MISSOURI S&T

Missouri University of Science and Technology

Scholars' Mine

Electrical and Computer Engineering Faculty Research & Creative Works

Electrical and Computer Engineering

01 Jan 2003

A Comparison of Dual Heuristic Programming (DHP) and Neural Network Based Stochastic Optimization Approach on Collective Robotic Search Problem

Nian Zhang

Donald C. Wunsch
Missouri University of Science and Technology, dwunsch@mst.edu

Follow this and additional works at: https://scholarsmine.mst.edu/ele_comeng_facwork

Part of the Electrical and Computer Engineering Commons

Recommended Citation

A Comparison of Dual Heuristic Programming (DHP) and Neural Network Based Stochastic Optimization Approach on Collective Robotic Search Problem

Nian Zhang
Applied Computational Intelligence Lab
Dept. of Electrical and Computer Engineering
University of Missouri-Rolla
1870 Miner Circle
Rolla, MO 65409 USA
nzhang@umr.edu

Donald C. Wunsch II
Applied Computational Intelligence Lab
Dept. of Electrical and Computer Engineering
University of Missouri-Rolla
1870 Miner Circle
Rolla, MO 65409 USA
dwunsch@ece.umr.edu

Abstract- An important application of mobile robots is searching a region to locate the origin of a specific phenomenon. A variety of optimization algorithms can be employed to locate the target source, which has the maximum intensity of the distribution of some detected function. We propose two neural networks algorithms: stochastic optimization algorithm and dual heuristic programming (DHP) to solve the collective robotic search problem. Experiments were carried out to investigate the effect of noise and the number of robots on the task performance, as well as the expenses. The experimental results showed that the performance of the dual heuristic programming (DHP) is better than the stochastic optimization method.

Keywords: Neural Networks, Dual Heuristic Programming (DHP), Adaptive Critic Designs.

1. INTRODUCTION

In recent years there has been growing interest in collective robotic search problem. The primary reason is that mobile robots can complete high-risk tasks. Mapping minefields, extraterrestrial and undersea exploration, detecting the location of chemical and biological weapons, and the location of explosive devices are its important applications. The goal of the team of robots is to find the origin of a specific phenomenon with the maximum intensity, by sharing information between robots, and to aggregate around the phenomenon. For example, we may need to drive a large number of inexpensive, expendable sensory robots in hazardous or hostile environments, with a particular emphasis on sensing concentrations of hazardous chemicals. In cases where human intervention through teleoperation is not available, the robot team must be deployed in a territory without supervision, requiring an autonomous decentralized coordination strategy. Spatial distribution of the mines, or more generally of the objects being searched for, can be regular, totally random, patchy or graded. Search strategies are affected by these distributions and need to be optimized for various environments [1].

Investigations of collective behavior are considerably rarefied, and studies involving collective search are rarer still.

The foraging problem [2][3][4][5], in which robots collect objects scattered in the environment, is a canonical problem related to the source location problem. Relevant results in robotics, which are inspired by animal behaviors, were discussed in [6]. A decentralized *alpha-beta coordination* is proposed for an agent team searching for source targets [7]. Its simulations confirm the ability of the team to find a source and stabilize the steady-state mean squared error. It has been shown in [8] that how space-filling curves can enhance the efficiency and robustness of geographic search by robot collectives. In [9], a control system employing an extended Kalman filter (EKF) and different styles of Global Position System (GPS) is introduced to control a mobile robot to search a given rectangular area. Three neural networks algorithms were also provided [10].

We propose neural networks based stochastic optimization algorithm and dual heuristic programming algorithm to solve the collective robotic search problem. The robots search for the target source by collaboration. The performances of the two approaches are compared.

2. PROBLEM DESCRIPTION

Let's assume we have a two-dimensional bounded Euclidean space. The domain is shown in Fig. 1. Several signal sources are randomly distributed in the domain. One of the sources is the target, which has the maximum intensity, and the others are classified as noises. We presume there are no obstacles in this domain and each source emits signal evenly in every direction. Robots don't know the location of the target source *a priori*. The searching will stop when all the robots converge to the target.

