)
                {
                    Array[i] = i;
                }
                int minPointIndex = 0;
                Random m = new Random(DateTime.Now.Millisecond);
                int nowRandom, temp;
                for (int i = Array.Length - 1; i > minPointIndex; i--)
                {
                    nowRandom = m.Next(0, i + 1);
                    temp = Array[i];
                    Array[i] = Array[nowRandom];
                    Array[nowRandom] = temp;
                }
                string result = "";
                int index = 0;
```

```
                for (int i = 0; i < num; i++)
                {
                    int xnow = Array[index] % disx + beginx;
                    int ynow = Array[index] / disx + beginy;
                    index++;
                    while (!ruler(xnow, ynow))
                    {
                        xnow = Array[index] % disx + beginx;
                        ynow = Array[index] / disx + beginy;
                        index++;
                    }
                    result += xnow.ToString() + "," + ynow.ToString();
                    result += "\r\n";
                }
                return result;
            }
        }
}
```

C: Iteration Process

Property: The three groups of data in the form of matlab array.
Meaning:

(Unhappiness) refers to the unhappiness value changing with the number of iterations increasing.
(Dark Size Rate) refers to the value of Dark Size with the number of iteration increasing.
(Time) the current number of iterations.

1. America

unhappiness=[26900.609375,26866.484375,26616.0625,26513.51953125,26343.94140625,25912.55078125,25068.109375,23907.44921875,21574.89453125,18011.48828125,13338.4140625,8865.76953125,6288.0859375,5178.52734375,4488.703125,4174.64453125,3919.00390625,3766.9609375,3679.9921875,3567.5859375,3497.52734375,3463.1953125,3380.24609375,3321.328125,3292.35546875,3308.67578125,3267.9375,3238.375,3181.2890625,3246.1171875,3194.29296875,3189.05859375,3174.33984375,3136