Expert Systems with Applications 41 (2014) 1663–1671

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

A variable size mechanism of distributed graph programs and its performance evaluation in agent control problems

Shingo Mabu [a,*], Kotaro Hirasawa [b], Masanao Obayashi [a], Takashi Kuremoto [a]

[a] Graduate School of Science and Engineering, Yamaguchi University, Tokiwadai 2-16-1, Ube, Yamaguchi 755-8611, Japan
[b] Information, Production and Systems Research Center, Waseda University, Hibikino 2-2, Kitakyushu, Fukuoka 808-0135, Japan

ARTICLE INFO

Keywords:
Evolutionary computation
Directed graph
Distributed structure
Variable size
Reinforcement learning
Decision making

ABSTRACT

Genetic Algorithm (GA) and Genetic Programming (GP) are typical evolutionary algorithms using string and tree structures, respectively, and there have been many studies on the extension of GA and GP. How to represent solutions, e.g., strings, trees, graphs, etc., is one of the important research topics and Genetic Network Programming (GNP) has been proposed as one of the graph-based evolutionary algorithms. GNP represents its solutions using directed graph structures and has been applied to many applications. However, when GNP is applied to complex real world systems, large size of the programs is needed to represent various kinds of control rules. In this case, the efficiency of evolution and the performance of the systems may decrease due to its huge structures. Therefore, we have been studied distributed GNP based on the idea of divide and conquer, where the programs are divided into several subprograms and they cooperatively control whole tasks. However, because the previous work divided a program into some subprograms with the same size, it cannot adjust the sizes of the subprograms depending on the problems. Therefore, in this paper, an efficient evolutionary algorithm of variable size distributed GNP is proposed and its performance is evaluated by the tileworld problem that is one of the benchmark problems of multiagent systems in dynamic environments. The simulation results show that the proposed method obtains better fitness and generalization abilities than the method without variable size mechanism.

1. Introduction

Genetic Algorithm (GA) (Holland, 1975) and Genetic Programming (GP) (Koza, 1992, 1994) are typical evolutionary algorithms that have been widely studied. A large number of real world applications have been also studied such as robot programming (Kamio & Iba, 2005), financial problems (Alfaro-Cid et al., 2008; Iba & Sasaki, 1999; Ruiz-Torrubiano & Suárez, 2010) and network security systems (Banković, Stepanović, Bojanić, & Nieto-Taladriz, 2007; Folino, Pizzuti, & Spezzano, 2005).

In order to create reliable systems using evolutionary algorithms, the program structures (phenotype and genotype representations) and how to efficiently evolve them are important issues. Therefore, as an extended algorithm of GA and GP, Genetic Network Programming (GNP) and its extension using reinforcement learning (GNP-RL) (Mabu, Hirasawa, & Hu, 2007; Hirasawa, Okubo, Katagiri, Hu, & Murata, 2001) have been proposed and applied to many applications (Mabu, Chen, Lu, Shimada, & Hirasawa, 2011; Hirasawa, Eguchi, Zhou, Yu, & Markon, 2008). The program of GNP is represented by directed graph structures and evolved by crossover and mutation. Originally, GNP was proposed because graph structures may have better representation abilities than strings and trees. In addition, human brain also has a graph (network) structure, so some inherent abilities may be involved in the graph structure. In fact, the graph structure has some advantages such as (1) reusability of nodes and (2) applicability to dynamic environments. Actually, GNP has the following features. (1) The directed graph structure automatically generates some repetitive processes like subroutines, and reuses nodes repeatedly during the node transition, which contributes to creating programs with compact structures. (2) Once GNP starts its node transition from the start node, GNP executes judgment nodes (if–then functions) and processing nodes (action functions) according to the connections between nodes without any terminal nodes. Therefore, the node transition implicitly memorizes the history of judgment and actions, which contributes to the decision making in dynamic environments because GNP can make decisions based not only on the current, but also the past information.

Evolutionary Programming (EP) (Fogel, Owens, & Walsh, 1966; Fogel, 1994) is a graph-based evolutionary algorithm to create finite state machines (FSMs) automatically, but the characteristics of EP and GNP are different. Generally, FSM must define the state transition rules for all the combinations of states and possible inputs, thus the FSM program will become large and complex when the number of states and inputs is large. On the other hand, the

* Corresponding author. Tel./fax: +81 836 85 9519.
E-mail address: mabu@yamaguchi-u.ac.jp (S. Mabu).

0957-4174/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.eswa.2013.08.063

evolution of GNP selects only the necessary nodes by changing connections between nodes, which means that GNP can judge only the essential inputs for making decisions at the current situations. As a result, GNP does not have to consider all the combinations of the inputs and actions, which makes the compact program structures.

When GNP is applied to complicated real world systems, large size of graph structures are needed to represent various kinds of rules for adapting to various kinds of situations. However, huge structures may decrease the efficiency of evolution, as a result, decrease the performance of the systems. Therefore, we have been studied distributed GNP (Yang, He, Mabu, & Hirasawa, 2012) based on the idea of divide and conquer which is used to create the complicated systems using relatively small size of the programs (Li, Tian, & Sclaroff, 2012). In Yang et al. (2012), the programs are divided into several subprograms which cooperatively control whole tasks, and this method is applied to a stock trading model, which shows better performances than the method without distributed structures. However, because the previous work divided a program into some subprograms with the same size, it cannot adjust the sizes of the subprograms depending on the problems. The best size of the structure is different depending on the difficulties of the task, thus the fixed size of the structure limits the representation ability of the solutions. In order to solve this problem, an efficient evolutionary algorithm of variable size distributed GNP (VS-DGNP) is proposed and its performance is evaluated by the tileworld problem (Pollack & Ringuette, 1990) that is one of the benchmark problems of multiagent systems in dynamic environments. The features of the proposed method are as follows.

