Applied Mathematical Modelling 39 (2015) 3150–3163

Contents lists available at ScienceDirect

Applied Mathematical Modelling

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

Solving differential equations with ant colony programming

M.Z.M. Kamali [a], N. Kumaresan [b,*], Kuru Ratnavelu [b]

[a] Centre for Foundation Studies in Science, University of Malaya, 50603 Kuala Lumpur, Malaysia
[b] Institute of Mathematical Sciences, University of Malaya, 50603 Kuala Lumpur, Malaysia

ARTICLE INFO

Article history:
Received 1 October 2012
Received in revised form 13 August 2014
Accepted 3 November 2014
Available online 27 November 2014

Keywords:
Ordinary differential equation
Nonlinear ordinary differential equation
Partial differential equation
Ant colony programming

ABSTRACT

Differential equations are widely used to model physical phenomena in the real world. In the present work, a nontraditional modified ant colony programming (ACP) method is implemented. The modified ACP algorithm is unique as it does not use the criteria of distance. It utilizes the probability function which related to the quantity of the pheromone level in the ACP. The motivation in this present work is to describe the consistency and the applicability of the nontraditional ant colony programming (ACP) method in solving various ordinary differential equations (ODE's) and partial differential equations (PDE's) problems. Comparatively, similar and exact solution is achieved by using the ACP approach. Illustrative numerical example as well as tables are presented for comparison purpose.

1. Introduction

With vast advanced in computer technology, a lot of different methods have been developed for solving ordinary differential equations (ODE's) and partial differential equations (PDE's). Among the methods that have been applied and used for solving these types of equations are the Runge–Kutta, radial basis function [1], genetic programming [2] and feedforward neural network [3]. Some of these methods compute solution in an array form which contains the value of the solution whereas others apply the basis functions to represent the exact solution and convert the original problem into a system of algebraic equations. In the feedforward neural network, the ODE's and PDE's depend on the function approximation capabilities. In order to obtain the solution, the feedforward neural network is trained to minimize the suitable error function, by employing the optimization techniques. Another alternative method, is the genetic programming (GP). It is an optimization process which was based on the large number of possible solutions through genetic operations such as mutation, crossover and replication. This technique forms generations of trial solutions expressed in an analytical closed form. In most of the problems, the exact solution can be obtained but there are cases when the solution cannot be expressed in an analytical form. When this happens, the need to get an approximate answer with controlled level of accuracy will be produced.

Currently there have been a surge of interest in automatic programming. This area of research is focusing on generating computer programs automatically. Koza was the first to propose Genetic Programming (GP) [4,5] which is the common example of automatic programming. Koza mentioned the five initial steps which should be fulfilled before searching for a program and this five basic steps are.

- Selection of terminal symbols.
- The choice of functions.

* Corresponding author.
E-mail addresses: mzmk2005@gmail.com (M.Z.M. Kamali), drnk2008@gmail.com (N. Kumaresan).

http://dx.doi.org/10.1016/j.apm.2014.11.003
0307-904X/

- Fitness function specification.
- Selection of certain parameters for controlling the run.
- Defining terminal criteria.

Many of these principles used in GP are almost similar and can be adapted to ACP. Therefore Boryczka [6–8], applied ACP as an alternative method for automatic programming with two different techniques: the expression and the program approach. In the expression approach, the quest for an approximating function is constructed in the form of an arithmetic expression. These expressions are in prefix notation. In the second technique, a sequence of assignment instructions is used in order to build the expression which can evaluate the function. Both techniques are based on a space graph whose nodes are represented by variables, functions and constants in the expression approach where as in the program approach, the nodes of the graph are represented by the assignment instructions. Although both approach had showed some promising results but the task for generating more general types of program is still very difficult. Other ACO algorithms that have been extended and used for solving symbolic regression problems, are ant programming (AP) [9], generalized ant programming (GAP) [10] and the dynamic ant programming (DAP) [11]. In this paper, the ACP is used to solve a set of differential equations (ODE's) and partial differential equations (PDE's). The main objective of this present work is to generate expressions for searching the analytical solutions for ODEs and nonlinear differential equations by using the ACP. Normally ACP studies depend on the distance as one of the important parameters. However in this work, the current ACP algorithm does not depend on the distance in the calculations. The present algorithm depends mainly on the probability function which is related to the quantity of the pheromone level. The ACP is equally comparable with the GP method presented by Tsoulos and Lagaris [12]. The ACP gives exact solutions within reasonable computational time.

2. Ant colony programming

The ant colony programming (ACP) is a stochastic approach which is implemented on a space graph [13]. The nodes of the graph are represented by variables, functions and constants. Functions are represented in terms of arithmetic operators, operands as well as Boolean functions. The set of functions defining a given problem is called a function set $\mathbb{F}$ and the collection of variables and constants to be used are known as the terminal set $\mathbb{T}$. In the ant colony algorithm, ants start their journey by wandering randomly and upon finding food return to their colony while depositing pheromone trails. If other ants find such a path which lead them to a food source from the colony, they will not travel randomly. Instead, they will follow the trail, returning and reinforcing it. As the time goes by, the pheromone trail starts to evaporate and this reduced its attractive strength. The idea is, if an ant takes more time to travel down the path and back again, the chance for the pheromone to evaporate faster is higher. Shorter path which is more frequently used by the ants, will increase the density of the pheromone, slowing down the evaporation process. Pheromone evaporation also helps from obtaining convergence to a locally optimal solution. If there are no evaporation at all, the paths chosen by the first ant would tend to be excessively attractive to the following ones. In that case the exploration of the solution space would be constrained. Thus when one ant finds a short path from the colony to a food source, other ants are more likely to follow that path, and positive feedback eventually leads to all the ants following a single path. The idea of the ant colony algorithm is to mimic this behavior with "simulated ants" walking around the graph representing the problem to solve. The ACP is implemented to generate a set of arithmetic expressions for solving ordinary differential equations (ODE's) and partial differential equations (PDE's). If the number of expression satisfies the fitness function, then it will become the optimal solution. The flowchart for the ACP is given in Fig. 1. The four basic steps are listed below which are vital for the searching process based on Boryczka and Wiezorek [14].

- Choice of terminals and functions.
- Construction of graph.
- Defining fitness function.
- Defining terminal criteria.

2.1. Terminals and functions

In the ACP approach, the symbols t_i and the functions f_i are shown in Table 1. The functions f_i can be evaluated as an arithmetic operator, arithmetic function and an arbitrarily defined function appropriate to the problem under consideration. The terminal symbols and functions have the power to express the solution to a problem based on the composition of functions and terminals specified. The terminal symbol or functions are being presented by using the power (arity) and this is given in Table 2.

The present work, included two new properties under this terminal symbols, the open bracket '(' and the close bracket ')'. The inclusion of this two new properties are vital in order to differentiate with the use of multiplication symbol '$*$'. In the work reported by Kumaresan and co-workers [13], the '$*$' symbol is used not only for representing multiplication operations but also for showing the implementations of bracket in the expressions. For example, the tour of an ant which generated an expression: $e*t + 1* + 5$ is actually representing an equation $e(t + 1) + 5$. If the number of nodes are increased, the expression