.91796875,3170.40234375,3125.671875,3159.71875,3119.90625,3106.5234375,3140.50390625,3106.01171875,3102.91015625,3103.1796875,3094.30859375,3085.80859375,3068.34375,3067.91015625,3060.9375,3033.58984375,3064.12890625,3061.98046875,3083.5234375,3040.8984375,3050.078125,3033.390625,3050.734375,3020.4921875,3033.1015625,3002.01953125,2998.6875,3037.62890625,2994.1484375,2963.640625,2952.171875,2964.30078125,2967.9453125,2930.1015625,2931.44921875,2906.17578125,2915.88671875,2917.57421875,2910.90625,2905.9609375,2901.7109375,2894.8828125,2883.41015625,2899.08203125,2883.17578125,2865.2109375,2876.12109375,2949.8984375,2916.57421875,2880.2578125,2875.359375,2832.265625,2870.464

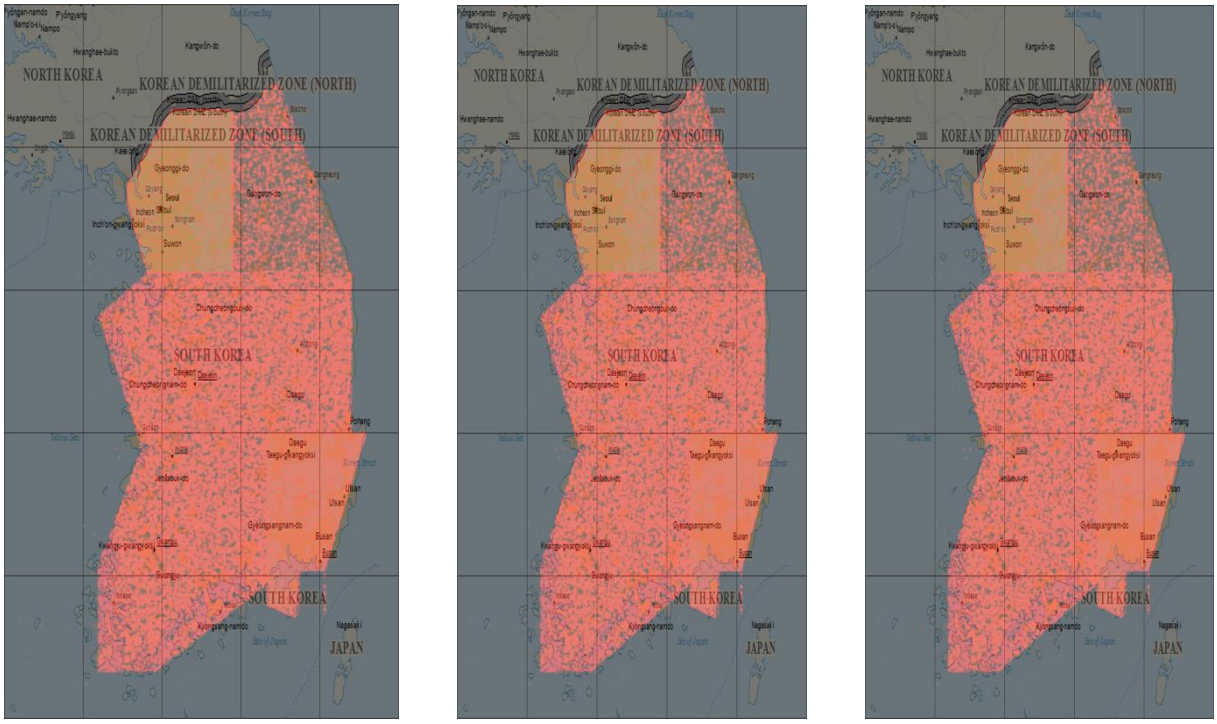
84375,2857.91015625,2872.421875,2845.16015625,2844.1953125,2840.33984375,2821.121
09375,2816.875,2833.15625,2830.2734375,2779.66796875,2786.8984375,2775.08984375,27
95.08203125,2753.10546875,2750.9375];darksizerate=[0.806760526315789,0.80675526315
7895,0.806743859649123,0.806743859649123,0.806731578947368,0.806705263157895,0.8
06649122807018,0.806589473684211,0.806435964912281,0.806226315789474,0.80604122
8070175,0.805833333333333,0.805868421052632,0.806462280701754,0.80799298245614,0
.810256140350877,0.812231578947368,0.813837719298246,0.815099122807018,0.8161640
35087719,0.817019298245614,0.817898245614035,0.818544736842105,0.81911491228070
2,0.819417543859649,0.819864912280702,0.820185087719298,0.820335964912281,0.8205
5701754386,0.820829824561404,0.820963157894737,0.821122807017544,0.821282456140
351,0.821275438596491,0.821430701754386,0.821561403508772,0.82180350877193,0.822
028070175439,0.82201052631579,0.822173684210526,0.822256140350877,0.82239298245
614,0.822574561403509,0.822628070175439,0.822766666666667,0.822821929824561,0.82
2934210526316,0.822916666666667,0.823143859649123,0.823279824561403,0.823035087
719298,0.823298245614035,0.823264035087719,0.823086842105263,0.823418421052632,0
.823390350877193,0.823524561403509,0.823570175438596,0.823826315789474,0.82385,0.
