A Novel Stochastic-Based Algorithm for Terrain Splitting Optimization Problem

Le Hoang Son, Nguyen Dinh Hoa

Abstract— This paper deals with the problem of displaying large Digital Elevation Model data in 3D GIS. Current approaches relate to the splitting algorithms by 2D Polygonal Vector Data such as Particle Swarm Optimization (PSO-TSA) and Genetic Algorithm (GA-TSA). We will, herein, present another method based on stochastic optimization for the considered problem. It also employs some ideas of Wife-Selection scenario and Stick Procedure. The new method allows us to quickly find the optimal saving threshold. The comparison with the state-of-the-art method will be made to verify the efficiency of the proposed method.

Keywords— Digital Elevation Model, Geographic Information Systems, Stochastic Optimization, Terrain Splitting.

I. INTRODUCTION

Terrain Splitting Optimization (TSO) problem was presented by the authors in [4]. Its purpose is to reduce the displaying time of large terrain data, especially in the format of Digital Elevation Model (DEM), in 3D Geographic Information Systems (GIS). Terrain data are used to represent the three-dimensional compositions in the relation with geographic factors. Depending on the resolution, the size of a terrain is varied to express the information attached to that terrain. For example, a 30m DEM terrain has a volume of 280 Megabytes (MB). The smaller the resolution of terrain is, more details are shown, and its size is increased as a result. Indeed, it takes long time to display such a terrain.

Mathematically, this problem is described below.

$$J_1 = \sum_{i=1}^{k} S_i \to \min , \quad (1)$$

$$\begin{cases} |S_i| \le \alpha \times S_{DEM} \\ |S_i - S_j| \le \varepsilon \times S_{DEM} \\ i = \overline{1,k},\ j = \overline{1,k}, i \ne j \end{cases} , \quad (2)$$

where S_i ($i = \overline{1,k}$) is the area of a small terrain in a processor of a computing system . S_{DEM} is the area of terrain. The parameter α is the saving threshold. The last parameter ε is the disparity. Normally, its range falls into (0, 5). In equation (2), the first line states that the memory space in a processor is saved by $(100-\alpha)$ percents. The second one confirms that the difference between the memory spaces in two processors is small.

Several soft computing methods were designed for TSO problem such as SESA [4], PSO-TSA [6] and GA-TSA [6].

They relied on the ideas of using natural evolution combined with heuristic search methods to specify the solutions. These algorithms achieved successful results as shown in the experiments of equivalent articles.

The aim of this note is to investigate another optimization method for this problem. It employs the new ideas of Wife-Selection scenario and Stick Procedure. The proposed method is named as *Stochastic sImulation Test based Terrain Splitting Algorithm* (SIT-TSA) and is compared with the algorithms above to verify the efficiency.

The remainder of this paper is organized as follows. Section 2 presents some related works for this problem. The proposed method is described in Section 3. Experimental results and discussions are given in Section 4. Finally, we will make conclusions and delineate future works in the last section.

II. RELATED WORKS

The authors in [4] introduced a conditional parallel partitioning method so-called SESA for TSO problem. The basic idea of this method is to traverse all partitions dividing n elements into k blocks. For each block, SESA calculates its area and checks the constraints. If a suitable partition is found, the algorithm will stop and output the results. Certainly, to reduce the number of traversed partitions, a pre-processing step based on geometric processing between polygons is carried out to arrange some elements into specific blocks. Additionally, parallel computing is also employed to accelerate the running time.

However, the saving threshold found in SESA is not optimal. Authors [6] presented two algorithms to solve the original problem. *The first one* based on Genetic Algorithm [2] namely GA-TSA employs some ideas of natural evolution, such as inheritance, mutation, selection, and crossover for finding the best saving threshold in a search area. In this algorithm, an individual is a collection of indexes of all polygons that represent for all blocks in a current solution. Then, through a fitness function, all individuals are sorted in the ascending order, and half of them are selected to reproduce a new generation by the mean of Cross Over and Mutation operations. After pre-defined maximal iteration steps, the best generation is found, and the saving threshold can be extracted from it.

The second algorithm in that literature started with an idea of Swarm Optimization, which is considered to be the most suitable strategy among all of Heuristic Optimization. The chosen algorithm to develop is Particle Swarm Optimization (PSO) which was invented by Kennedy et al. [3]. Indeed, the algorithm was named PSO-TSA. The basic idea of this algorithm lies on *Seed Procedure*. Basically, k seeds are evenly distributed in the space. Each seed represents for a number of polygons. If the constraints (2) are not met, PSO algorithm is used to generate a new population until the stopping condition is reached. In the last iteration, the particle holding *gBest* value will be outputted if it satisfies the constraints.