(a)   (b)
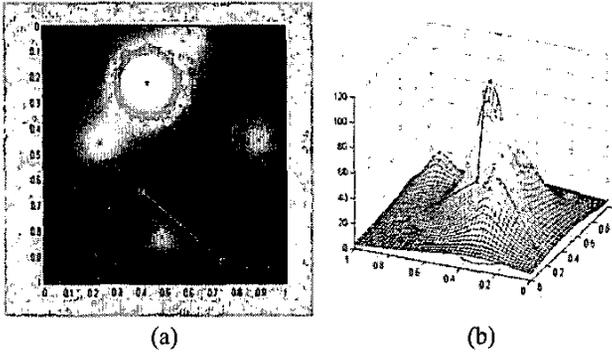
Fig. 1 Robot search region with five sources and four robots. Experiments were carried out on this 1 by 1 length unit square domain. The red star denotes the target, and the blue crosses denote the noises. The square blocks in different colors denote the robots. The lines indicate each step of an individual robot. In (a), x-axis and y-axis denotes the coordinates of sources and robots. The illumination of each source is shown. In (b), x-axis and y-axis denotes the coordinates of sources and robots. The z-axis denotes the illumination of each source.

3. STOCHASTIC OPTIMIZATION APPROACH

The only way that we detect the sources is the specific signals they emit. Robots measure the sources by the *illumination*, E, which is defined as light flux per unit area of surface [11]. Total illumination at a point is the sum of illumination magnitudes from all sources. Thus, we have the following expression for the distribution of illumination:

$$E(x,y) = \sum_i \frac{I^i}{h^{i^2} + (x - s_x^i)^2 + (y - s_y^i)^2} \quad (1)$$

Where x and y are the coordinates of the robot, I^i is the intensity of i-th source with coordinates of s_x^i, s_y^i, and the altitude h^i. However, we are searching for the source of maximum intensity. Expression (1) can only help us obtain the maximum illumination. In order to assure that the target source has the maximum of (1), it is necessary for us to set the intensity of the target to I and the intensity of noise sources to 0.3. Although we know the intensities of the sources, the robots don't. We approximate the distribution of illumination based on the known magnitudes at the explored locations. Using the obtained approximation the robots estimate the locations of the sources, and then go over there to check, and then correct the model. The search will finish if the model does not require further adjustments.

The neural network architecture is shown in Fig. 2. The neural network is continuously trained on–line to estimate the magnitude at a location of given coordinates. The network is customized in such a way so it can compute (1). The

coordinates P and intensities I of the sources are inputs to the network. The transfer function of the first layer is I / n_1, which means neurons in the first layer will calculate the quotient of the pair of intensity and coordinates. The transfer function of the second layer is *linear*. The neural network provides us with a gradient of approximation error, which we use to train the network. The constraints are lower and upper bounds for location and intensity of a source. The neural network is trained using the constrained optimization algorithm from the Optimization Toolbox in Matlab [12].
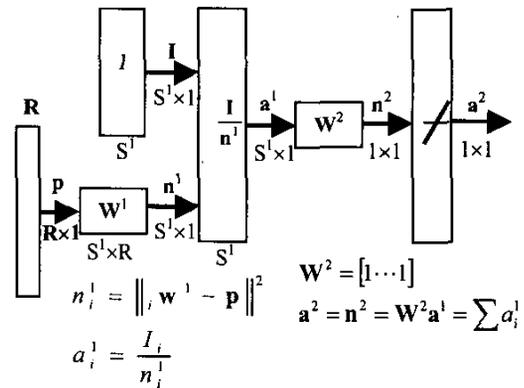
Fig. 2 Neural network architecture
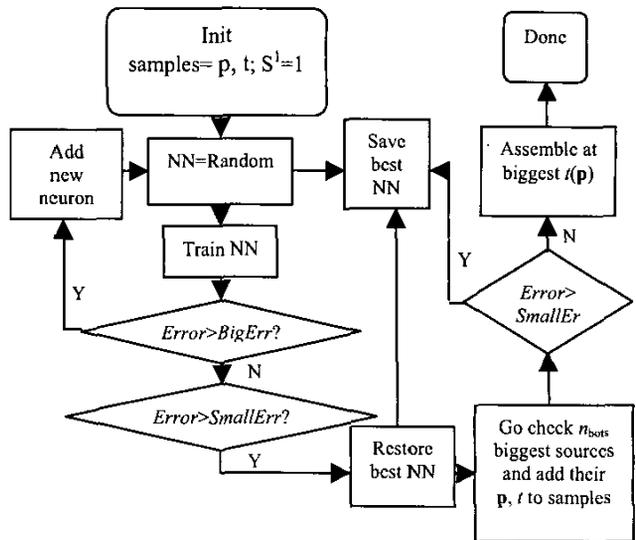
Fig. 3 Algorithm flowchart

