Applied Soft Computing xxx (2014) xxx–xxx

Contents lists available at ScienceDirect

Applied Soft Computing

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

Continuous probabilistic model building genetic network programming using reinforcement learning

Xianneng Li[a,b], Kotaro Hirasawa[a,b,*]

[a] Graduate School of Information, Production and Systems, Waseda University, Hibikino 2-7, Wakamatsu-ku, Kitakyushu, Fukuoka 808-0135, Japan
[b] Information, Production and Systems Research Center, Waseda University, Hibikino 2-7, Wakamatsu-ku, Kitakyushu, Fukuoka 808-0135, Japan

A R T I C L E I N F O

Article history:
Received 17 April 2013
Received in revised form
13 September 2014
Accepted 18 October 2014
Available online xxx

Keywords:
Estimation of distribution algorithm
Probabilistic model building genetic network programming
Continuous optimization
Reinforcement learning

A B S T R A C T

Recently, a novel probabilistic model-building evolutionary algorithm (so called estimation of distribution algorithm, or EDA), named probabilistic model building genetic network programming (PMBGNP), has been proposed. PMBGNP uses graph structures for its individual representation, which shows higher expression ability than the classical EDAs. Hence, it extends EDAs to solve a range of problems, such as data mining and agent control. This paper is dedicated to propose a continuous version of PMBGNP for continuous optimization in agent control problems. Different from the other continuous EDAs, the proposed algorithm evolves the continuous variables by reinforcement learning (RL). We compare the performance with several state-of-the-art algorithms on a real mobile robot control problem. The results show that the proposed algorithm outperforms the others with statistically significant differences.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

Despite the selection operator based on the concept of "Survival-of-the-fittest", classical Evolutionary Algorithms (EAs) generally evolve the population of candidate solutions by the random variation derived from biological evolution, such as crossover and mutation. However, numerous studies report that the results of EAs strongly rely on the configurations of the parameters associated with the stochastic genetic operators, such as crossover/mutation rate. For concrete problems, the parameter settings generally vary. Hence, the parameter tuning itself becomes an optimization problem. Meantime, the stochastic genetic operators sometimes may not identify and recombine the building blocks (BBs, defined by high-quality partial solutions) correctly and efficiently due to the implicit adaptation of the *building block hypothesis* (BBH) [1,2], which causes the problems of premature convergence and poor evolution ability. These reasons have motivated the proposal of a new class of EAs named estimation of distribution algorithm (EDA) [3], which has received much attention in recent years [4,5]. As the name implies, EDA focuses on estimating the probability distribution of the population using statistic/machine learning to construct a probabilistic model. Despite the selection operator which is also used to select the set of promising samples for the estimation of probability distribution, EDA replaces the crossover and mutation operators by sampling the model to generate new population. By explicitly identifying and recombining the BBs using probabilistic modeling, EDA has drawn its success to outperform the conventional EAs with fixed, problem-independent genetic operators in various optimization problems.

Numerous EDAs has been proposed, where there are mainly three ways to classify the existing EDAs. (1) From the model complexity viewpoint, EDAs can be mainly classified into three groups [6]: univariate model, pairwise model and multivariate model, which identify the BBs of different orders. Univariate model assumes there is no interactions between the elements,[1] hence constructing the probabilistic model by marginal probabilities to identify BBs of order one. Similarly, pairwise and multivariate models use more complex methods to model BBs of order two and more. One can easily observe that estimating the distribution is not an easy task and modeling more accurate model generally requires higher computational cost [4,7]. (2) From the perspective of individual structures, EDA can mainly be classified into two groups,

* Corresponding author at: Graduate School of Information, Production and Systems, Waseda University, Hibikino 2-7, Wakamatsu-ku, Kitakyushu, Fukuoka 808-0135, Japan. Tel.: +81 93 692 5261; fax: +81 93 692 5261.
E-mail addresses: sennou@asagi.waseda.jp (X. Li), hirasawa@waseda.jp (K. Hirasawa).

[1] The elements refer to the variables/alleles in genetic algorithm (GA), or nodes in genetic programming (GP).

http://dx.doi.org/10.1016/j.asoc.2014.10.023
1568-4946/© 2014 Elsevier B.V. All rights reserved.

which are probabilistic model building genetic algorithm (PMBGA) [8] and PMB genetic programming (PMBGP) [9]. PMBGA studies the probabilistic modeling using GA's bit-string individual structures. PMBGP explores EDA to tree structures which provide more complex ways to represent solutions for program evolution. (3) For different problem domains, EDA can be grouped into discrete EDAs and continuous EDAs, which solve the optimization problems of discrete domain [4,8] and continuous domain [10–13].

A novel EDA, called probabilistic model building genetic network programming (PMBGNP), was recently proposed [14–16]. PMBGNP is inspired by the classical EDAs, however, a distinguished directed graph (network) structure [17–21] is used to represent its individual. Hence, it can be viewed as a graph EDA that extends conventional EDAs like bit-string structure based PMBGA and tree-structure based PMBGP. The fundamental points of PMBGNP are:

1. PMBGNP allows higher expression ability by means of graph structures than conventional EDAs.
2. Due to the unique features of its graph structures, PMBGNP explores the applicability of EDAs to wider range of problems, such as data mining [22,14,23] and the problems of controlling the agents' behavior (agent control problems) [16,24–26].

