Differential evolution in shortest path problems

Pedro Guerreiro
University of Algarve
Faro (Portugal)
pmguerre@ualg.pt

Mário Jesus
University of Algarve
Faro (Portugal)
mjesus@ualg.pt

Alberto Márquez
University of Seville
Sevilla (Spain)
almar@us.es

Abstract

This paper proves that the Differential Evolution (DE) algorithm is valid to solve the Shortest Path (SP) problem in random, median sized networks. From the trials, we have obtained an 9% accuracy, in the worst case scenario.

1 Introduction

Differential Evolution (DE) is an Evolutionary Algorithm (EA) introduced by Storn and Price [9, 10] and, although the authors primary applications were to solve real continuous space problems, soon there were several versions of DE for integer, discrete and combinatorial problems [5, 8, 7]. DE is an algorithm that is emerging very rapidly amongst other EA because it is very powerful and very simple to implement, when compared to other EA.

In this paper we will demonstrate the usability of DE in a permutation-based combinatorial optimization problem, namely, determining the Shortest Path (SP) between two nodes in a network. The shortest path is a combinatorial problem that has several deterministic algorithms for solving it, for example, the Dijkstra, the Bellman-Ford or the A* algorithms [3, 2, 4], just to name a few.

We will focus to determining and estimate the shortest path between two nodes in a network, composed of D nodes and E edges. As each edge has a certain weight, we pretend to minimize the total sum of the weights of the edges from the source node to the destination node. Note that in the SP problem, as opposed for instance, to the Travelling Salesman Problem (TSP), the number of nodes in the path can be variable. This, as far as we know, has never been addressed in any DE variant.

The rest of this papers is arranged as follows: Section 2 presents the classic DE algorithm, as introduced in [9] and Section 3 presents our version of DE to Combinatorial Optimization. Section 4 provides the discussion of some results and Section 5 have the conclusions and future work.

2 Classic Differential Evolution

DE is a relatively new member of the EA family, introduced in 1995 by Storn and Price [9, 10], and since then, it widespread very quickly, mainly due to its simple implementation and to be very powerful. It has also very few control parameters, just two (F and CR), which makes it simple to tune: F is a real constant factor parameter, usually $\in [0, 2]$, and CR is the crossover rate $\in [0, 1]$.

DE was first introduced to work on real continuous spaces, and has a few variants, introduced in [10], using the notation $DE/x/y/z$, where x specifies the vector to be mutated, normally "*rand*" for a random vector selected from the current population or "*best*" for the best vector from the current population, y is the number of difference vectors, and z defines the crossover scheme, usually "*bin*" for a binomial crossover, or "*exp*" for exponential crossover. We will use the variant known as $DE/rand/1/bin$ to explain the basic algorithm of DE.

J. Cáceres, C. Grima, A. Márquez, M.L. Puertas (eds.); Avances en Matemática Discreta en Andalucía1 (vol.2), Carmona (Sevilla), Noviembre 2011, pp. 1–6

The main objective of any EA is optimize a target function, usually defined in the form

$$\min\left(f(X)\right) : \mathbb{R}^n \to \mathbb{R} \tag{1}$$

where X is a vector defined as

$$X = (x_1, x_2, ..., x_n)\,, \tag{2}$$

being n is the number of elements of the vector. Each element is usually limited by a lower (x^L) and upper (x^U) bound, forming this way the search space, in the form

$$x_j^L \leq x_j \leq x_j^U\,,\ j = 1, ..., n. \tag{3}$$

As most EA, DE uses a population of vectors (or individuals, in the EA jargon), evolving its values in each iteration (or generation) until a goal is achieved.

In DE, the initial population is created using an uniform random generated population, according to

$$X = \{x \in \mathbb{R} : x_j^L \leq x_j \leq x_j^U\,,\ j = 1, ..., D\} \tag{4}$$

where x_j^L and x_j^U refers to the lower limit and upper limit for each element, and D is the number of elements (or chromosomes) of each individual.

Then, in each generation g, for each individual $x_{g,i}$ of the population, a mutant individual will be generated according to

$$v_{g+1,i} = x_{g,r_1} + F \cdot (x_{g,r_2} - x_{g,r_3}),\ i = 1, ..., NP \tag{5}$$

where $r_1, r_2, r_3 \in \{1, ..., NP\}$ are mutually different random individuals and different from the current index i, F is a real constant factor parameter $\in [0, 2]$ that controls the scale of the differential variation $(x_{g,r_2} - x_{g,r_3})$ and NP is number of individuals in the population. A lower value for F limits the focus of the search space to a short neighbourhood, and could trap the algorithm in a local optimum, but an high value could disperse the values.

Now the trial population is created, applying a crossover to each mutant individual, according to the formula

$$u_{g+1,i,j} = \begin{cases} v_{g+1,i,j} & \text{if } (rand(j) \leq CR) \text{ or } j = rand_i(k) \\ x_{g,i,j} & \text{otherwise} \end{cases},\ j = 1, ..., D;\ i = 1, ..., NP \tag{6}$$

where $rand(j)$ is the j^{th} evaluation of an uniform random generator $\in [0, 1[$, CR is the crossover rate constant parameter $\in [0, 1]$ and $rand_i(k)$ is a random index $\in [1, ..., D]$, generated for each i index, ensuring that in each trial individual, at least one chromosome is from the mutant individual, guaranteeing this way a variation in the population. A lower value for CR means that the individual will be very similar to the one from the current population, and a high value means that it will be similar to the mutant one.