249

The algorithm is briefly outlined in Fig. 3. We start from one neuron, which represents one source. P stands for coordinate, and t stands for the source intensity. We initialize the network by random and then train it. We have two threshold values for the error: *BigErr* and *SmallErr*. When the error is bigger than *BigErr*, we consider the chosen number of sources in the model (i.e. number of neurons) insufficient, and thus add new neurons to the network; if the error is between *SmallErr* and *BigErr*, we perform the Monte-Carlo search for a better approximation training with different initial weights. Once a satisfactory approximation (i.e. error < *SmallError*) is found we consider the parameters of the network to be the coordinates and intensities of the sources. Let's define n as the number of robots. We choose the n brightest sources and send the robots to examine these estimated locations. The robots move with constant length of steps, and measure the illumination at each step. Therefore, at every step the sample set is supplemented with new samples. The robots remember the history of all the visited points. Whenever the neural network does not change with new points, that is, the error remains under the *SmallError*, the search will end. The robots will look through the track history, and then all go to the brightest source. Otherwise, the Monte-Carlo search is repeated.

4. DUAL HEURISTIC PROGRAMMING (DHP) APPROACH

DHP uses the critic network to estimate the derivatives of J function with respect to the state vector [13]. The cost-to-go function J in the Bellman equation of dynamic programming is expressed as follows:

$$J(t) = \sum_{k=0}^{\infty} \gamma^k U(t+k) \qquad (2)$$

Where γ is a discount factor for finite horizon problems $(0<\gamma<1)$, and $U(\cdot)$ is the utility function. The critic is trained forward in time, which is of great importance of real-time operation. The critic network tries to minimize the following error measure over time:

$$\|E\| = \sum_t E^T(t)E(t) \qquad (3)$$

where

$$E(t) = \frac{\partial J[Y(t)]}{\partial Y(t)} - \gamma \frac{\partial J[Y(t+1)]}{\partial Y(t)} - \frac{\partial U(t)}{\partial Y(t)} \qquad (4)$$

where $\partial(\cdot)/\partial Y(t)$ is a vector containing partial derivatives of the scalar (.) with respect to the components of the state vector, Y(t). According to the chain rule, each of n components of the vector E (t) can be finally determined by

$$E_j(t) = \frac{\partial J(t)}{\partial R_j(t)} - \gamma \frac{\partial J(t+1)}{\partial R_j(t)} - \frac{\partial U(t)}{\partial R_j(t)}$$
$$- \sum_{k=1}^{m} \frac{\partial U(t)}{\partial A_k(t)} \frac{\partial A_k(t)}{\partial R_j(t)} \qquad (5)$$

where

$$\frac{\partial J(t+1)}{\partial R_j(t)} = \sum_{i=1}^{n} \lambda_i(t+1) \frac{\partial R_i(t+1)}{\partial R_j(t)}$$
$$+ \sum_{k=1}^{m} \sum_{i=1}^{n} \lambda_i(t+1) \frac{\partial R_i(t+1)}{\partial A_k(t)} \frac{\partial A_k(t)}{\partial R_j(t)} \qquad (6)$$

$\lambda_i(t+1) = \partial J(t+1)/\partial R_i(t+1)$, and n, m are the numbers of outputs of the model and the action networks, respectively. The adaptation of action network is implemented by propagating $\lambda_i(t+1)$ back through the model down to the action and its goal can be expressed as equation:

$$\frac{\partial U(t)}{\partial A(t)} + \gamma \frac{\partial J(t+1)}{\partial A(t)} = 0, \forall t. \qquad (7)$$

