International Journal of Fuzzy Logic and Intelligent Systems, vol. 5, no. 4, December 2005 pp. 333-338

Object Search Algorithm under Dynamic Programming in the Tree-Type Maze

In-Hun Jang, Dong-Hoon Lee, and Kwee-Bo Sim*

School of Electrical and Electronic Engineering, Chung-Ang University
221, Heukseok-Dong, Dongjak-Gu, Seoul, 156-756, Korea
Tel : +82-2-820-5319, Fax : +82-2-817-0553, E-mail : kbsim@cau.ac.kr

Abstract

This paper presents the target object search algorithm under Dynamic Programming (DP) in the Tree-type maze. We organized an experimental environment with the concatenation of Y-shape diverged way, small mobile robot, and a target object. By the principle of optimality, the backbone of DP, an agent recognizes that a given whole problem can be solved whether the values of the best solution of certain ancillary problem can be determined according to the principle of optimality. In experiment, we used two different control algorithms: a left-handed method and DP. Finally we verified the efficiency of DP in the practical application using our real robot.

Key words : Dynamic Programming (DP), Left-handed method, Distributed Autonomous Robotic System(DARS), Tree-type maze

1. Introduction

Recently, the robotic systems have brought forth various applications such as factory automation, space industry, and military operation. Especially, the research of artificial intelligence, i.e., fuzzy logic, artificial neural network, genetic algorithm, reinforcement learning enlightens a robotic agent; thus now the robot can complete a given mission by itself.

A goal search algorithm in the maze environment has received much spotlight to offer a new way of path finding, from the start position to the goal, within shorter time. For historic research example, *Kurozumi* et al. used an improved reinforcement learning scheme for path planning with mobile robots [1]. *Kuroaumi*'s method was the coalescence of both a reinforcement learning and a potential method. *Lanzi* presented the generalization capabilities of XCS and applied his mechanism to solve the maze environment [2].

In the dynamic programming (DP), the solutions to subproblems, are constructed incrementally from those of smaller subproblems and are cached to avoid recomputation [3]. In the main role of DP, an agent becomes so cognitive that it can solve the whole problem if the values of the best solution of certain subproblem can be determined. DP also successively approximates optimal evaluation functions by solving recurrence relation, instead of conducting searches in state space [4-5].

Even though those researches showed redoubtable works, the approach to a real world application still has the limitation due to the uncertainty of control algorithms, and most algorithms are confined in the software world only. In this paper, we present the real robot application of DP to find a target object in the tree-type maze. Compared with the left-handed method, one of the strongest algorithms to find a goal in maze, we verify the superiority of DP.

The organization of this paper is as follows. In section 2, we introduce the concept of simple dynamic programming. In section 3, the target object search algorithm under DP is presented. Experimental results from the application of DP to find the object are presented in section 4. In section 5, conclusion and future works are presented.

Manuscript received Sep. 1, 2005; revised Nov. 30, 2005.

* Corresponding author

This research was supported by the project of *Developing SIC (Super Intelligent Chip) and Its Applications* under the program of Next generation technologies of Ministry of Commerce, Industry, and Energy.

2. The Basic Concept of Dynamic Programming

The term dynamic programming refers to a collection of algorithms that can be used to computer optimal policies given a perfect model of the environment as a Markov Decision Process (MDP). Classical DP algorithms are of limited utility in reinforcement learning both because of their assumption of a perfect model and because of their great computational expense, but they are still important theoretically. In this paper, we usually assume that the environment is a finite MDP. That is, we assume that its state and action sets, S and $A(s)$, for $s \in S$, are finite, and that its dynamics are given by a set of transition probabilities, $P^a_{ss'} = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}$, and expected immediate rewards, $R^a_{ss'} = E\{r_{t+1} \mid a_t = a, s_t = s, s_{t+1} = s'\}$, for all $s \in S$, $a \in A(s)$, and $s' \in s^+$ (s^+ is S plus a terminal state if the problem is episodic). Although DP ideas can be applied to problems with continuous state and action spaces, except solutions are possible only in special cases. A common way of obtaining approximate solutions for tasks with continuous states and actions is to quantize the state and action spaces and then apply finite-state DP methods.

The key idea of DP, and of reinforcement learning generally, is the use of value functions to organize and structure the search for good policies.