823856140350877,0.823763157894737,0.823768421052632,0.824066666666667,0.8241043
85964912,0.823893859649123,0.824197368421053,0.824190350877193,0.82418859649122
8,0.82414298245614,0.824456140350877,0.824534210526316,0.824614912280702,0.82469
7368421053,0.82459298245614,0.824638596491228,0.824911403508772,0.8250631578947
37,0.825311403508772,0.825337719298246,0.825274561403509,0.825321929824561,0.825
350877192983,0.82564649122807,0.825576315789474,0.825400877192983,0.82552982456
1404,0.825360526315789,0.82539649122807,0.825761403508772,0.82584298245614,0.825
906140350877,0.825761403508772,0.825869298245614,0.82590350877193,0.82597105263
1579,0.825981578947368,0.825975438596491,0.825711403508772,0.826007894736842,0.8
2589473684210 5];time=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,
26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,
56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,
86,87,88,89,90,91,92,93,94,95,96,97,98,99,100];

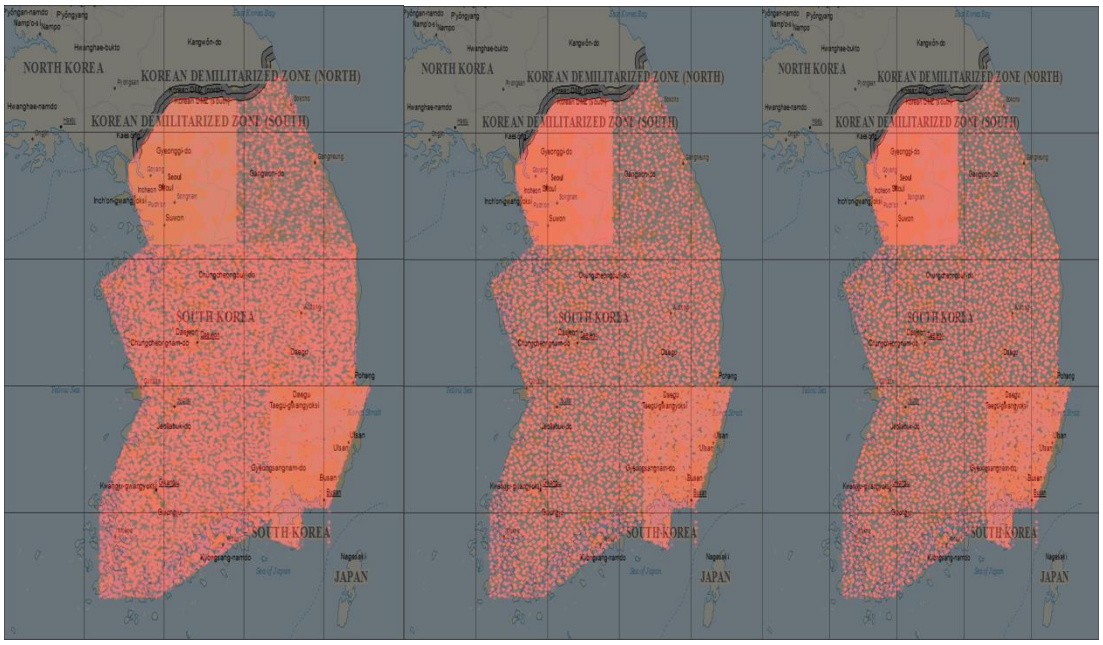
2. South Korea

unhappiness=[165447.917663574,165446.179382324,165432.451843262,165418.783935547

,165398.508239746,165330.733825684,165204.368041992,165004.942443848,164623.9838
86719,163792.155822754,162202.852722168,159384.91394043,154786.206054688,147826.
709960938,137947.956848145,123764.079101563,110219.212524414,98574.7533569336,89
486.8029174805,82509.6358642578,76912.2512817383,72344.7843017578,68493.47473144
53,65234.2277832031,61551.4036865234,58938.1560058594,56147.8622436523,53392.676
7578125,51177.9025268555,48804.9706420898,46843.3500976563,45223.0872802734,4333
5.6172485352,41519.7015991211,40299.227722168,39066.5181274414,38086.188293457,3
6714.1469116211,35877.0191650391,34656.3984985352,33936.2702636719,33011.6214599
609,32411.8844604492,31775.3551635742,31238.1726074219,30558.2590332031,29970.31
62841797,29670.3889160156,29153.936340332,28710.7369995117,28236.8756713867,2798
1.409362793,27813.6340332031,27425.1836547852,27148.841796875,26882.2927856445,2
6759.1612548828,26560.3038330078,26372.0529174805,26029.9056396484,25879.2127075
195,25750.1529541016,25495.2236328125,25224.6206665039,25288.0551147461,25091.35
