ELSEVIER

Computers and Chemical Engineering 25 (2001) 257–266

Computers & Chemical Engineering

www.elsevier.com/locate/compchemeng

Evolutionary algorithms approach to the solution of mixed integer non-linear programming problems

Lino Costa, Pedro Oliveira *

Department of Production and Systems Engineering, University of Minho, 4710 Braga, Portugal

Received 7 January 2000; received in revised form 25 October 2000; accepted 25 October 2000

Abstract

The global optimization of mixed integer non-linear problems (MINLP), constitutes a major area of research in many engineering applications. In this work, a comparison is made between an algorithm based on Simulated Annealing (M-SIMPSA) and two Evolutionary Algorithms: Genetic Algorithms (GAs) and Evolution Strategies (ESs). Results concerning the handling of constraints, through penalty functions, with and without penalty parameter setting, are also reported. Evolutionary Algorithms seem a valid approach to the optimization of non-linear problems. Evolution Strategies emerge as the best algorithm in most of the problems studied. © 2001 Elsevier Science Ltd. All rights reserved.

Keywords: Evolutionary algorithms; Genetic algorithms; Evolution strategies; Mixed integer non-linear programming

1. Introduction

Real world problems can, in general, be formulated as mixed integer non-linear programming problems (MINLP). These problems, due to their combinatorial nature, are considered difficult problems. Gradient optimization techniques have only been able to tackle special formulations, where continuity or convexity had to be imposed, or by exploiting special mathematical structures. Approaches based on stochastic algorithms have been used. These approaches, also know as adaptive random search, have successfully tackle MINLP, mostly in the area of chemical engineering (Reklaitis, Ravindran, & Ragsdell, 1983; Salcedo, 1992; Banga & Seider, 1996). In recent years, a vast amount of work has been published on applications of evolutionary algorithms (Genetic Algorithms, Evolution Strategies, Simulated Annealing, etc.) to the solution of MINLP in many engineering applications. These algorithms are distinct from conventional algorithms since, in general, only the information regarding the objective function is required. Moreover, they start from a pool of points that evolves over time, possibly in the direction of the optimum. It should be stressed that the objective of this on going research is not to find the 'best' algorithm for all problem instances, but to compare different algorithms, in order to find out classes of problems which may be more suitable for certain algorithms than others.

The general formulation of the problem is as follows:

$$\begin{aligned} &\min f(x) \\ &\text{subject to} \\ &h(x) = 0 \\ &g(x) \leq 0 \\ &x_j \text{ integer}, \quad j \in I \\ &x \in X = \{x | x \in R^n, \quad x^l \leq x \leq x^u\} \end{aligned} \tag{1}$$

where

$$h \in R^m, \quad g \in R^p.$$

In this work, 7 test problems, proposed by independent authors, are studied using Genetic Algorithms (GAs) and Evolution Strategies (ESs). These problems arise from the area of chemical engineering, and represent difficult non-convex optimization problems, with continuous and discrete variables. Comparisons are made with an M-SIMPSA algorithm (Cardoso, Salcedo, & Feyo de Azevedo, 1996a,b; Cardoso, Salcedo,

* Corresponding author. Fax: +351-53-253604741.
E-mail address: pno@dps.uminho.pt (P. Oliveira).

PII: S0098-1354(00)00653-0

Feyo de Azevedo, & Barbosa, 1997; Cardoso, 1998) based on the combination of the non-linear simplex method of Nelder and Mead and Simulated Annealing.

2. Description of the genetic algorithm

2.1. Introduction

Genetic Algorithms are search algorithms that mimic the process of natural selection (Goldberg, 1989). Thus, unlike conventional algorithms, GAs start from a pool of points, usually referred to as chromosomes. In order to implement a simple GA it is necessary to define the representation of the search space (usually a binary representation) and an evaluation function which permits the comparison between the different chromosomes in terms of their fitness. New points in the search space are generated by the application of genetic operators, thus originating a new generation of points.

Fig. 1 presents the GA flow chart. An initial population of bit strings (chromosomes) is generated. Each of these strings is then evaluated. The selection operator assures that strings are copied to the next generation with a probability associated to their fitness values. Therefore, this operator mimics the survival of the fittest in the natural world. Although selection assures that in the next generation the best chromosomes will be present with a higher probability, it does not search the space, because it just copies the previous chromosomes. The search results from the creation of new chromosomes from old ones. The crossover operator, takes two randomly selected chromosomes; one point along their common length is randomly selected, and the characters of the two parent strings are swapped, thus generating two new chromosomes. Crossover by itself does not involve any change in the actual values of the chromosomes. The mutation operator, randomly selects a position in the chromosome and, with a given probability, changes the corresponding value. This operator does assure that new parts of the search space are explored, which selection and crossover could not fully guarantee. The GA proceeds as the following Algorithm (Fig. 1):

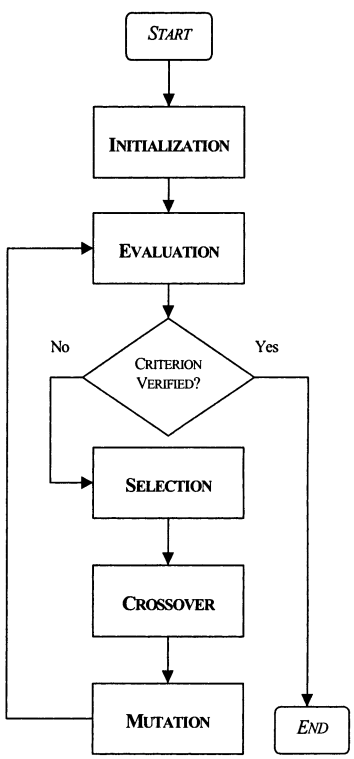

Fig. 1. Genetic algorithm flow chart.

Algorithm 1 (*Genetic algorithm*).

1. *Initialization* The initial population, consisting of points in the search space, is randomly created.
2. *Evaluation* Each chromosome in the population is evaluated through the objective (fitness) function.
3. *Genetic operators* The search is performed, by creating a new population from the previous one, through the application of the genetic operators.
4. *Stopping criterium* These steps (2–3) are repeated till the population converges or the specified number of generations is reached.

2.2. Constraint handling

Most of GAs implementation on constrained optimization use the penalty function method (Goldberg, 1989), in such a way that, the fitness function $F(x)$ is defined as follows,

$$F(x) = f(x) + R\left(\sum_{k=1}^{p} [\max\{0, g_k(x)\}]^2 + \sum_{l=1}^{m} [h_l(x)]^2\right) \quad (2)$$

i.e. as the sum of $f(x)$, the objective function and two penalty terms denoting the equality and inequality constraints violations, where R is a penalty coefficient.

This approach has a major drawback. The setting of the penalty parameters usually requires extensive experimentation. In this work, we compare this approach with the constraint handling method proposed by Deb (1998). A penalty term is introduced which does not depend on a penalty parameter. Thus, the fitness function is defined as

$$F(x) = \begin{cases} f(x) & \text{if } g(x) \le 0 \text{ and } h(x) = 0 \\ f_{\max} + \left(\sum_{k=1}^{p} [\max\{0, g_k(x)\}] + \sum_{l=1}^{m} |h_l(x)|\right) & \text{otherwise} \end{cases} \quad (3)$$