2.1 Value Functions

Almost all reinforcement learning algorithms are based on estimating value functions – functions of states (or of state-action pairs) that estimate how good it is for the agent to be in a given state (or how good it is to perform a given action

A policy, π, is a mapping from each state, $s \in S$, and $a \in A(s)$, to the probability $\pi(s,a)$ of taking action a when in state s. Informally, the *value* of a state s under a policy π, denoted $V^{\pi}(s)$, is the expected return when starting in s and following π thereafter. For MDPs, we can define $V^{\pi}(s)$ formally as

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\} \quad (1)$$

where $E_{\pi}\{\}$ denotes the expected value given that the agent follows policy π. Note that the value of the terminal state, if any, is always zero. We call the function V^{π} the *state-value function for policy* π.

Similarly, we define the value of taking action a in state s under a policy π, denoted $Q^{\pi}(s,a)$, as the expected return starting from s, taking the action a, and thereafter following policy π:

$$Q^{\pi}(s,a) = E_{\pi}\{R_t \mid s_t = s, a_t = a\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\} \quad (2)$$

We call Q^{π} the *action-value function for policy* π

The value functions V^{π} and Q^{π} can be estimated from experience. For example, if an agent follows policy π and maintains an average, for each state encountered, of the actual returns that have followed that state, then the average will converge to the state's value, $V^{\pi}(s)$, as the number of times that state is encountered approaches infinity. If separate averages are kept for each action taken in a state, then these averages will similarly converge to the action values, $Q^{\pi}(s,a)$. We call estimation methods of this *kind Monte Carlo methods* because they involve averaging over random samples of actual returns. Of course, if there are very many states, then it may not be practical to keep separate averages for each state individually. Instead, the agent would have to maintain V^{π} and Q^{π} as parameterized functions and adjust the parameters to better match the observed returns[4].

2.2 Dynamic Programming (DP)

The DP is an optimization procedure that is particularly applicable to problems requiring a sequence of interrelated decisions [4]. DP has following two aspects of reinforcement learning.

- DP approximates optimal evaluation functions by solving recurrence relations, instead of conducting searches in state space.
- Backing up state evaluations is a main property of iterative procedures used to solve the recurrence relations.

The usual DP procedure consists of determining the four specified definition: an optimal value function, an optimal policy function, a recurrence relation, and boundary conditions. Figure 1 is an illustrative example of a simple cost path problem.

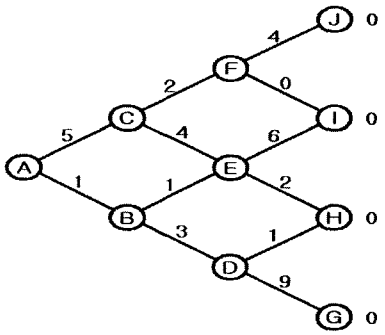

Fig. 1. A simple cost path problem.

In Fig. 1, we consider that the objective of the learning agent is to choose actions so as to drive the agent to a goal state $g \in G \subset S$ at a minimum accumulated cost, where G is a set of goal states and S is the set of all states [3][4]. We notate a measure of value, the optimal value function $V(s)$ as follows:

$V(s) \equiv$ the value of the minimum cost of going from a state s to a goal g.

According to the recurrence relation, the minimal cost of a state s must equal the cost of the best action a, plus the minimal cost of the next state designate by the action a:

$$V(s) = \min_{a \in actions}\{\text{cost}(s,a) + V(\text{next}(s,a))\}, \ \forall s \in S - G \quad (3)$$

where next(s, a) presents the state subsequent to state s dictated by action a, and cost(s, a) signifies the incurred cost of the transition following action a [3]. The value of the goal states are defined by

$$V(s) = 0, \ \forall s \in G. \quad (4)$$

Equation (4) also shows the boundary conditions. The optimal policy function π is clear from Equation (4); thus $\pi(s) = a$ such that $V(s) = \min_{a \in \text{actions}}\{\text{cost}(s, a) + V(\text{next}(s, a))\}$ [4].

