International Journal of Hybrid Information Technology
Vol. 10, No. 7 (2017), pp.11-22
http//dx.doi.org/10.14257/ijhit.2017.10.7.02

A Tile Data Cache Replacement Policy Based on Hierarchical Relationship and Correlation Degree

XingHan Chen[1],Feixiang Chen[2] and Jiaxing Liu[3]

[1first author] *Department of Information Science and Technology, Beijing Forestry University Beijing 100083, China, fishxhchen@126.com*

[*2Corresponding Author] *Department of Information Science and Technology, Beijing Forestry University Beijing 100083, China, fxchen@126.com*

[3] *Department of Information Science and Technology, Beijing Forestry University Beijing 100083, China, liujiaxing_tao@163.com*

Abstract

To solve the problems that server and network pressure is too large and tile response time is too long in tile spatial data transmission, a data cache preplacement method named Value Evaluation of Associated Tiles was put forward in this paper. This method combined the tiles of hierarchy and the correlation between adjacent tiles, and the value of tiles was defined. On the basis of this method, the tile value and cache space were abstracted as 0/1 knapsack problem, and solved by ant colony algorithm. Experimental results showed that this method was significantly improved in tile hit rate and byte hit rate, especially in the small cache capacity, the cache hit rate was significantly higher than other algorithms.

Keywords: *tile data; hierarchical relationship; correlation degree; cache replacement policy; Ant colony algorithm*

1. Introduction

The development and integration between network technology with spatial information services bring geospatial information service network develops rapidly. It affects the quality of network spatial geographic information service that traffic surge due to sustained growth in the scale of users and mass properties of spatial data bring network congestion and server overload [1]. According to the users' requirement, Network spatial geographic information service based on tile granularity employ spatial data streaming transmission technology to transmit multiresolution tile data to users [2-5]. This on-demand service can avoid unnecessary spatial data transmission which effectively reduces the network bandwidth pressure and alleviates the pressure on server. Therefore, establishing a reasonable caching mechanism in the client side eases server pressure and reduces system requirements for network bandwidth and server performance.

Many scholars have done some researches on tile-pyramid data cache mechanism at the client side [6-8, 15-16]. Tile cache lifetime excess and popularity replacement (TCLEPR) policy was proposed in reference [16], in which tiles that life span is beyond average cache life and access popularity is lowest are swap out. Based on tile granularity, TCLEPR brings higher cache-hit rate while it does not take into account tiles spatial relationship and size differences. Reference [7] mainly reported the tiles prefetching policy and replacement policy. It calculated neighbor tiles' prefetching probability by the tile's transition probability, and the tile with least transition probability was replaced by the tile with biggest prefetching probability. However, only the neighbor relation has been investigated and it is not been involved that the effect of hierarchy factors on the probability of tiles being called. Meanwhile this method computation amount is too large. In reference [8], multi-core location aware cache replacement policy (MLCR) was proposed which is based on the history request number of tiles in the multicore GPU shared cache group to improve tile cache hit rate, while it is only aimed at the 3D scene in mobile devices.

Users have their usage patterns when they use map services. For example, they tend to use spatial data that can provide maximum comfort and spatial data requested by single user tends to more centralized in space and more repeatable. In the network spatial geographic information service based on tile granularity, different resolution level is an important factor to affect the clarity and comfort of spatial data. On the client side, magnifying, shrinking, dragging are the most commonly used operating. Zooming into or zooming out of a map is in essence to call different resolution level data in the same geographic range; dragging is in essence to call neighbor tiles of the visual areas tiles.

In this paper, we present a tile data cache replacement policy, Value Evaluation of Associated Tiles (VEAT), which is based on multi resolution hierarchical relations between tile data and relations between adjacent tiles. VEAT fully takes into account the impact on the user experience when to measure tiles storage value. Tile value is adjusted by level factor which is calculated according to the tile request history. Tile's associated value is calculated by the value of neighbor tile. By tiles' associated value and sizes, tile data cache replacement is abstracted into 0/1 knapsack problem which is settled by ant colony algorithm.

2. Tile Cache Index Design

The tiles stored in the tile-based server of network geographical information service are generated from hierarchical cutting map data in spatial database (such as setting the size of tiles as 256×256 pixel) [9-10]. These tiles constitute the multi resolution hierarchical spatial data tile Pyramid model, as shown in Figure 1 In the multi resolution layer pyramid model of tiles, the topper the level is, the bigger the level is and the higher the resolution is. At different levels, the geographical range represented by all tiles on the same level is same. When users request tiles, the client can send a request to the server for a couple of tiles at the corresponding resolution level based on the field of view and resolution. One specific tile through two-dimensional coordinates and level number obtains unique identification.