Manuscript received on January, 2013.
Le Hoang Son is with VNU University of Science, Vietnam National University.
Nguyen Dinh Hoa is with VNU Hanoi Information Technology Institute, Vietnam National University.

In the experiments, PSO-TSA obtains a smaller saving threshold than GA-TSA does. However, the running time of PSO-TSA is longer than that one of GA-TSA. Certainly, two algorithms are better than SESA both by the saving threshold and running time. Therefore, PSO-TSA is considered as the state-of-the-art algorithm for TSO problem.

III. THE PROPOSED METHOD

A. *Basic Ideas*

In this section, we present another approach for TSO problem. It is a stochastic, agent - based approach named *Stochastic sImulation Test based Terrain Splitting Algorithm* (SIT-TSA). Keep the problem in mind and temporarily forget about the previous ideas of using PSO and GA, all we need to know is a basic principle that covers all activities of SIT-TSA algorithm. In a simple way, it can be understood as: "*It is supposed to be no satisfied solution with probability* $1-p$ *after a series of failed stochastic simulation tests on various possibilities derived from the original sample*". In the other word, this principle behaves as a similar way to Monte Carlo method - a class of computational algorithms that rely on repeated random sampling to compute their results [1].

Monte Carlo method is widely used in various fields such as in statistical physics, particularly Monte Carlo molecular modeling as an alternative for computational molecular dynamics as well as to compute statistical field theories of simple particle and polymer models. Therefore, a Monte Carlo - like approach is suitable for our problem in case of the number of polygons is large, and an intermediate answer is required. Now, consider the following *Wife-Selection scenario* in Fig. 1: "Once upon a time, there is a great Kingdom with a wise, brave king. Everybody admires him a lot. However, this king has only a son, and he pays attention to nothing except hunting or playing with friends. The old man is very sad because he is getting older, and his son cannot replace him. Sharing the worries, some servants suggest the king to find his son a fiancée. He recognizes it a good idea and immediately announces to all villages in the Kingdom. Besides, the old king assigns a duke to follow with the prince to come to each village for searching. Nevertheless, the prince is very lazy and does not want to go to all villages. The duke, who is a good mathematician, calculates by probability which village has more girls satisfying the king's conditions. However, these villages are still much, and it takes a lot of time to speak with all girls in a specific one. Instead, the duke chooses random girls from each village and tests. If he finds a suitable girl, then she will be added to his lists. The test is repeated in other selected villages. Finally, the duke will give the prince his list and let him choose the fiancée. If no suitable girl is found on the duke's list, the prince has to wait for next selection in some following years".

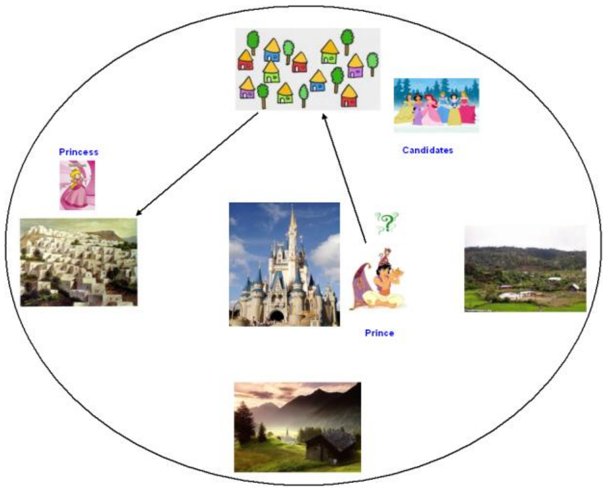


Fig 1. Wife – Selection scenario

SIT-TSA acts like the scenario above. An agent is randomly initiated at a polygon. By using a local search method, an ordered list of polygons is established. Then, by multiple tests with random 'sticks' dividing this list into k blocks, a candidate list of this agent is set up. This process is repeatedly performed for other agents. Finally, the optimal solution with minimal saving threshold parameters will be chosen from all the candidate lists. Certainly, we use parallel computing as an important aid for the reduction of total computing time due to independent works between agents.

B. *The Algorithm*

Input: a terrain, a polygon shape dataset, the total number of polygons (l), the number of processors (k), the disparity ε and a test range $[t_0, t_1]$.