Let us assume that the goal is Ⓗ in Fig. 1. At the first state, if an agent takes an action from Ⓐ to ʙ(move downward), it yields by equation (3) and (4)

$$\begin{aligned} V(s_0) &= \min_{a \in actions}\{\text{cost}(s_0, down) + V(\text{next}(s_1, a_{next}))\} \\ &= \min_{a \in actions}\{1 + V(\text{next}(s_1, a_{next}))\} \end{aligned} \quad (5)$$

where s_0, s_1, s_2, and s_3 are elements of S. Now, we need to determine $V(\text{next}(s_1, a_{next})$ so as to define the value of $V(s_0)$. Second, if the agent takes an action from ▣ to ▣ (move upward), it yields

$$V(s_1) = \min_{a \in actions}\{\text{cost}(s_1, up) + V(\text{next}(s_2, a_{next}))\} = \min_{a \in actions}\{1 + V(\text{next}(s_2, a_{next}))\}. \quad (6)$$

Third, if the agent moves toward ▣ from ▣

$$V(s_2) = \min_{a \in actions}\{\text{cost}(s_2, down) + V(\text{next}(s_3, a_{next}))\} = \min_{a \in actions}\{1 + V(\text{next}(s_3, a_{next}))\}. \quad (7)$$

At the final state, the agent recognize that s_3 is the goal state by its value is 0; that is

$$V(s_3) = 0 \quad (8)$$

due to s_3 is satisfied with the boundary condition. Consequently, by the chain relationship equation (5), (6), and (7), the minimum cost from $s_0 \rightarrow s_3$ is as follows:

$$V(s_0 \rightarrow s_3) = 1 + 1 + 2 + 0 = 4. \quad (9)$$

As we have already assumed ▣ is the goal, the learning is complete.

3. The Target Object Search under Dynamic Programming in the Tree-Type Maze

In this section, we introduce a search algorithm based under dynamic programming to find a goal in the tree type maze.

Figure 2 presents an example of the tree-type maze environment. As we can know intuitively, its architecture is very similar to the tree hierarchy of depth three. The only thing has been changed is that we refer the *'depth'* as the *'state.'* $P_{x,y}$, the values upon links, is the probability of goal find if an agent takes an action from N_x to N_y. In this sense, the cost function $V(s)$ corresponds to the probability function $P(s)$. However, in this case we have no pre-acquired data of the task environment; thus we need to assign appropriates value upon each $P_{x,y}$ after an action was taken. To simplify the task, we consider only two conditions; those are 'Here is the goal or not?' and 'Here is the boundary or not?' Simply, we use a notation 'G-yes / G-no' and 'B-yes / B-no.'

By these conventions, we can easily determine the probability values. If 'G-no and B-no' after the first action, then, the agent just allocate 50 to correspondent $P_{x,y}$, because we only consider about two condition, 'goal or not' and 'boundary or not?'. Therefore, we can think the probability of goal find of this action is 50 percent. If 'G-yes and B-yes' after the second action, the $P_{x,y}$ is 100. Otherwise, if 'G-no and B-yes', related $P_{x,y}$ is 0. If the maze is of depth four, the value after the second action is 25, because the result is the one choice out of four possible actions could be taken. Thus, when the tree is depth of n, if 'G-no and B-no' after an action, $P_{x,y}$ will be

$$Px, y = 100/2^{n+1} \quad (10)$$

where $x = n^{\text{th}}$ state and $y = (n+1)^{\text{th}}$ state respectively.

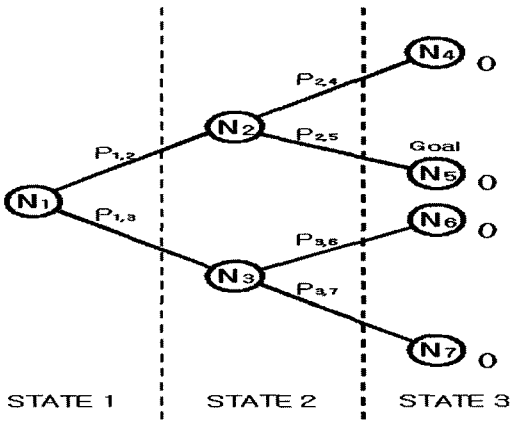

Fig. 2. An example of the tree-type maze environment.

Finally, the agent reaches to the goal as checking these probability values by the recurrent characteristic of DP. The total algorithm is represented in Table 1.