Adaptation of DHP is summarized in Fig. 4. In the figure, the discount factor is assumed to be equal to 1. The critic network is shown in two consecutive moments in time and pathways of backpropagation are depicted in dash lines.
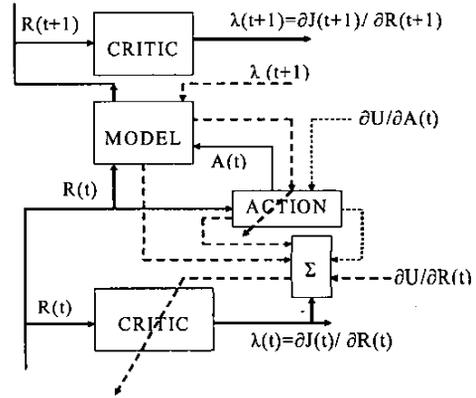
Fig. 4   Adaptation in DHP

4.1 The State Vector

In our case, the state vector $R(t)$ consists of five components at time step t: $v_{lr}(t), v_{mr}(t), v_{sr}(t), f_r(t)$, and $s_r(t)$, where $v_{lr}(t)$ stands for the direction of the center of mass for the large field strength robots relative to robot r, $v_{mr}(t)$ stands for the direction of the center of mass for the medium field strength robots relative to robot r, $v_{sr}(t)$ stands for the direction of the center of mass for the small field strength robots relative to robot r, $f_r(t)$ stands for its field intensity, and

$s_r(t)$ stands for its status. They are determined by the following expressions.

$$v_{cmr}(t) = \frac{\arctan(\dfrac{y_{cm}(t) - y_r(t)}{x_{cm}(t) - x_r(t)})}{2\pi} \qquad (8)$$

where cm stands for l, m or s, $x_{cm}(t)$ and $y_{cm}(t)$ are coordinates of the center of mass of *large*, *medium*, or *small* field strength robots, respectively, and v_{cmr} is the normalized direction of the corresponding center of mass relative to robot r.

$$x_{cm}(t) = \frac{\sum_{r \in C} x_r(t) f_r(t)}{\sum_{r \in C} f_r(t)}$$

$$y_{cm}(t) = \frac{\sum_{r \in C} y_r(t) f_r(t)}{\sum_{r \in C} f_r(t)} \qquad (9)$$

Where $x_r(t)$ and $y_r(t)$ are the coordinates of robot r at time step t. C is the class of robots having large, medium, or small field intensities. $f_r(t)$ stands for the field intensity of the robot r, which can be calculated in (1).

$$x_r(t+1) = \cos(2\pi A_r(t)) + x_r(t) \qquad (10)$$

$$y_r(t+1) = \sin(2\pi A_r(t)) + y_r(t) \qquad (11)$$

Where $A_r(t)$ is the output of the action network for robot r at time step t. The status $s_r(t)$ is calculated as follows:

$$s_r(t) = \frac{f_r(t)}{f_{\max}(t)} \qquad (12)$$

Where $f_{\max}(t)$ is the maximum field intensity that robots have ever sensed.

4.2 Utility Function

The utility function is designed as the sum of mean square of the distance of each robot from the source.

$$U(t) = \sum_r \frac{1}{2}[(x^* - x_r(t+1))^2 + (y^* - y_r(t+1))^2] \qquad (13)$$

Where x^* and y^* are the coordinates of the target source, $x_r(t+1)$ and $y_r(t+1)$ are the coordinates of robot r at time step t+1.

The utility function makes use of the positions of the robots at time step t+1, but not t. This is because of the consideration that one may not know the cost that a robot incurs until it takes an action. If we use the positions of the robots at time step t instead, we would in fact calculate the one step cost of the previous step. For example, if we use the position of a robot at time step t instead, then utility function in (13) is only

the distance between the robot's starting position and the target source. In other word, it has nothing to do with the cost that it incurs at its first step.

5. TRAINING OF MODEL, CRITIC, AND ACTION NEURAL NETWORKS

5.1 Critic and Action Neural Networks

DHP design for the robotic search problem is shown in Fig. 5. We combine the action network and the critic network as one network, called Action-Critic Module. The inputs to the model neural network are the state vector and the action at time step t, and its outputs are the state vector at time step t+1. The outputs of the model network are the inputs to the Action-Critic module, and the outputs of the A-C module are the derivative of the cost-to-go function with respect to each state vector component.