When users request tiles, according to the user's current window angle center and the window boundary latitude and longitude range to determine the required tiles' resolution level (***level***) and two dimensional coordinate range (***{Xmin***,***Ymin}*** to ***{Xmax***,***Ymax}***). On the client side, cache index and tile data are stored separately. The tile cache index can uniquely determine if a tile is present in the cache and where the tile is stored. When building tile index, the hierarchy, row and column of tile data are encoded into a string (***TileID***). Each tile data has a unique ***TileID*** which can uniquely identify a tile map block:

$$\boldsymbol{TileID} = (\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{level}) \qquad (1)$$

Where ***level*** represents tile hierarchy, ***i*** represents tile's line number in the ***level*** hierarchy, ***j*** represents tile's column number in the ***level*** hierarchy. ***TileID(i,j,level)*** represents the encoding of tile whose hierarchy is ***level***, line number is ***i*** and column number is ***j***. To quickly locate the index entry, index entry uses hash storage.

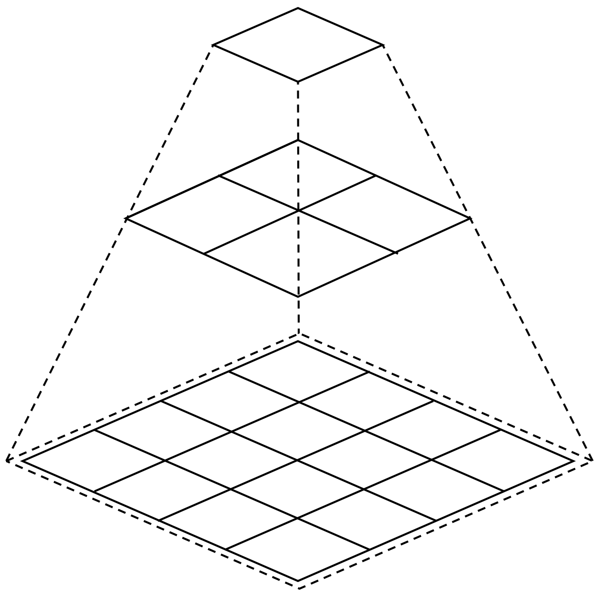

Figure 1. Tile Pyramid Model

Assume index term is ***ind*** and ***TileID*** is the keyword for ***ind***. ***TileID*** is the unique identification of ***ind***. Assume ***index*** is an index set for storing data in the cache:

$$\forall\, \boldsymbol{ind} \in \boldsymbol{Index}\,, \boldsymbol{ind} = \left(\boldsymbol{TileID}, \boldsymbol{size}, \boldsymbol{frequency}, \boldsymbol{T_{last}}, \boldsymbol{T_{first}}, \boldsymbol{level}, \boldsymbol{value}\right) \quad (2)$$

ind(TileID) ϵ ***Index*** represents index entry ***ind*** whose key is ***TileID*** in index set ***Index***. ***size*** represents space occupied by the tile. ***T_first*** represents the first time the tile is cached and ***T_last*** represents the last time the tile is hit. ***frequency*** represents hit counts of the tile after ***T_first***. In order to avoid the sudden increase in frequency when tile data is requested multiple times in short time which impact tile value evaluation, cache protection policy is set for tiles cached. Static time is defined with ***T_least***. A tile's ***frequency*** only increased by 1 if the tile is hit repeatedly within time ***T_least***. ***value*** represents value of the cache entry which is evaluated by algorithm in each cache replacement.

3. VEAT Strategy

3.1 Determination of Cache Update Value of VEAT Algorithm

When new tile enters and the cache space cannot contain it, tiles of low cache value in the cache area should be replaced through cache replacement strategy. The traditional cache replacement strategies are as follows [11-12]. In LRU (Least Recently Used) strategy, every tile's last hit time is recorded and tiles in the cache area that have not been hit for the longest duration are replaced and excluded from the cache area. In LFU (Least Frequently Used) strategy, the times of tiles being hit are recorded and tiles with the fewest times within a specific period are replaced and excluded from the cache area. In FIFO (First In First Out) strategy, tiles that first enter the cache are replaced and excluded from the cache area. In SIZE algorithm, the tile with the greatest performance indicators or a couple of tiles with relatively greater indicators are replaced and excluded from the cache area.

VEAT algorithm uses cache update value to replace cache. Firstly, the evaluation model of cache value is determined, in which value of each tile is evaluated in order to maximize the total cache value of all tiles stored in the cache. The request intensity of tiles is closely related to the visit frequency determined by times of history visits of tiles ***Fre(TileID)*** and survival time in cache ***T_present-T_first (TileID)***. Taking times of being hit in history and survival time into account, the cache value of one tile is determined as follows:

$$\begin{cases} \boldsymbol{H_{level}(i,j)} = \left(\frac{\boldsymbol{Fre(TileID)}}{\boldsymbol{T_{present} - T_{first}(TileID)}}\right)^{\boldsymbol{\sigma}} \\ \boldsymbol{TileID = (level, i, j)} \end{cases} \quad (3)$$

where $\boldsymbol{H_{level}(i,j)}$ is the cache value of the tile in ***i*** line , ***j*** row and ***level*** hierarchy, $\boldsymbol{T_{present}}$ is the time of the system at present, $\boldsymbol{T_{first}(TileID)}$ is the time the tile first enters cache, and ***Fre(TileID)*** is the visit frequency during its cache time, and $\boldsymbol{\delta}$ is the accommodation coefficient of the equation and is fitted by experimental data.