Table 1. The goal find algorithm in the tree-type maze.

For each s, a initialize $P_{x,y}$ zero.
Observe the current state s.
Do forever
- Select the action a, up or down, at the current state
- Observe the new state s' whether the goal or not
- Check the boundary
- Assign the value of $P_{x,y}$ to the state transition
- $s \leftarrow s'$

4. Experiments with Two Different Maze Searching Algorithms

We performed experiments by applying two different maze searching algorithm: left-handed method versus DP. In section IV-A, we introduce our small mobile robot system. Experimental results under two different maze searching algorithm will be presented in the following sections.

4.1 Architecture of Small Mobile Robot

Our small mobile robot system consists of two subparts and a main controller part. Figure 3 shows two subparts, sensor emitter-detector and motor driver, a main part, and the appearance of the robot [6].

The subparts are infrared sensor part and motor driving part. Each subpart has its own controller, PIC16f873A, to perform its unique function more efficiently. The main controller part, Atmega128, controls the two subparts to avoid process collision and to determine the actions at each state. Then, drive

the motors to move the robot toward the next state under the applied algorithm.

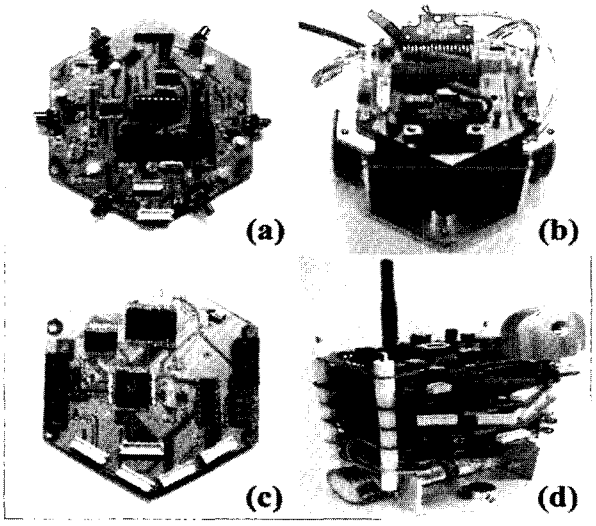

Fig. 3. (a) Sensor emitter-detector pair, (b) motor driver, (c) a main controller, and (d) the appearance of robot.

The emitter is Kodenshi EL-1kl3, high-power GaAs infrared sensor. The detector is ST-1kla, high sensitivity NPN silicon photo-transistor. NMB PG25L-024 stepping motor is used as the driving part. Its characteristics are the following: drive voltage-12V, drive method 2-2 phase and 0.495 step angle [5][6].

4.2 Experiments

The task of the robot is: *"Find the target object, that is the goal, while tracking through the tree-type maze."* We set up the boundary wall of the goal as white to let the robot recognize by its infrared sensor value. Therefore, when the robot meets the boundary and goal, analog-to-digital value is almost the maximum value, because the white may reflect the most of

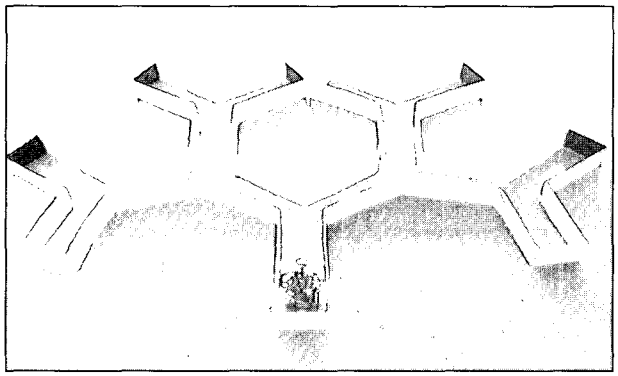

Fig. 4. The appearance of the maze environment.

infrared emission. Otherwise, we set up the boundary wall, which is not a goal, as black so as to give the robot information with the zero value. Figure 4 presents the maze environment being used in the experi- ments. Even though the maze seems like little different, it has the same topological hierarchy with the tree of depth four.