The critic network has two layers. There are 20 neurons in the hidden layer. The transfer function of the hidden layer is *logsig*. There are five neurons in the output layer. The transfer function of the output layer is *satlin*. Thus, the critic has five outputs, each of which is the derivative of the cost-to-go function with respect to the state vector component. We use (10) and (11) to compute the next state of the robotic search system. $\partial R_i(t+1)/\partial R_j(t)$ and $\partial R_i(t+1)/\partial A_k(t)$ can be calculated directly from the equations of the dynamics, (10) and (11). And $\partial A_k(t)/\partial R_j(t)$ can be obtained from the action network: $\partial U(t)/\partial R_j(t)$ and $\partial U(t)/\partial A_k(t)$ can be calculated according to the definition of the utility function. So we have all the components required to calculate the target of output of the critic network in (5).

The action network has two layers. There are 20 neurons in the hidden layer. The transfer function of the hidden layer is *logsig*. There is one neuron in the output layer. The transfer function of the output layer is *satlin*. Thus, the action network has only one output, the direction. Each robot can make a move of one grid square at each time step.
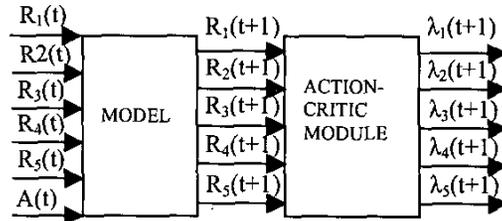
Fig. 5 DHP Architecture for Robotic Search Problem

251

5.2 Model Network

DHP has an important advantage since its critical neural network builds a representation for the derivatives of J directly by being explicitly trained on them through $\partial U(t) / \partial R(t)$ and $\partial U(t) / \partial A(t)$. For example, in the area of model-based control, the model neural network can be pretrained. The partial derivatives of the utility function U(t) with respect to $A(t)$, and R(t), $\partial U(t) / \partial A(t)$ and $\partial U(t) / \partial R(t)$, respectively, are obtained by backpropagating the utility function, U(t) through the Model network as shown in Fig. 6. To adapt the action neural network, only the derivatives $\partial J(t) / \partial R(t)$ or $\partial J(t) / \partial A(t)$ are required, rather than the J function itself.
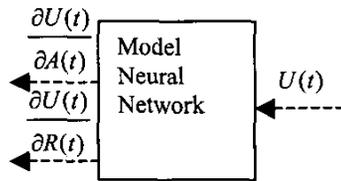
Fig. 6 Backpropagation of U(t) through the Model network

6. SIMULATION AND EXPERIMENTAL RESULTS

Experiments were performed using multiple robots. We compare the performance of neural networks trained by stochastic optimization approach and the dual heuristic programming (DHP) approach. The performance is evaluated by the average route length per robot. The less the route length per robot the robots take, the faster the robots converge at the target source. Two experiments were done. In the first experiment, two sources are randomly distributed in the domain. We increase the number of robots, and observe the performance of the two neural networks approaches. The route length comparison is shown in Fig. 7.
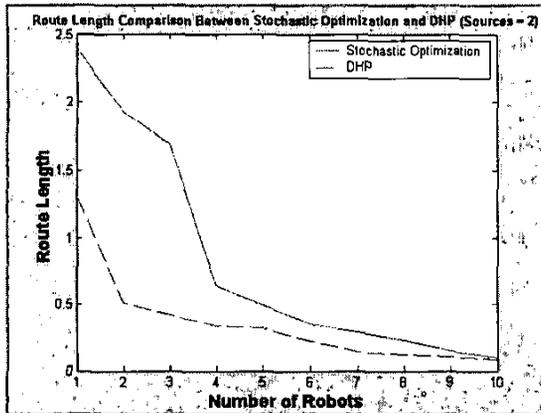
Fig. 7 Route length comparison of stochastic optimization approach and DHP approach when sources = 2.

From Fig. 7, we find that DHP has better results against stochastic optimization, especially when the number of robots is less than the number of sources. In addition, the more the robots, the less route length the robots take. This is because the more the robots, the higher probability that the robots are close to the target source. Moreover, when the number of robots is much more than the sources (i.e. 10 to 2), the difference between the two approaches is small.