Output: The optimal saving threshold α.

SIT-TSA:

Step 1: Divide the total number of agents following by the number of processors in the system. Then, the number of agents in each processor is calculated as follows,

$$NA = [l/2k], \tag{3}$$

where l is the total number of polygons and k is the number of processors. Started nodes for all agents are ascending chosen from node 1.

Step 2: For each processor, we find the candidate lists for all agents in it. In specific, consider a polygon as a node in a graph. Let V_C and V_D are the sets containing started nodes of all agents in this processor and visited nodes in an agent's life, respectively. Initially, $V_D = \Phi$. Two agent parameters a, b are also randomly initiated by positive values,

$$a = rand(0,1) \text{ and } b = 1-a. \tag{4}$$

Step 3: Move the first node I_o of V_C to V_D.

$$V_C = V_C \setminus \{I_o\}, \tag{5}$$

$$V_D = V_D \cup \{I_o\}. \tag{6}$$

Step 4: For any node j which is not in V_D, calculate the probability,

$$P(I_0, j) = \frac{1}{a \times d(I_0, j) + b \times SP_{I_0 j}}, \tag{7}$$

where $d(I_0, j)$ is the distance between polygons I_0 and j, $SP_{I_0 j}$ is the area of two polygons I_0 and j.

Step 5: Choose node j_o which satisfies the condition below and add it to V_D,

$$P(I_0, j_0) = \max\{P(I_0, j)\},\ j \notin V_D, \tag{8}$$

$$V_D = V_D \cup \{j_o\}. \tag{9}$$

Step 6: Repeat Step 4 and Step 5 with j_o is the started node until all nodes are in V_D. Then, we will have a sequence $\{X_1, X_2,.., X_l\}$ extracted from V_D where X_t is a polygon, $t = \overline{1,l}$.

Step 7: Select a number of tests for this agent ($NoTest$) by the probability in equation (7),

$$NoTest = MinTest + [(MaxTest - MinTest)\times \max\{P(i, j)\}], \tag{10}$$

where $[MinTest, MaxTest]$ is a given range of the number of test cases. Indexes i and j are two consecutive nodes in V_D.

Step 8: For each test case, perform Stick Procedure to find a suitable solution. In essence, we select $(k-1)$ random 'stick' to put into the sequence $\{X_1, X_2,.., X_l\}$.

```
8.1 $Stick(l, k, i)$                                    (11)
8.2 If $k = 0$ then stop;
8.2 $Stick(i) \leftarrow rand(1, l-k+1)$;
8.3 $Stick(l - Stick(i), k-1, i+1)$;
8.4 End.
```

Step 9: For each block in this test case separated by two continuous 'sticks', calculate the area of all polygons in it (SP_i, $i = \overline{1,k}$).

Step 10: Check the constraints (2) for the current solution. If they are satisfied, we will find the maximum below, and add this test case's solution into the agent's candidate list.

$$\alpha_{Test\_Case} = \max\left\{\frac{SP_i}{S_{DEM}}\right\},\ i = \overline{1,k}. \tag{12}$$

Step 11: Repeat from Step 8 to Step 10 with other test cases. We will receive an agent's candidate list with different values α_{Test_Case}. Find the optimal solution for this agent,

$$\alpha_{agent} = \min\{\alpha_{Test\_Case}\}. \tag{13}$$

Step 12: Repeat from Step 2 to Step 11 with other agents in this processor. Extract the optimal solution of the processor from a list of solutions of agents α_{agent},

$$\alpha_{processor} = \min\{\alpha_{agent}\}. \tag{14}$$

Step 13: Synchronize all processors in the system and find the optimal saving thresholds from $\alpha_{processor}$ if they exist.

$$\alpha_{optimal} = \min\{\alpha_{processor}\}. \tag{15}$$

We then conclude that for given ε, the optimal solution is $\alpha_{optimal}$. Otherwise, no suitable solution is found.

Some important points can be drawn from this algorithm.

- *Firstly*, two agent parameters a and b are different to each agent. As we can see, in equation (7), the probability to choose a next node depends on these parameters. Because we have two criterions $SP_{I_0 j}$ and $d(I_0, j)$, the larger parameter will decide which criterion will be followed. Moreover, the advantage of random agent parameters can be seen as the way to avoid local solutions between agents due to different sequences of them.

- *Secondly*, in Step 8, when selecting 'sticks' to put into a sequence, it is possible that some test cases are the same. Therefore, the number of 'real' test will be reduced. To increase it, we supplement a quantity related to the maximal probability of two nodes in the sequence. Thus, the number of test cases is still in a given range $[MinTest, MaxTest]$.