92529297,25054.037902832,24881.1744384766,24829.6321411133,24662.4432373047,2462
2.0205688477,24684.0314941406,24436.9653320313,24233.1793212891,24057.258605957,
24027.5704956055,23971.3952026367,23889.4287719727,23752.6275024414,23747.717712
4023,23679.4645996094,23497.5029907227,23471.2862548828,23568.1653442383,23324.6
268310547,23319.5997314453,23187.6696166992,23228.6776733398,23338.1210327148,23
149.9685058594,23064.4884643555,22942.8352661133,23063.8648071289,23044.02941894
53,23080.8684692383,22931.8436279297,23034.3438720703,22921.4536132813,22806.151
4282227,22737.7619628906,22691.1784057617];darksizerate=[0.608013586956522,0.60801
3586956522,0.608014039855073,0.608013586956522,0.608015851449275,0.608017210144
927,0.608027626811594,0.608049365942029,0.608060688405797,0.608094655797101,0.60
8158061594203,0.608280344202899,0.608468297101449,0.608867300724638,0.609692028
985507,0.611290307971014,0.612985960144928,0.614874094202899,0.617291666666667,0
.619641304347826,0.622160326086956,0.62486865942029,0.627650815217391,0.63060960
1449275,0.633573369565217,0.636522644927536,0.639734601449275,0.642881793478261,
0.646054347826087,0.648977355072464,0.652104619565217,0.655142210144928,0.658096
467391304,0.66094972826087,0.66379981884058,0.666458786231884,0.669154891304348,
0.671552083333333,0.67413768115942,0.676586050724638,0.678669384057971,0.6808147
64492754,0.682805706521739,0.684697463768116,0.686545289855072,0.68821422101449
3,0.689998188405797,0.691623641304348,0.693227355072464,0.694672101449275,0.6960
70652173913,0.697272644927536,0.698717391304348,0.699807065217391,0.70092934782
6087,0.702123641304348,0.702985960144928,0.70392527173913,0.704889039855072,0.70
5991394927536,0.706757699275362,0.707511322463768,0.708265398550725,0.709002717
391304,0.709669384057971,0.710359148550725,0.711105525362319,0.711465579710145,0
.712066576086957,0.712572463768116,0.712942934782609,0.713564311594203,0.7138550
72463768,0.714405344202899,0.715009510869565,0.715419836956522,0.71600588768115
9,0.716385416666667,0.716775815217391,0.717147644927536,0.717716485507246,0.7179