Different solution and geographical ranges of multi-solution tiles affect the elaborate degree and comfort degree of tile map, which makes multi-solution tiles call frequency be variation in different levels. In network spatial geographical information service, some users prefer some specific levels' tiles data. As a result, when cache value is the same determined by other factors, users are estimated to use tiles that can bring them more comfort. The comfort degree function is set as **M(*level*)** which indicates the comfort degree in ***level*** hierarchy:

$$\mathbf{M(level)} = \boldsymbol{\beta} \times \boldsymbol{C(level)} \quad (4)$$

C(level) is the level factor, suggesting the visit rate in history of ***level***. Users prefer data that bring them maximum comfort when visiting tile data with different resolution levels and identical geographical ranges. Consequently, level factor ***C(level)*** can be regarded as positively correlated to comfort factor ***M(level)***. Level factor can be calculated as follows:

$$\boldsymbol{C(level)} = \frac{\boldsymbol{S(level)}}{\sum_{\boldsymbol{l=0}}^{\boldsymbol{num}} \boldsymbol{S(l)}} = \frac{\boldsymbol{S(level)}}{\boldsymbol{SUM}} \quad (5)$$

where ***S(level)*** refers to the total number of tiles in ***level*** hierarchy in the cache, and ***num*** refers to total number of levels. Therefore, $\sum_{\boldsymbol{l=0}}^{\boldsymbol{num}} \boldsymbol{S(l)}$ represents the total number of tiles in all resolution levels in cache, which means that the total number of tiles in the cache is represented by ***SUM***.

Introducing comfort degree function ***M*** to the evaluation model of cache:

$$\begin{cases} \boldsymbol{H_{level}(i,j)} = \boldsymbol{\alpha} \times \left(\frac{\boldsymbol{Fre(TileID)}}{\boldsymbol{T_{present} - T_{first}(TileID)}}\right)^{\boldsymbol{\sigma}} \\ \quad + \boldsymbol{\beta} \times \boldsymbol{M(level)}^{\boldsymbol{\gamma}} \\ \boldsymbol{TileID = (level, i, j)} \end{cases}$$

(6)

where $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, and $\boldsymbol{\gamma}$ are accommodation coefficients of the equation and fitted by experimental data. And $\boldsymbol{\alpha}$+$\boldsymbol{\beta}$=1.

When a tile is requested, tiles that have same resolution level with requested tile and the space distance from the tile within limits are called association tiles. Windows used by users are usually rectangle. It is hypothesized that window is made up of ***row*column*** tiles. When tiles data need to be loaded into the windows, ***row*column*** tiles whose spatial location is proximate in the same resolution level should be loaded. When one tile is requested, tiles in the same resolution level and proximate two-dimensional coordinates are very possible to be requested simultaneously. After one tile is requested, when users carry out translation operation of maps, tiles close to tiles requested are also very likely to be requested due. Therefore, cache value of tiles is influenced by cache value of relation tiles in addition to inherent properties.

Since the wide range of association tiles with the specific tiles and difficult to specify, calculation of all association tiles brings too much complexity. As a response, this paper selects representative tiles of four directions as relation tiles. ***T(level,i,j)*** refers to the tile of ***i*** line and ***j*** row of ***level*** hierarchy. If ***T(level,i,j)*** is the center of the window which suggests the tiles in window rectangle area ranging from ***T(level, i-row+1, j-column+1)*** to ***T(level,i+row-1,j+column-1)***. At the same moment, the likelihood of users carrying out translation operation in up, down, left, right directions is regarded as identical. Tiles selected in the four directions are ***T(level,i-(row-1)/2,j-(column-1)/2)***, ***T(level ,i-(row-1)/2,***

j+(column-1)/2), ***T(level, i+(row-1)/2,j-(column-1)/2)***, and ***T(level, i+(row-1)/2,j+(column-1)/2)*** respectively, in which "/"suggests exact division.

Introducing association tiles to the evaluation of cache value:

$$\boldsymbol{RH_{level}(i,j) = H_{level}(i,j) * R_1 + \left(H_{level}\left(i-\frac{row-1}{2}, j-\frac{column-1}{2}\right) + H_{level}\left(i-\frac{row-1}{2}, j+\frac{column-1}{2}\right) + H_{level}\left(i+\frac{row-1}{2}, j-\frac{column-1}{2}\right) + H_{level}\left(i+\frac{row-1}{2}, j+\frac{column-1}{2}\right)\right) * R_2} \quad (7)$$

where $\boldsymbol{RH_{level}(i,j)}$ is the association cache value of tile ***T(level,i,j)***, $\boldsymbol{R_1}$、$\boldsymbol{R_2}$ are regulatory factors of the equation and fitted by experimental data, and $\boldsymbol{R_1}$ $+\boldsymbol{R_2}$=1. When tiles of cache are replaced, association value of tiles is regarded as evidence of replacement.