- *Thirdly*, we will try some combinations of adjacency polygons in a sequence to form solutions. In essence, our algorithm belongs to the greedy approaches. A sequence, in this way, is an ordered list of polygons whose combination between them will create more solutions than original sequence's ones with a specific probability. Consequently, it is better to investigate in a 'good' sequence.

- *Fourthly*, to ensure the time condition and avoid similar solutions between agents, we limit the number of agents to the half of the number of polygons, and process it by parallel computing. Then, the computational time and the solution will be ameliorated.

- *Finally*, SIT-TSA method converges to the global solution instead of the local one due to the best solution selection process among all agents. Moreover, it can answer quickly whether a solution may exist for a given parameters ε or not.

C. An Example

Assume that we have a polygon shape dataset below (Fig.2). The number of processors $k = 3$. The number of agents in each processor is $NA = 1$. In the first processor, $V_C = \{1\}$ and $V_D = \Phi$. Thus, a sequence found by the first agent and some tests from it is illustrated through Fig. 3.

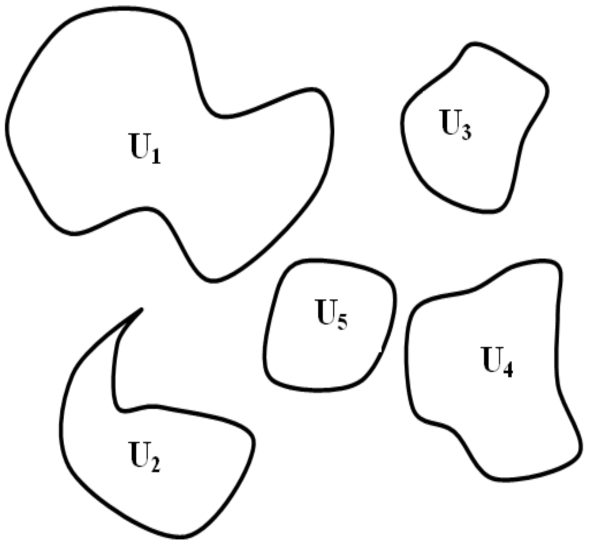
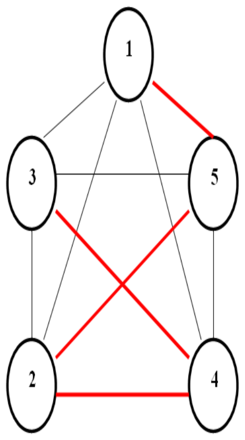


Fig 2. A polygon shape dataset



{1,5}	{2,4}	{3}
{1,5,2}	{4}	{3}
{1}	{5}	{2,4,3}
…	…	…

Fig 3. (a) a sequence of an agent; (b) some tests from this sequence

IV. EXPERIMENTS

A. *Theoretical Evaluation*

In this section, we will evaluate the proposed algorithm both by time and space complexity. In SIT-TSA algorithm, Step 4 to Step 6 requires $[l \times (l-1)]/2$ calculations. Step 7 to Step 11 takes $NoTest \times l$ operations. Each processor has $[l/2k]$ agents. Therefore, the total computing time of the algorithm is,

$$\left[\frac{l}{2k}\right] \times \left(\frac{l \times (l-1)}{2} + NoTest \times l\right) \approx \frac{1}{2k} \times O(l^3). \qquad (16)$$

The memory space in each processor to store V_D, probabilities of all nodes, and the sequences of all agents is $O(l)$. Therefore, the total memory space is $k \times O(l)$.

B. *Experimental Setup*

Theoretical time complexity does not always indicate clearly the speed of an algorithm. For this, measurements of CPU time often give better information. Therefore, in this part, we have implemented the proposed algorithm in addition to PSO-TSA in C programming language, and executed them on a Linux Cluster 1350 with eight computing nodes of 51.2GFlops. Each node contains two Intel Xeon dual core 3.2 GHz, 2 GB Ram. In SIT-TSA, the parameters $[MinTest, MaxTest]$ and ε are initially set as $[10,100]$ and 0.001, respectively. In PSO-TSA, the population and the maximal iteration are 1000 and 100, respectively. Terrain data are taken from Bolzano - Bolzen province [5].