4.2.1. Left-handed method

First, we adapted the left-handed method, which makes the robot to follow the wall of its left side, to find a goal. The more did the tree-maze have depth, the longer time had been taken to complete the task. If we call the total time to complete the goal checking in one Y-shape as a *step*, the total time to find the goal can be mathematically defined as

$$\textit{the total time to find a goal} = 2^{n-1} \times \textit{steps} \quad (9)$$

where n is the number of state. The equation (9) shows that the left-handed method is not so strong a method to be adapted to the task. Moreover, it would be very time and power consuming in the real world situation. Therefore, we performed the experiment with the tree of depth four and derived a general formula for more bigger tree architecture. Figure 5 shows that the robot is searching the goal under the left-handed method in the tree-maze environment.

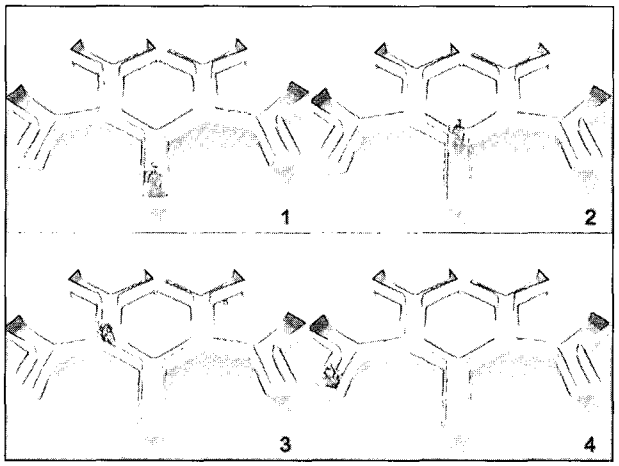

Fig. 5. The robot is searching the goal under the left-handed method.

4.2.2. Dynamic Programming

Second, we applied the DP to find a goal. We allowed the robot to predicts its direction randomly, left or right, and store it at each state. In this sense, the performance of DP depended on how correct the prediction was. Therefore, total time to complete the goal checking can be formulized as follows:

$$\textit{the total time to find a goal} = \begin{cases} \textit{depth} \times \textit{steps}, & \text{if every prediction was correct.} \\ (2^n - 1) \times \textit{steps}, & \text{if every prediction was wrong.} \end{cases} \quad (10)$$

In the experiment, the *step* value was 2.4 second and the depth of the tree was 4. Consequently, the robot found a goal within 9.6 second for the best case and $(2^4-1)\times 2.4 = 36$ second for the worst case. Figure 6 shows that the robot is performing the task under DP in the maze.

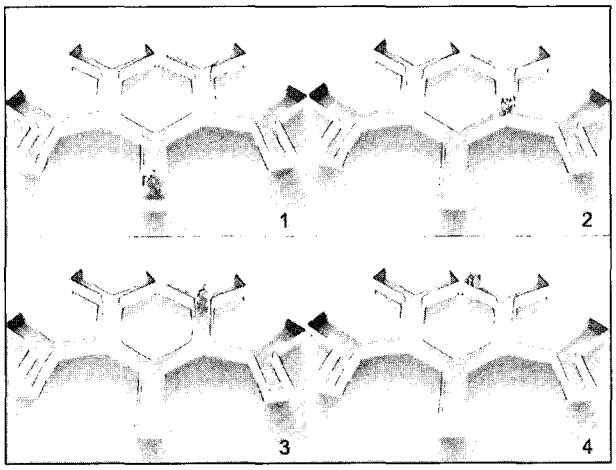

Fig. 6. The robot is searching the goal under DP.

The performance of DP over the left-handed method is depicted in Fig. 7.

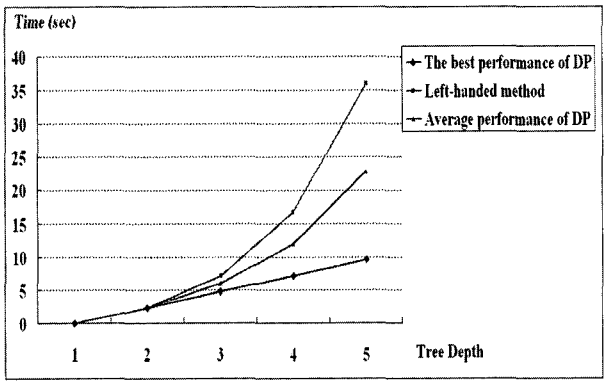

Fig. 7. The performance of DP over the left-handed method.