3.2 Knapsack Problem Abstraction with the VEAT Strategy

Due to limited cache space, when cache value of one tile is considered, the space that the tile takes up should also be taken into account. This question can be otherwise solved by 0/1 knapsack problem[13]. The maximum bearing capacity of a knapsack is $\boldsymbol{W}$, $\boldsymbol{n}$ items are provided, the weight of item $\boldsymbol{i}$ is $\boldsymbol{w_i}$, and how to select items to maximize the total value without exceeding the maximum bearing capacity should be solved.

If the total number of tile Indexes in the cache is ***N***, ***TileID(z)*** represents the ***TileID*** of the ***z*** index, the size of the tile corresponding to ***TileID*** is ***Size(z)***, and the association cache value of the tile is ***RH(z)***. The cache space can be abstracted to the size of the knapsack which is ***V***. The state variable of the tile is ***State(z)***. When tiles are selected into the cache, ***State(z)*** is 1, otherwise it is 0. The total value of all the tiles of the cache is $\sum_{z=1}^{N}\boldsymbol{RH(z)State(z)}$, and the total space they take up is$\sum_{z=1}^{N}\boldsymbol{Size(z)State(z)}$. The solution to 0/1 knapsack problem maximizes the total cache value $\sum_{z=1}^{N}\boldsymbol{RH(z)State(z)}$ through determining the value of ***State(z)(z=1,2,3,…,N)***.

$$\begin{cases} \boldsymbol{Max} \sum_{z=1}^{N}\boldsymbol{RH(z)State(z)} \\ \boldsymbol{State(z) \in \{0,1\}} \quad \boldsymbol{z \in \{1,2,3,\ldots,N\}} \\ \sum_{z=1}^{N}\boldsymbol{Size(z)State(z) \le V} \end{cases} \quad (8)$$

3.3 Ant Colony Algorithm Solution with the VEAT Method

To enlarge the total value of cache as much as possible, and to avoid the convergence to a locally optimal solution, ant colony algorithm, a member of the heuristic algorithm family, is applied to solve computational problems. Ant colony algorithm is a simulated evolutionary algorithm which is proposed based on the behavior of ants in the natural world [14]. While passing a trail, ants would lay down pheromone to send information to other ants, the existence and strength of which can be sensed within limits. The pheromone density will become higher with the increase in ant number, and over time, the pheromone will evaporate.

Ants would leave such pheromone on something, tiles in this case, and once choices are completed, they would find out a set of, or sets of, optimum solutions, whose tiles would then be left with more pheromone than others. Pheromone, therefore, can be updated as follows when ant colony algorithm is applied to solve the 0/1 knapsack problem:

$$\boldsymbol{T_z(t+p) = (1-\rho)T_z(t) + \triangle T_z} \quad (9)$$

$\boldsymbol{T_z(t)}$ is the amount of pheromone on tile $\boldsymbol{Z}$ at $\boldsymbol{t}$. ***ρ ∈(0,1)*** is the pheromone evaporation coefficient, ***(1-ρ)*** is the pheromone residual coefficient, and $\triangle\boldsymbol{T_z}$ is the amount of pheromone added within $\boldsymbol{p}$. The added amount of pheromone $\triangle\boldsymbol{T_k}$ can be represented as:

$$\triangle \mathrm{T_z} = \begin{cases} \boldsymbol{\rho T_z(t) + 0.1}, \textbf{z is the optimal solution of the current round} \\ -\frac{\mathbf{Size(z)}}{\mathbf{V}} \boldsymbol{\rho T_z(t)}, \textbf{z is not the optimal solution of the current round} \end{cases} \tag{10}$$

The influence of tile size on pheromone is considered in this formula. The bigger the tile, the more cache storage space it occupies; under the condition that nothing is hit, a tile that occupies more cache space has higher pheromone evaporation coefficient.

In the ant colony algorithm, the probability of each tile being selected by ants is called selection probability. The probability of ant $\boldsymbol{k}$ selecting tile $\boldsymbol{z}$ at $\boldsymbol{t}$ is:

$$\boldsymbol{P_z^k(t)} = \begin{cases} \frac{T_z^{\mu}(t)\eta_z^{\upsilon}(t)}{\sum_{m \notin tabu_k} T_m^{\mu}(t)\eta_m^{\upsilon}(t)}, z \notin tabu_k \\ 0,\ others \end{cases} \tag{11}$$

$\boldsymbol{tabu_k}$ is a tabu list, in which tile that has been judged by ant $\boldsymbol{k}$ is deposited and cannot be selected any more. $\boldsymbol{\eta_z(t)}$ is a heuristic function, representing the desirability of ant $\boldsymbol{k}$'s independent choice of tile $\boldsymbol{z}$. $\boldsymbol{\mu}$ is a pheromone factor representing the influence of pheromone on ants' selection results, and $\boldsymbol{\upsilon}$ is a desirability heuristic factor representing the influence of cache content property on ants' selection results. The bigger the value of $\boldsymbol{\mu}$, the more likely for later ants to follow the former ones selections; the bigger the value of $\boldsymbol{\upsilon}$, the more likely for ants to use heuristic function to make independent selection.