In the previous research, it has been demonstrated that PMBGNP can successfully outperform classical EAs with the above problems.

However, PMBGNP is mainly designed for discrete optimization problems. In other words, it cannot deal with (or directly handle) continuous variables which are widely existed in many real-world control problems. To solve this problem, the simplest way is to employ discretization process to transfer the continuous variables into discrete ones, however, which will cause the loss of solution precision.

This paper is dedicated to an extension of PMBGNP to continuous optimization in agent control problems. Different from most of the existing continuous EDAs developed by incremental learning [10], maximum likelihood estimation [11], histogram [27] or some other sorts of machine learning techniques [28–32], the proposed algorithm employs the techniques of reinforcement learning (RL) [33], such as actor critic (AC), as the mechanism to estimate the probability density functions (PDFs) of the continuous variables. Although most of the classical continuous EDAs formulate the PDFs of continuous variables by Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$, the proposed algorithm applies AC to calculate the temporal-difference (TD) error to evaluate whether the selection (sampling) of continuous values is better or worse than expected. Based on the idea of trial-and-error, a scalar reinforcement signal which can decide whether the tendency to select the sampled continuous value should be strengthened or weakened is formulated by the gradient learning for the evolution of Gaussian distribution (μ and σ).

Most importantly, as an extension of PMBGNP, the proposed algorithm mainly possesses the ability to solve the agent control problems, rather than the conventional continuous EDAs only for function optimization problems. Accordingly, the applicability of continuous EDAs is explored in certain degrees.

In this paper, the proposed algorithm is applied to control the behavior of a real autonomous robot, Khepera robot [34,35], in which the robot's wheel speeds and sensor values are continuous variables. To evaluate the performance of this work, various classical algorithms are selected from the literature of standard EAs, EDA and RL for comparison.

The rest of this paper is organized as follows. Section 2 briefly introduces the original framework of PMBGNP in the discrete domain. In Section 3, extending PMBGNP to continuous domain is explained in details. The experimental study is shown in Section 4. Finally we conclude this paper in Section 5.

2. Probabilistic model building genetic network programming

2.1. Directed graph (network) structure

From the explicit viewpoint, PMBGNP distinguishes itself from the classical EDAs by using a unique directed graph (network) structure to represent its individual, depicted in Fig. 1. The directed graph structure is originally proposed in a newly graph-based EA named Genetic Network Programming (GNP) [17,18,36]. Three types of nodes are created to form the program (individual) of GNP:

- *Start node*: it has no function and conditional branch.
- *Judgment node*: it has its own judgment function and multiple conditional branches.
- *Processing node*: it has its own processing function but no conditional branch.

Each program is composed of one start node, multiple judgment and processing nodes. Start node only plays the role on deciding the first node to be executed. Judgment nodes imitate the "if-then" decision-making functions to deal with the specific inputs of the problems, such as the sensor values of the robot. Processing nodes enforce the action functions for task solving, such as determining the wheel speeds of the robot. By separating judgment and processing functions, the distinguished directed graph can deal with various combinations of judgments and processing to efficiently evolve the compact programs by only selecting the necessary judgments and processing. Such separation and selection by necessity can efficiently generate partially observable markov decision process (POMDP) [36] and ensure high generalization ability. The number of judgment and processing nodes is defined in advance and problem specific. As a result, the directed graph never causes the bloat problem of GP. Although a small number of nodes is prepared, such a structure can obtain good performance by well realizing the repetitive process by the frequent reuse of nodes.

More empirically, the directed graph can be encoded into bit-strings as shown in Fig. 1, which is defined by a tuple

$$G = (N_{\text{node}}, B, \text{LIBRARY}),$$

where N_{node} and B are the sets of nodes and branches in an individual, respectively; *LIBRARY* is a set of judgment and processing functions given by the tasks. Each node $i \in N_{\text{node}}$ is defined by a tuple [2]

$$i = (NT_i, NF_i, B(i), C_i).$$

NT_i defines the node type, where 0, 1 or 2 for start, judgment or processing node, respectively. $NF_i \in$ LIBRARY represents its function. $B(i)$ represents its set of branches. C_i consists of a set of C_{ik} indicating the node connected from the k_{th} branch of node i.

In standard GNP, NT_i, NF_i and $B(i)$ are generally unchanged, while evolution is carried out to change C_i, which means that the task of evolution is to find the optimal solution $g^* \in G$ by evolving the node connections.

In GNP, since the predefined and unchanged start node without function is only used to determine the first node to be executed, the start node and its branch are not considered in the formulation of G for simplicity.

[2] V_i and x_i denote the state-value and continuous variables of node i, which will be described in the next section. They are not included in the discrete PMBGNP.