5. Conclusion and Further Works

In this paper, we presented the application of DP to find a goal in the tree-type maze environment. First, we built a small mobile robotic system and organized the tree-type maze with Y-shaped blocks. Then, we performed the experiment under two different algorithms: the left-handed method and DP.

Under the left-handed method, the more increased is the tree depth, the more time is spent in the ratio of 2 power of n to complete the given task. Under DP, however, the result is up to the probability of correct prediction; thus we could obtain a splendid result for the best case.

For the further research, first, we need to apply DP more complicate problem. In this paper, we applied DP to the ideal application. We also set up the maze environment as the tree-shaped. Therefore, the research of more practical application should be followed. Second, the more sophisticated prediction error handling is required. With the error data, we can set the fuzzy membership function, modular network model, and so on. Finally, the total robot system should be refined to obtain better results.

References

[1] R. Kurozumi, S. Fujisawa, T. Yamamoto, and Y. Suita, "Path planning for mobile robots using an improved reinforcement learning scheme," in *Proc. of the Society of Instrument and Control Engineers*, 2002, pp. 2178-2183.

[2] P. L. Lanzi, "A study of generalization capabilities of XCS," in *Proc. of the 7th International Conference on Genetic Algorithms*, 1997, pp. 418-425.

[3] S. Russell and P. Norvig, *Artificial Intelligence: a Modern Approach*, New Jersey: Prentice Hall, 2003, pp. 287-295.

[4] Richard S. Sutton and Andrew G. Barto, Reinforcement Learning, MIT press.

[5] J. Jang, C. Sun, and E. Mizutani, *Neuro-Fuzzy and Soft Computing*, New Jersey: Prentice Hall, 1997, pp. 258-273.

[6] H.-U. Yoon, S.-H. Whang, D.-W. Kim, and K.-B. Sim, "Strategy of cooperative behaviors for distributed autonomous robotics systems," in *Proc. of the 10th International Symposium on Artificial Life and Robotics*, 2005, pp. 151-154.

[7] H.-U. Yoon and K.-B. Sim, "Hexagon-based Q-learning for object search with multiple robots," in *Proc. of the 1st International Conference on Natural Computation*, 2005, pp. 713-722.

In-Hun Jang

In-Hun Jang received his B.S. and M.S. degrees in the Department of Control and Instrumentation Engineering from Chung-Ang University in 1993 and 1999 respectively. He is currently Doctor course in the School of Electrical and Electronics Engineering from Chung-Ang University, Korea. His research interests include artificial brain, intelligent robot, smart home, and home network etc.

E-mail : inhun@wm.cau.ac.kr

Dong-Hoon Lee

Dong-Hoon Lee received his B.S. degree in the Department of Electrical and Control Engineering from Sunchon National University, Korea, in 2005. He is currently in a Master's Program in the Chung-Ang University, Korea. His research interests are in machine Learning, distributed autonomous robotic system, intelligent robotic system applications etc.,

E-mail : sky52929@wm.cau.ac.kr

Kwee-Bo Sim

Kwee-Bo Sim received his B.S. and M.S. degrees in the Department of Electronic Engineering from Chung-Ang University, Korea, in 1984 and 1986 respectively, and Ph.D. degree in the Department of Electrical Engineering from The University of Tokyo, Japan, in 1990. Since 1991, he has been a faculty member of the School of Electrical and Electronics Engineering at Chung-Ang University, where he is currently a Professor. His research interests are in artificial life, intelligent robot, intelligent system multi-agent system, distributed autonomous robotic system, machine learning, adaptation algorithm, soft Computing(neural network, fuzzy system, evolutionary computation), artificial immune systems, evolvable hardware, artificial brain, intelligent home, home networking, intelligent sensor, and ubiquitous computing etc. He is a member of IEEE, SICE, RSJ, KITE, KIEE, KFIS, and ICASE Fellow. He is currently Editor-in-Chief of the International Journal of KFIS and Chief of vice President of the KFIS.

Phone : +82-2-820-5319
Fax : +82-2-817-0553
E-mail : kbsim@cau.ac.kr
http://alife.cau.ac.kr