79166666667,0.718372282608696,0.718813858695652,0.719101449275362,0.71948596014
4927,0.71970606884058,0.719878170289855,0.720115942028986,0.720475543478261,0.72
0803894927536,0.721029438405797,0.721252717391304,0.721451992753623,0.721701992
753623,0.722034420289855,0.722198822463768,0.722472826086957,0.722905797101449,0
.72313768115942,0.72321240942029];time=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,1

9,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100];

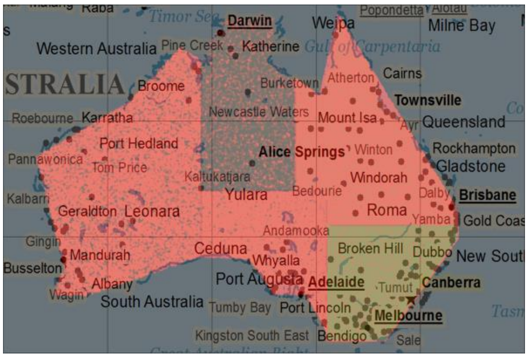


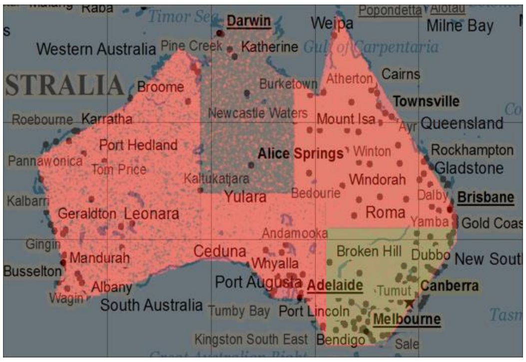


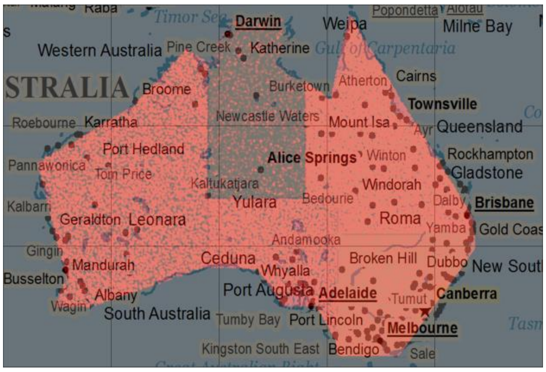
3. Australia

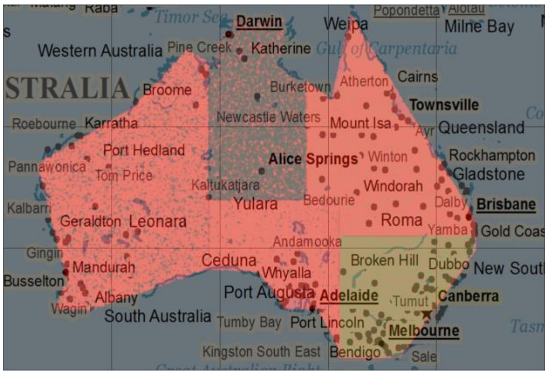
unhappiness=[1062749.44567871,1062685.93511963,1062601.32635498,1062492.19146729,1062213.26043701,1061713.78759766,1060570.73571777,1058290.79797363,1054149.80682373,1045904.88153076,1031234.54333496,1007874.91748047,970011.70300293,902832.805358887,784252.918579102,662480.533691406,609686.564880371,606191.146118164,600749.250915527,593518.563720703,577203.460571289,566062.11126709,552376.88446044

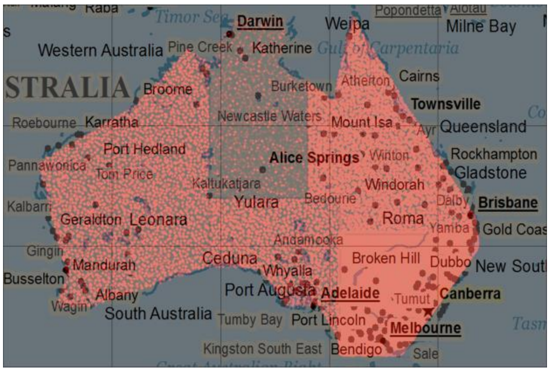
9,540832.379821777,532438.08807373,520862.561950684,515248.557495117,509061.1373
90137,498346.437866211,500395.377746582,488117.503234863,483776.728088379,484289
.785095215,476776.605407715,469005.737915039,467241.933532715,463698.1953125,465
169.96307373,457756.936706543,456275.02734375,455312.750305176,448620.823181152,
449200.528320313,447200.845581055,442180.83581543,444991.58795166,433351.625,443
224.131530762,437709.08416748,439178.674560547,432635.536987305,432058.133911133
,434820.938415527,431534.834472656,426567.645690918,428974.222167969,426121.3623
04688,430532.632873535,418416.467773438,427459.96295166,422206.334594727,423687.
816162109,423342.8125,425250.182617188,417795.632385254,423070.717712402,417791.
957519531,420122.30456543,415494.007995605,414418.344177246,410808.008483887,411
098.609741211,411399.852844238,407204.54473877,407526.48449707,403221.718566895,
407189.087463379,404300.629455566,405492.085998535,405337.797973633,402040.51629
6387,401623.403625488,404104.767333984,398366.157531738,404670.811828613,398099.