C. *Saving Threshold Comparison*

Firstly, we calculate the values of saving threshold for SIT-TSA and PSO-TSA algorithms following by different number of polygons and number of processors in two terrain data. We compare the saving thresholds of two algorithms for a specific number of polygons and processors, and find the smallest one. Then, we increase the number of cases for the algorithm that has the smallest value of saving threshold by one. The statistics are grouped following by the number of polygons. Results are illustrated in Fig. 4. Similarly, we perform another comparison following by the number of processors, and summarize the results in Fig. 5.

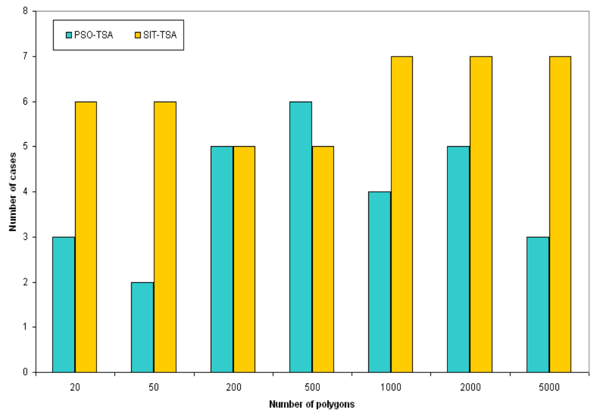


Fig 4. The number of cases following by the number of polygons

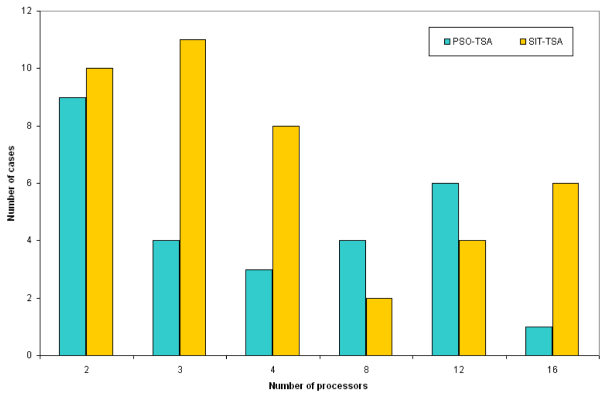


Fig 5. The number of cases following by the number of processors

Fig. 4 clearly shows that the number of cases generated by SIT-TSA is larger than the one of PSO-TSA. For example, when the number of polygons is 20, SIT-TSA produces six best cases when PSO-TSA makes three cases only. The maximal difference between two algorithms is four cases when the numbers of polygons are 50 and 5000. PSO-TSA is only better than SIT-TSA when the number of polygons is 500. In general, SIT-TSA still brings more cases than PSO-TSA does.

Fig. 5 reconfirms that SIT-TSA generates more cases than PSO-TSA does. The maximal difference between two algorithms is larger than the result in Fig. 4. In fact, this number is 7 when the number of processors is 3. PSO-TSA is better than SIT-TSA when the numbers of processors are 8 and 12. For the remains, SIT-TSA is shown to obtain better results than PSO-TSA.

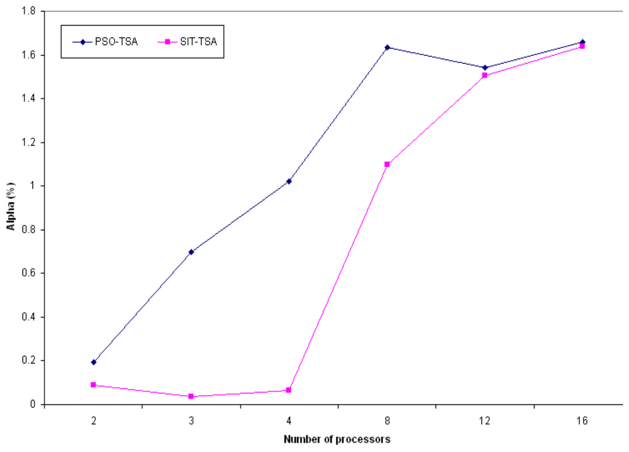


Fig 6. Average saving thresholds of algorithms by number of processors

In Fig. 6, we calculate the average numbers of saving threshold following by the number of processors in both algorithms. A comparison between them is made in this figure. This test clearly points out that the α values of SIT-TSA are smaller than the ones of PSO-TSA. The maximal and minimal differences between two lines are found at 0.96 and 0.02 when the numbers of processors are 4 and 16, respectively.

Some remarks are extracted through this section:

- *Firstly*, SIT-TSA really brings better results than PSO-TSA.