The heuristic function $\boldsymbol{\eta_z(t)}$ represents the judgement made by ants according to the value of cache tile without the influence of pheromone. Based on the associated cache value of tile and the size of tile, the heuristic function can be defined as:

$$\boldsymbol{\eta_z(t)} = \frac{\boldsymbol{RH(z)}}{\boldsymbol{Size(z)}} \tag{12}$$

3.4 VEAT Algorithm Procedure

When a new tile needs to gain access to the memory cache or local cache, and the cache space is insufficient, cache replacement of contents can be carried out based on the VEAT algorithm, so that the total value of data stored in the cache can be maximized and the hit rate of tile can be increased. Since the tiles called in the client windows are consecutive, requests for multiple consecutive tiles will generally be sent. If $\boldsymbol{N}$ new tiles are requested at the same time, the cache replacement algorithm needs to be executed for $\boldsymbol{N}$ times. To improve the algorithm performance, this paper carries out an integrated processing of those multiple tiles that are requested consecutively, together with only one cache replacement.

Descriptions of the VEAT Algorithm:

1）When a new tile needs to gain access to cache, cache objects will be created and initialized based on the attribute information of this new tile, such as level, line number, column number and size. ***TilesID*** should be determined according to the level, line number and column number of the tile, the frequency of cache should be initialized as 1, and the built-up time of cache should be recorded.

2）Whether the cache space can be used to store this tile? If the cache space is insufficient, then please go to step 3; if not, store this tile.

3）Initialize the parameters of the ant colony algorithm. The pheromone on each tile $\boldsymbol{T_z(t)}$ should be initialized as a fixed value $\boldsymbol{T_0}$; the maximum iteration number of this algorithm should be set as ***TMAX*** and the number of ants should be set as $\boldsymbol{N_{ant}}$. The tabu list of all ants $\boldsymbol{tabu_k}$ should be initialized as null, and the solution set list of each ant $\boldsymbol{result_k}$ should also be initialized. Such parameters as $\boldsymbol{\delta}$、$\boldsymbol{\alpha}$、$\boldsymbol{\beta}$、$\boldsymbol{\gamma}$、$\boldsymbol{R_1}$、$\boldsymbol{R_2}$、$\boldsymbol{\upsilon}$、$\boldsymbol{\mu}$ should be given a value.

4）According to formula (11), ant $\boldsymbol{k}$ calculates the probability of every tile being visited and gets $\boldsymbol{P_z^k(t)}$. Based on the probability value, ant $\boldsymbol{k}$ then chooses the next tile $\boldsymbol{z}$ to be stored into cache and adds the tile $\boldsymbol{z}$ to the tabu list $\boldsymbol{tabu_k}$. Try to put tile $\boldsymbol{z}$ to the solution set list $\mathbf{result_k}$. If the total volume of tiles in the solution set list is smaller than cache capacity $\boldsymbol{V}$, then add tile $\boldsymbol{z}$ to this solution set list by changing the ***State (z)*** into 1; if not, new judgement should be made until all remaining tiles cannot be added to the solution set list any more. Cycle this process until all ants obtain the solution set.

5）Find out the solution set that can maximize the value of $\sum_{z=1}^{N} \boldsymbol{RH(z)State(z)}$ in the solution sets of all ants. If the total cache value of this solution set is bigger than that of the global optimum solution, then replace the current optimum solution. If the iteration number is smaller than $\boldsymbol{TMAX}$, then update pheromone and go to step 4); if not, go to step 6).

6）If the newly arrived data is in the optimum solution, then store it into cache and replace the data that is not in the optimum solution with the newly arrived data; if not, do not store the newly arrived data. The VEAT algorithm execution ends.

4. Experiment Evaluation

4.1 Experiment Environment

This experiment applies Fiddler to collect the request log data of users who do not set cache, and 213,200 records are collected, with a storage size of 1.747 G. The resolution of tiles is 256×256, the biggest tile occupies 26.6 kb, and the smallest one occupies 197 b. SimpleScalar is used as the simulation platform in the experiment. The experiment environment is Lenovo T440，CPU 1.60GHz，RAM 4G. First, the log data of users are processed to extract the level, line number, column number and size of tiles, and request time. Based on these data, the process of calling tiles on the client will then be simulated. In this process, each tile will be evaluated with the VEAT algorithm to turn the relation between cache space and cache tiles into the 0/1 knapsack problem; if the cache space is insufficient, then apply the ant colony algorithm to carry out a cache replacement.

Parameter settings in the VEAT algorithm are as follows:

1）In the experiment, the values of $\boldsymbol{\sigma}$ and $\boldsymbol{\gamma}$ are fitted, and when 0.8 is chosen for $\boldsymbol{\sigma}$ and 0.6 for $\boldsymbol{\gamma}$, optimal results can be achieved.