The complicated structure can be divided into several substructures with different sizes. Genetic operations can be executed inside each substructure and between substructures, respectively, as a result, the efficient optimization can be done. Migration of nodes from a certain substructure to another substructure is executed to optimize the sizes of the substructures appropriately.

The rest of this paper is organized as follows. In Section 2, the basic structure of GNP and GNP with reinforcement learning (GNP-RL) is reviewed. In Section 3, the structure of distributed GNP and how to realize variable size structure are explained. In Section 4, after simulation environments and conditions are explained, the results and analysis are described. Section 5 is devoted to conclusions.

2. Review of genetic network programming with reinforcement learning

Because the proposed Variable Size Distributed GNP (VS-DGNP) is based on GNP with Reinforcement Learning (GNP-RL), the structure of GNP-RL and its learning and evolution mechanisms are firstly reviewed. An individual of GNP-RL is represented by a directed graph structure, evolved by crossover and mutation, and the node transition rules are learned by reinforcement learning.

2.1. Basic structure of original GNP

Before explaining the detailed structure of GNP-RL, the basic structure of original GNP is introduced (Fig. 1). It consists of a number of judgment nodes and processing nodes and one start node. The number of nodes are determined in advance depending on the complexity of the problems. Each judgment node has a if–then branch decision function and each processing node has an action function. For example, a judgment node examines a sensor input of agents, and a processing node determines agent actions, e.g., go forward, turn left, turn right, etc. The role of the start node is to determine the first node to be executed. Therefore, the node transition starts from the start node, and a sequence of judgments and processing is created by the connections between nodes and judgment (if–then) results.

2.2. Additional components in GNP-RL: subnodes

The difference between the original GNP and GNP-RL is as follows. GNP-RL has subnodes in each judgment and processing node as shown in Fig. 2. In the original GNP, one function is assigned to each node and executed when the node is visited. In GNP-RL, several functions (two functions in Fig. 2) are assigned as subnodes and one of them is selected and executed by a reinforcement learning algorithm. Therefore, reinforcement learning selects better function/subnode for each node and the route of the node transition is optimized. For example, subnode 1 in Fig. 2(a) has a judgment function of "judge forward", and subnode 2 has that of "judge right". Subnode 1 in Fig. 2(b) has a processing function of "go forward", and subnode 2 has that of "turn left".

2.3. Genotype representation

The graph structure is realized by the combination of gene structures shown in Fig. 3. NT_i shows the node type of node i. $NT_i = 0$ means start node, $NT_i = 1$ means judgment node, and $NT_i = 2$ means processing node. d_i is a time delay which shows the time spent on the execution of node i. In this paper, d_i of judgment node is set at 1 and that of processing node is set at 5. The time delay is useful to fix the maximum number of nodes to be executed in each action step. In this paper, one action step is defined as a certain time units that an agent can use for executing judgments and processing. If five time units (time delay) are assigned to one action step, the action step ends when the time units used by the node transition become five or exceed five. That is, GNP can execute "less than five judgments nodes and one processing node" in one action step to determine an action. If "five judgment nodes" are else executed before visiting a processing node, one action step ends without taking any actions. The explanation on the time delay is explained in Mabu et al. (2007) in more detail.

Q_{i1},Q_{i2}, ... are Q values (Sutton & Barto, 1998) assigned to the subnodes in node i. ID_{i1},ID_{i2}, ... are the node functions of the subnodes. Q value estimates the sum of the discounted rewards obtained in the future. The contents of the functions are described in the node function library. For example, $NT_i = 1$ and $ID_{i2} = 2$ show that the function of subnode 2 in node i is J_2. Node functions used in the simulations of this paper are shown in Table 1 in Section 4.1.

$C^A_{i1}, C^B_{i1}, \ldots$ and $C^A_{i2}, C^B_{i2}, \ldots$ show the next node numbers connected from node i. For example, subnode 1 in node i is connected to $C^A_{i1}, C^B_{i1}, \ldots$. The superscripts A and B correspond to the judgment results, i.e., if the judgment result is A at subnode 1, the next node becomes C^A_{i1}.

2.4. Node transition and its learning algorithm

In this paper, Sarsa (Sutton & Barto, 1998) which is one of the reinforcement learning algorithms is used for the learning of GNP-RL. The reason why Sarsa is selected is that on-policy algorithm (Sarsa) is effective to optimize state transitions considering the effect of ε – greedy policy used in this paper. The node transition of GNP-RL is carried out as follows.

When the current node is a judgment node, one of the subnodes is selected according to ε – greedy policy, i.e., the subnode with the largest Q value is selected with the probability of $1 - \varepsilon$, otherwise one subnode is randomly selected. After executing the function of the selected subnode, the current node is transferred to the next node according to the judgment result and connection. In Fig. 2(a), suppose subnode 1 is selected, then one of the connec-