854064941,400211.803283691,399729.696655273,398620.96472168,400232.575866699,395
485.042053223,399496.406066895,400093.348388672,396289.335571289,397094.59881591
8,393555.149902344,392064.570739746,391223.954711914,389971.461853027,393546.276
733398,390760.555664063];darksizerate=[0.569749812453113,0.569749812453113,0.56974
943735934,0.56974943735934,0.569752438109527,0.569753188297074,0.56975468867216
8,0.569757689422356,0.569767816954239,0.56978544636159,0.569809827456864,0.56985
1087771943,0.569981620405101,0.570189797449362,0.570709677419355,0.571833083270
818,0.57303375843961,0.574388222055514,0.575905101275319,0.577583645911478,0.579
393848462116,0.58149887471868,0.583730307576894,0.585924231057764,0.58836346586
6467,0.590940360090022,0.593503750937734,0.596163915978995,0.598831582895724,0.6
01498499624906,0.604049512378095,0.606608777194299,0.608884846211553,0.61131170
2925731,0.613753563390848,0.616018004501125,0.618082145536384,0.620155288822206,
0.622325206301575,0.624370967741935,0.626154913728432,0.62796511627907,0.6295911
47786947,0.631154538634659,0.632640660165041,0.634048387096774,0.63552963240810
2,0.636900225056264,0.638386346586647,0.639740060015004,0.640933983495874,0.6422
36309077269,0.643089647411853,0.644031132783196,0.644859339834959,0.64599437359
3398,0.647079894973743,0.64792160540135,0.648774568642161,0.649686046511628,0.65
0425731432858,0.651052138034509,0.65162528132033,0.652349587396849,0.6529257314
32858,0.653567891972993,0.654358964741185,0.654750937734434,0.655415603900975,0.
656086271567892,0.656445236309077,0.656969242310578,0.657401725431358,0.6578927
23180795,0.658273818454614,0.658721305326332,0.658994748687172,0.65938709677419
3,0.659689422355589,0.660079144786197,0.660459489872468,0.660805326331583,0.6611
33533383346,0.661512378094524,0.661756189047262,0.662122655663916,0.66260165041
2603,0.662802700675169,0.663187546886722,0.663306451612903,0.663717554388597,0.6
63926106526632,0.664150037509377,0.664345086271568,0.664456114028507,0.66478057
0142536,0.665092273068267,0.665256189047262,0.665494373593398,0.665527006751688,
0.665969242310578];time=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,
25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,
55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,
85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100];

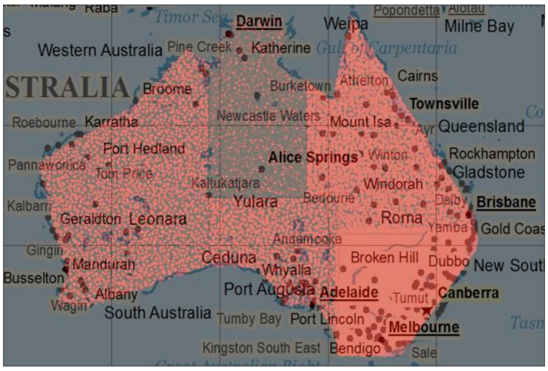

Raba
Timor Sea
Darwin
Weipa
Popondetta
Alotau
Milne Bay
Western Australia
Pine Creek
Katherine
Gulf of Carpentaria
STRALIA
Broome
Burketown
Atherton
Cairns
Townsville
Roebourne
Karratha
Newcastle Waters
Mount Isa
Ayr
Queensland
Port Hedland
Alice Springs
Winton
Rockhampton
Pannawonica
Tom Price
Gladstone
Kaltukatjara
Windorah
Yulara
Bedourie
Dalby
Brisbane
Kalbarri
Geraldton
Leonara
Roma
Yamba
Gold Coas
Andamooka
Gingin
Ceduna
Broken Hill
Dubbo
New Sout
Mandurah
Whyalla
Busselton
Port Augusta
Adelaide
Tumut
Canberra
Albany
Wagin
South Australia
Tumby Bay
Port Lincoln
Melbourne
Kingston South East
Bendigo
Sale


Raba
Timor Sea
Darwin
Weipa
Popondetta