- *Secondly,* as illustrated in Fig. 5 and Fig. 6, we should choose the number of processors from 3 to 8 in order to obtain the best results of SIT-TSA algorithm.

D. Running Times Comparison

In this section, we will make a comparison of the running times between two algorithms following by the number of polygons. The results are shown in Fig. 7. Obviously, SIT-TSA is faster than PSO-TSA when the number of polygons is smaller than 750. The largest difference between two algorithms is 595 times when the number of polygons is 20. The difference is getting smaller when the number of polygon increases. When the number of polygons is larger than 1000, the difference is below one, and SIT-TSA is slower than PSO-TSA.

The reason for slow running times of SIT-TSA, when the number of polygons increases, can be recognized through the incremental level. In PSO-TSA, the average increment between two consecutive numbers of polygons is approximately 2.5. This number in case of SIT-TSA is 17.3. As such, more polygons are added, the running times of SIT-TSA are longer.

Some remarks can be found from this test:

- *Firstly*, SIT-TSA is faster than PSO-TSA when the number of polygons is below 750.

- *Secondly*, from Fig. 4 to Fig. 7 we get a remark that the number of polygons should be smaller than 500 to get the best results of SIT-TSA algorithm in both the saving threshold and the running times.

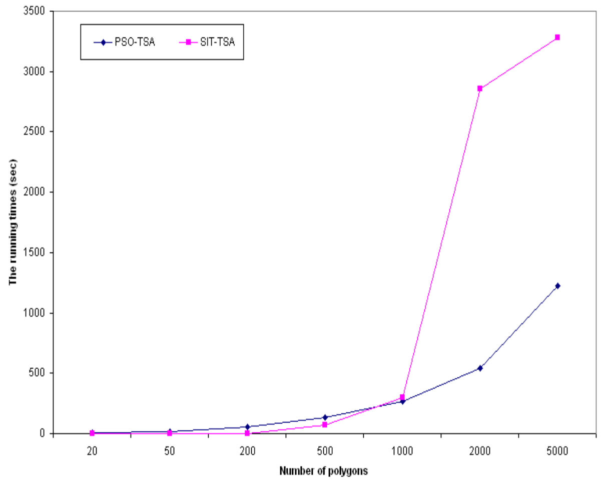


Fig 7. The running times following by the number of polygons

V. CONCLUSION

In this paper, we introduced a novel stochastic-based optimization algorithm namely SIT-TSA for TSO problem. This method employed the ideas of Wife-Selection scenario and Stick Procedure to find the optimal solution with the supports of parallel computing. It was verified by time and space complexity as well as numerical experiments. The experimental results showed that SIT-TSA obtains better saving thresholds than PSO-TSA and is suitable for TSO problem.

Future works will concern some methods to store terrains in a database as well as perform attribute queries in a 3D GIS system.

VI. ACKNOWLEDGMENT

This work is sponsored by the research project of VNU Hanoi Information Technology Institute, Vietnam National University (VNU-ITI - Project No. QCT.12.07) and the research grant of National Foundation for Science and Technology Development (NAFOSTED - Project No. 102.01-2012.14).

REFERENCES

[1] Anderson, H. L., “Metropolis, Monte Carlo and the MANIAC”, *Los Alamos Science*, vol. 14, 1986, pp. 96-108.

[2] Holland, J. H., “Adaptation in natural and artificial system”. Ann Arbor: The University of Michigan Press, 1975.

[3] Kennedy, J., Eberhart, R. C., “Particle swarm optimization”, In: *Proceedings of IEEE International Conference on Neural Networks*, Piscataway, NJ, 1995, pp. 1942-1948.

[4] Son, L. H., Thong, P. H., Linh, N. D., Hoa, N. D., Cuong, T. C., “Some Results of 3D Terrain Splitting By 2D Polygonal Vector Data”, *International Journal of Machine Learning and Computing*, vol.1, no. 3, 2011, pp. 253-262.

[5] Son, L. H., Thong, P. H., Linh, N. D., Cuong, T. C., Hoa, N. D., “Developing JSG Framework and Applications in COMGIS Project”, *International Journal of Computer Information Systems and Industrial Management Applications*, vol. 3, 2011, pp. 108-118.

[6] Thien, N. D., Son. L. H., Lanzi, P. L., Thong, P. H., ”Heuristic Optimization Algorithms For Terrain Splitting and Mapping Problem”, *International Journal of Engineering and Technology*, vol. 3, no. 4, 2011, pp. 376-383.