2）$\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ stand for the influence ratio of the factor of tile level. When 0.8 is chosen for $\boldsymbol{\alpha}$, the average hit rates of cache are optimal, so 0.8 is chosen for $\boldsymbol{\alpha}$ and 0.2 for $\boldsymbol{\beta}$.

3）The values of $\boldsymbol{R_1}$ and $\boldsymbol{R_2}$ stand for how much influence associated tiles can exert on the associated value of tiles. 0.9 is chosen for $\boldsymbol{R_1}$ and 0.1 for $\boldsymbol{R_2}$.

4）In the ant colony algorithm, 0.4 is chosen for parameter $\boldsymbol{\mu}$, and 0.6 for parameter $\boldsymbol{\upsilon}$.

4.2 Experiment Results and Analysis

The processing results of users' request log data are set as data set A, data set B, data set C and data set D. FIFO, LFU, LRU and VEAT are applied respectively to each set, to simulate the calling of tiles under the condition of different cache capacities, and to calculate their byte hit rates and tile hit rates. Figure 2 and Figure 3 show the byte hit rates and tile hit rates of the four algorithms applied under the condition that the cache sizes are 10M, 20M, 30M, 40M, 50M, and 60M.

The experiment results in Figure 2 and Figure 3 indicate that both byte hit rates and tile hit rates of the four algorithms will become higher with the increase in cache size. When the cache capacity is enlarged to a certain point, however, the increase of cache hit rates will level off, and the cache hit rates of the four algorithms get close to each other.

Among the four algorithms, FIFO has the lowest cache hit rates in most cases, with values far smaller than those of the other algorithms. LRU is better than FIFO, which considers making adjustment to cache items after cache hits. LFU is, however, better than LRU, which can avoid the influence of periodical and accidental events on LRU hit rates. VEAT has cache hit rates higher than the other three algorithms do, especially so or more so when the cache capacity is low. VEAT algorithm not only considers such factors as the time and the space tiles occupy, but also the influence of such spatial factors as tile level relations and associated tiles on the hit rates; and besides, this algorithm also considers as much valuable information as possible that is related to tile data. Optimal results, therefore, are achieved for different data sets under the condition of different cache capacities. In general, the order from high to low of the cache hit rates of the four algorithms is VEAT, LFU, LRU and FIFO.