In experiment 2, we increase the number of sources to four, and observe the performance of the two neural networks approaches with the increase of the number of robots. The route length comparison is shown in Fig. 8. From Fig. 8, we find that the route lengths of both of the two approaches are much greater than that in Fig. 7. When the number of robots is more than the number of sources, the route length of two approaches decrease considerably. In addition, route length of DHP is less than the stochastic optimization approach when the number of robots is less than the number of sources. Moreover, when the number of robots exceeds the number of sources, the difference between the two approaches is very small.
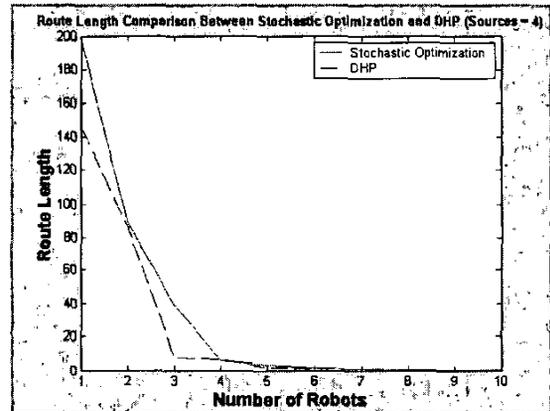
Fig. 8 Route length comparison of stochastic optimization approach and DHP approach when sources = 4.

7. CONCLUSION

This paper has presented two neural networks approaches to solve the collective robotic search problem. The stochastic optimization approach works effectively if the number of sources is not greater than the number of the robots. In this case, there will be a high probability that there will be one robot stand by each source. It is also shown that the more the robots, the less the route length per robot. In addition, the more the noise, the more route length per robot. Dual Heuristic Programming (DHP) takes much less route

length than the stochastic optimization approach, especially when the number of robots is less than the sources. Since the expenses are proportional to the route length, DHP proves to be an efficient approach to save more expenses in a very noisy environment. This is because DHP is a promising neural network design method to solve optimal control problem under the conditions of noise and uncertainty.

ACKNOWLEDGEMENT

REFERENCES

[1] Erol Gelenbe, Nestor Schmajuk, John Staddon and John Reif, "Autonomous Search By Robots and Animals: A Survey", Robotics and Autonomous Systems, Vol. 22, pp. 23-34, 1997.

[2] R. Arkin and J. Hobbins. Dimensions of Communication and Social Organization in Multi-agent Robotic Systems. Proceeding of Simulation of Adaptive Behavior, 1994.

[3] S. Goss and J. Deneubourg. Harvesting by a group of robots. Proceeding of European Conference of Artificial life, 1992.

[4] M. Mataric. Interaction and Intelligent Behavior. MIT AI Lab Tech Report AITR-1495, 1994.

[5] L. Steels. Cooperation between distributed agents through self-organization. European Workshop on Modeling Autonomous Agents in a Multi-Agent World, 1990.

[6] In-Keun Yu, C. S. Chou, Y.H. Song. "Application of the Ant Colony Search Algorithm to Short-term Generation Scheduling Problem of Thermal unites", Proceeding of 1998 International Conference on Power System Technology, Vol. 1, pp. 552-556, 1998.

[7] Steven Y. Goldsmith and Rush Robinett, "Collective Search by Mobile Robots Using Alpha-Beta Coordination", Collective Robotics Workshop '98, Agent World, Paris, 1998.

[8] Shannon V. Spires and Steven Y. Goldsmith. "Exhaustive Geographic Search with Mobile Robots Along Space-Filling Curves. Proceeding of the first International Workshop, CRW '98. Paris, France, July 1998.

[9] Brad Bauer and Gerald Cook. "Algorithm and Simulation Development of Autonomous Robotics Search", Proceeding of the 23rd International Conference on Industrial Electronics, Control and instrumentation, 1997. Vol. 3, pp. 1278-1283.

[10] Nian Zhang, Alexander Novokhodko, Donald C. Wunsch II, and Cihan H. Dagli, "A Comparative Study of Neural Networks Based Learning Strategies on Robotic Search Problems", *Proceedings of the SPIE Application and Science of Computational Intelligence Conference*, Vol. 4390: 214-235, Orlando, Florida, April 2001.

[11] George Shortley and Dudley Williams. Elements of Physics. Prentice Hall, Englewood Cliffs, N.J. 1961.

[12] Thomas Coleman, Mary Ann Branch, and Andrew Grace, Optimization Toolbox User's Guide, The MathWorks Inc., 1999.

[13] Danil V. Prokhorov and D.C. Wunsch II, "Adaptive Critic Designs," IEEE Trans. on Neural Networks, Vol. 8, No. 5, pp.997-1007, 1997.