Alotau
Milne Bay
Western Australia
Pine Creek
Katherine
Gulf of Carpentaria
STRALIA
Broome
Burketown
Atherton
Cairns
Townsville
Roebourne
Karratha
Newcastle Waters
Mount Isa
Ayr
Queensland
Port Hedland
Alice Springs
Winton
Rockhampton
Pannawonica
Tom Price
Gladstone
Kaltukatjara
Windorah
Yulara
Bedourie
Dalby
Brisbane
Kalbarri
Geraldton
Leonara
Roma
Yamba
Gold Coas
Andamooka
Gingin
Ceduna
Broken Hill
Dubbo
New Sout
Mandurah
Whyalla
Busselton
Port Augusta
Adelaide
Tumut
Canberra
Albany
Wagin
South Australia
Tumby Bay
Port Lincoln
Melbourne
Kingston South East
Bendigo
Sale


Raba
Timor Sea
Darwin
Weipa
Popondetta
Alotau
Milne Bay
Western Australia
Pine Creek
Katherine
Gulf of Carpentaria
STRALIA
Broome
Burketown
Atherton
Cairns
Townsville
Roebourne
Karratha
Newcastle Waters
Mount Isa
Ayr
Queensland
Port Hedland
Alice Springs
Winton
Rockhampton
Pannawonica
Tom Price
Gladstone
Kaltukatjara
Windorah
Yulara
Bedourie
Dalby
Brisbane
Kalbarri
Geraldton
Leonara
Roma
Yamba
Gold Coas
Andamooka
Gingin
Ceduna
Broken Hill
Dubbo
New Sout
Mandurah
Whyalla
Busselton
Port Augusta
Adelaide
Tumut
Canberra
Albany
Wagin
South Australia
Tumby Bay
Port Lincoln
Melbourne
Kingston South East
Bendigo
Sale


Raba
Timor Sea
Darwin
Weipa
Popondetta
Alotau
Milne Bay
Western Australia
Pine Creek
Katherine
Gulf of Carpentaria
STRALIA
Broome
Burketown
Atherton
Cairns
Townsville
Roebourne
Karratha
Newcastle Waters
Mount Isa
Ayr
Queensland
Port Hedland
Alice Springs
Winton
Rockhampton
Pannawonica
Tom Price
Gladstone
Kaltukatjara
Windorah
Yulara
Bedourie
Dalby
Brisbane
Kalbarri
Geraldton
Leonara
Roma
Yamba
Gold Coas
Andamooka
Gingin
Ceduna
Broken Hill
Dubbo
New Sout
Mandurah
Whyalla
Busselton
Port Augusta
Adelaide
Tumut
Canberra
Albany
Wagin
South Australia
Tumby Bay
Port Lincoln
Melbourne
Kingston South East
Bendigo
Sale


Raba
Timor Sea
Darwin
Weipa
Popondetta
Alotau
Milne Bay
Western Australia
Pine Creek
Katherine
Gulf of Carpentaria
STRALIA
Broome
Burketown
Atherton
Cairns
Townsville
Roebourne
Karratha
Newcastle Waters
Mount Isa
Ayr
Queensland
Port Hedland
Alice Springs
Winton
Rockhampton
Pannawonica
Tom Price
Gladstone
Kaltukatjara
Windorah
Yulara
Bedourie
Dalby
Brisbane
Kalbarri
Geraldton
Leonara
Roma
Yamba
Gold Coas
Andamooka
Gingin
Ceduna
Broken Hill
Dubbo
New Sout
Mandurah
Whyalla
Busselton
Port Augusta
Adelaide
Tumut
Canberra
Albany
Wagin
South Australia
Tumby Bay
Port Lincoln
Melbourne
Kingston South East
Bendigo
Sale


Raba
Timor Sea
Darwin
Weipa
Popondetta
Alotau
Milne Bay
Western Australia
Pine Creek
Katherine
Gulf of Carpentaria
STRALIA
Broome
Burketown
Atherton
Cairns
Townsville
Roebourne
Karratha
Newcastle Waters
Mount Isa
Ayr
Queensland
Port Hedland
Alice Springs
Winton
Rockhampton
Pannawonica
Tom Price
Gladstone
Kaltukatjara
Windorah
Yulara
Bedourie
Dalby
Brisbane
Kalbarri
Geraldton
Leonara
Roma
Yamba
Gold Coas
Andamooka
Gingin
Ceduna
Broken Hill
Dubbo
New Sout
Mandurah
Whyalla
Busselton
Port Augusta
Adelaide
Tumut
Canberra
Albany
Wagin
South Australia
Tumby Bay
Port Lincoln
Melbourne
Kingston South East
Bendigo
Sale