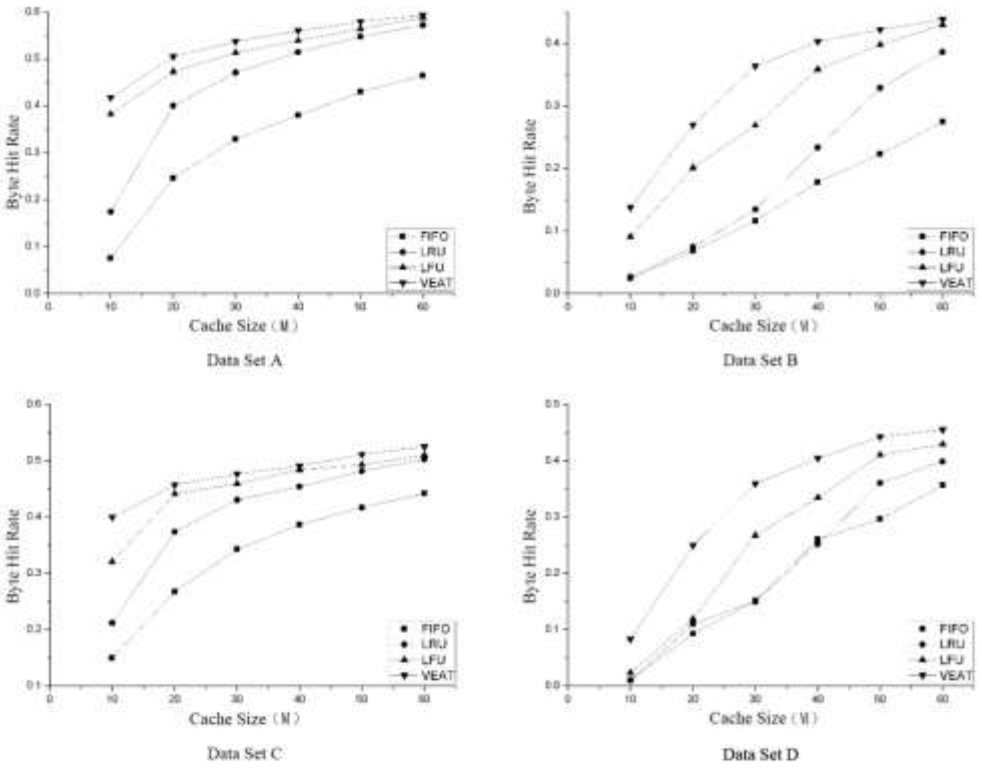

Figure 2. Byte Hit Rate

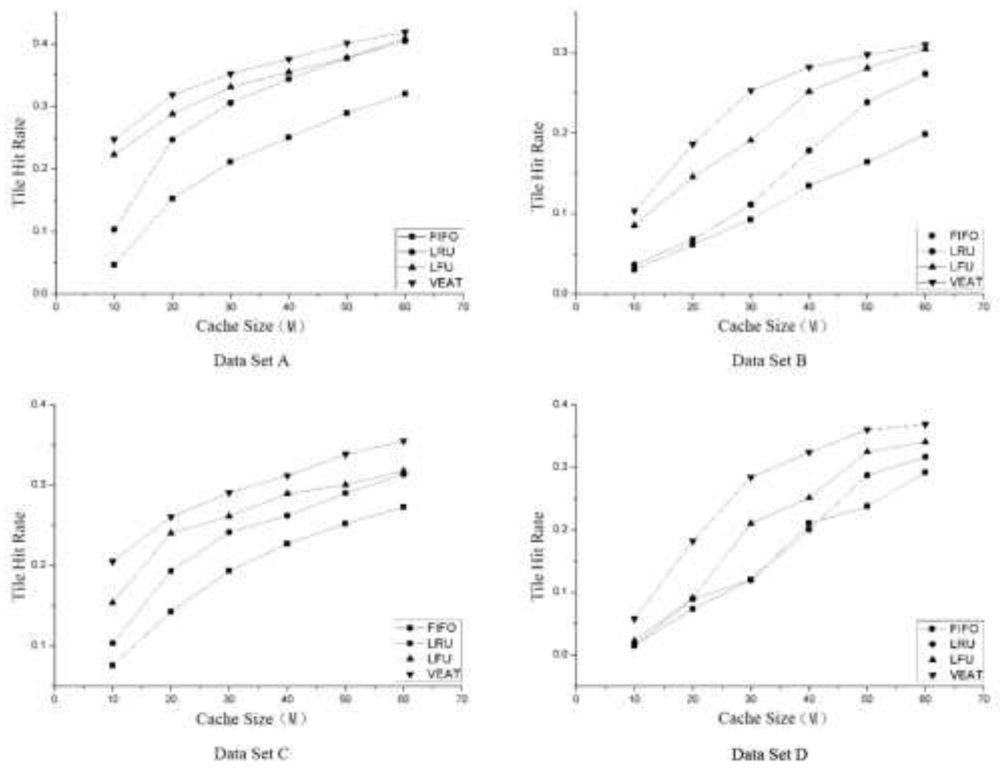

Figure 3. Tile Hit Rate

This experiment shows the results of byte hit rates and tile hit rates of the four algorithms. In addition, another experiment is carried out to explore the cache hit rates of the four algorithms at every stage in the data reading process. For the limit of space, this paper only mentions the simulation results of the byte hit rates and request hit rates in the recording process in the data set B, achieved by the four algorithms under the condition of 20M cache capacity, as shown in Figure 4.

Figure 4 indicates the variation of cache hit rates in the recording process in the data set B. The experiment results suggest that the cache hit rates of the four algorithms finally level off after they fluctuate for a period of time. VEAT algorithm gains an edge in cache hit rates when 10% to 20% of data is read, indicating that this algorithm can adjust the value of tiles based on a small number of tile requests to obtain relatively high cache hit rates.

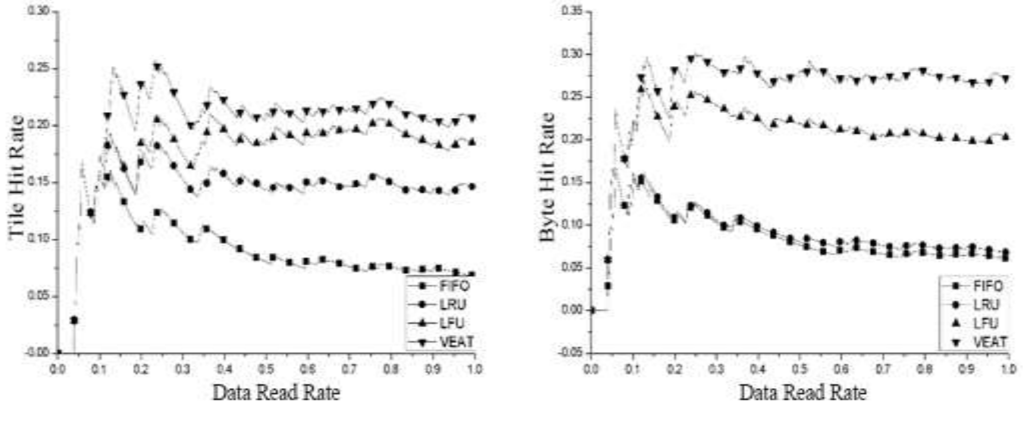

Figure 4. Trends of Cache Hit Rate of Data Set B

5. Conclusion and Discussion

In this paper, we design a tile cache index, and propose a cache replacement policy named VEAT for tile which combines the level of relationship and tile correlation degree. In VEAT, cache capacity and tiles' associated value are abstracted as 0/1 knapsack problem and solved with ant colony algorithm.

The results show that VEAT algorithm is superior to LFU algorithm in byte hit rate and tile hit rate. And, VEAT algorithm combined with multiple factors can still play its advantage to get a higher hit rate while FIFO, LRU, LFU algorithms are limited when the cache space is limited or the amount of data is less.

In the experimental results of byte hit rate and tile hit rate, the byte hit rate is higher than the tile hit rate with the same data set and the same cache capacity. For this result, this paper makes a hypothesis that users tend to access tiles that contain large amounts of information. In the next step, we will focus on the influence of tile size, location, and the level of tiles on the tile value.

Acknowledgments

This work was supported by Fundamental Research Funds for the Central Universities (TD2014-02).

References

[1] W. Hao, Y.U. Zhanwu, Z. Wu, "The Research on the Algorithm of Spatial Data Cache in Network Geographic Information Service", Acta Geodaetica Et Cartographica Sinica, vol. 38, no. 4, **(2009)**, pp. 348-355.

[2] H. Xia, L. Meng. "Research on Spatial Image Streaming Model Based on BitTorrent", Acta Geodaetica Et Cartographica Sinica, vol. 42, no. 2, **(2013)**, pp. 225-232.

[3] R. Li, X. Tang, X. Shi, J. Fan, Z. Gui, "A replication strategy based on optimal load balancing for a heterogeneous distributed caching system in networked GISs", Geomatics & Information Science of Wuhan University, vol. 40, no. 10, **(2015)**, pp. 1287-1293.

[4] M.H. Jeong, Y.C. Suh, "A Study on Tile Map Service of High Spatial Resolution Image Using Open Source GIS", Journal of Korean Society for GeoSpatial Information System, vol. 17, no. 1, **(2009)**, pp. 167-74.

[5] S. Wei, L. Zhiqing, J. Mengkai, L. Chengming, "Optimized Design and Implementation of the Tile Map Dynamic Caching Middlewware", Bulletin of Surveying and Mapping, vol. 1, **(2014)**, pp. 37-40.

[6] H. Wang, "Massive Spatial Data Cache Replacement Policy Based on Tile Lifetime and Popularity", Geomatics & Information Science of Wuhan University, vol. 34, no. 6, **(2009)**, pp. 667-670.

[7] R. García, E. Verdú, L. M. Regueras, J. P. D. Castro, "A neural network based intelligent system for tile prefetching in web map services", Expert Systems with Applications, vol. 40, no. 10, **(2013)**, pp. 4096-4105.

[8] M.F. Uluat, V. İşler, "Ensemble adaptive tile prefetching using fuzzy logic", International Journal of Geographical Information Science, vol. 30, no.6, **(2016)**, pp. 1117-1136.

[9] L. Yi, "Parallel Batch-Building Remote Sensing Images Tile Pyramid with MapReduce", Geomatics & Information Science of Wuhan University, vol. 38, no.3, **(2013)**, pp. 278-282.

[10] J. Liu, Q. Gan, Y. Zhang, R. Chunlei, "Implementation of fast spatial matching and image fusion algorithm for tile map", Science of Surveying and Mapping, vol.40, no.11, **(2015)**, pp. 85-88.

[11] L. Liu, X. Xiong, "Least Cache Value Replacement Algorithm", Journal of Computer Applications, vol. 33, no.4, **(2013)**, pp. 1018-1022.

[12] J.-L. Wu, Q. Yang, "A Web Cache Replacement Algorithm Based on Collaborative Filtering", Computer Engineering & Science, vol. 37, no.11, **(2015)**, pp. 2128-2133.

[13] K. Nagar, Y. N, Srikant, "Fast and Precise Worst-Case Interference Placement for Shared Cache Analysis", ACM Transactions on Embedded Computing Systems, vol. 15, no.3, **(2016)**, pp. 1-26.

[14] M. Dorigo, V. Maniezzo, A. Colorni, "Ant system: optimization by a colony of cooperating agents", IEEE Transactions on Systems Man & Cybernetics Part B Cybernetics A Publication of the IEEE Systems Man & Cybernetics Society, vol. 26, no.1, **(1996)**, pp. 29-41.

[15] Y.-K. Kang, Ki-Chang Kim, "Probability-Based Tile Pre-fetching and Cache Replacement Algorithms for Web Geographical Information Systems", Advances in Databases and Information Systems, Vilnius, Lithuania, **(2001)** September 25-28.

[16] C.C. Hsiao, S.L. Chu, S.S. Dai, "Efficient rendering and cache replacement mechanisms for hierarchical tiling in mobile GPUs", Global High Tech Congress on Electronics, **(2012)**, pp. 170-175.

Authors

XingHan Chen, he received his Master degree from Beijing Forestry University.His research interests include WebGIS and Mobile GIS.