The only step left is to select which individuals will "survive" to the next generation, and which will "die". DE does this in a simple manner: each individual of the trial population $u_{g+1,i}$ is compared with the corresponding individual of the current population $x_{g,i}$, and the best one proceeds to the next generation. Mathematically

$$x_{g+1,i} = \begin{cases} u_{g+1,i} & \text{if } f(u_{g+1,i}) \leq f(x_{g,i}) \\ x_{g,i} & \text{otherwise} \end{cases} \tag{7}$$

3 Differential Evolution in Combinatorial Optimization

Our approach to use DE in combinatorial optimization has evolved through different ideas and concepts, but has found a strong point in the concept of using an adaptation of the classic DE, and then penalize the invalid paths, that will certainly appear.

In EA, there are two classic ways of dealing with the constrains of a problem: whether correct them, or penalize their value in the objective function [6]. After some trial and error, we choose the second one, through a pre-process that consists in determining the shortest path between any two nodes in the network using the Dijkstra algorithm, and saving this values to posterior use. This values are not used anywhere in DE, except in the evaluation function, as a penalty factor for the invalid paths, and are calculated previously to speed up the algorithm. We also use the values to have the optimum value to compare to the one obtained in DE.

Also, we give each node a number $\in [1, ..., D]$, D being the number of nodes the the network, and this number will be used as the value of the node to optimize.

Our algorithm starts by generating a uniform random population, in the following manner: we first create a pool of values $\in [1, ..., D]$, representing all nodes of the network, and this pool is used to generate the initial path. The first element of the path is always the source node, and this element is then removed from the pool. Next we randomly remove another node from the pool and add it to the path, repeating this process until the destination node is reached. In the worst case, the destination node will be the last one, and our individual will have D elements, otherwise the length will be less then D. This means that the length of each individual of the population can be different.

To implement the mutation we use a variant of (5), cutting each value to its integer part (as each value corresponds to a node)

$$v_{g+1,i} = \text{int}\left(x_{g,r_1} + F \cdot (x_{g,r_2} - x_{g,r_3})\right), \ i = 1, ..., NP \tag{8}$$

but due to the fact that not all individuals are the same size, we need to make sure that they are, before applying (8). This is done very simply by adding null values to the end of the two shortest length paths (from x_{g,r_2}, x_{g,r_2} and x_{g,r_3}), until all sizes are equal.

However, after the mutation, each element of the path can be out the boundaries $[1, ..., D]$ or can exist duplicated elements. This creates an incorrect path, and we must fix this elements before proceeding. To do this we replace each element that is either $\notin [1, ..., D]$ or duplicated, by a random value selected from a pool of all elements $\in [1, ..., D]$ that are not yet in the path.

It should be said, however, that the mutation is applied only to the inner elements in the path, leaving the initial (source) and final (destination) nodes out, guaranteeing this way that they are always the first and last node in the path.

The crossover is implemented in a very similar way to the generic DE, but remember that in our implementation, each individual can have a different size. So we implement (6), replacing D with $D_{min} = \min\left(\text{length}(v_{g+1,i}), \text{length}(x_{g,i})\right)$.

$$u_{g+1,i,j} = \begin{cases} v_{g+1,i,j} & \text{if } (rand(j) \leq CR) \text{ or } j = rand_i(k) \\ x_{g,i,j} & \text{otherwise} \end{cases}, j = 1, ..., D_{min} \tag{9}$$

But we must make sure that the last element in the trial path is our destination node, whether it comes from the mutant individual or from the current population. If it is not, then we must append the rest of the elements $\in [D_{min}, D_{min+1}, ..., D_{max}]$ from the longest path, to the trial individual. Either way, we have assured that the path still starts in the source and ends in the destination, as is required.

Finally, the selection is the greedy DE selection, as indicated in (7). But to evaluate each individual, we will use the pre-process values calculated previously, in the following manner

$$\mathrm{f(x)} = \sum_{j=1}^{D-1} \begin{cases} w_{\mathrm{x}_j,\ \mathrm{x}_{j+1}} & \text{if } \mathrm{x}_{j+1} \in \mathrm{N}\left(\mathrm{x}_j\right) \\ SP_{\mathrm{x}_j,\ \mathrm{x}_{j+1}} * 10 & \text{otherwise} \end{cases} \tag{10}$$

where $\mathrm{x} = x_{g,i}$, just to simplify the expression, $w_{\mathrm{x}_j,\ \mathrm{x}_{j+1}}$ is the weight of the edge between node x_j and node x_{j+1}, $\mathrm{N}\left(\mathrm{x}_j\right)$ is a function that gives the neighbours of node x_j, and $SP_{\mathrm{x}_j,\ \mathrm{x}_{j+1}}$ is the value of the sum of the edges that are the shortest path between node x_j and x_{j+1}. 10 is an arbitrary factor to penalize the value of the invalid segment of the path.

4 Results

In this section we will present the results of our algorithm in a couple of problems, to see the efficiency and results. We will use a couple of networks available via the OR-Library [1] to test our algorithm, namely *rcsp2* and *rcsp10*, with $D = 100$ nodes and with $D = 200$ nodes, respectively.

DE must be calibrated to each problem, in order to have a suitable performance and accuracy. This is done using its parameters: the scale factor F and the crossover rate CR. After some tests, we came to the conclusion that $F = 0.8$ and $CR = 0.4$ where good values, and those are the ones we will use in all instances of our problems. We also limit the number of generations to 20000 in the first problem (*rcsp2*).

In the EA community, it is usual the evaluate the results of an algorithm as the average of several executions, so we have executed our algorithm 10 times for each problem, and in Figure 1 we can see the evolution of the best solution for problem *rcsp2* in all 10 executions. As we can see, the best solution is reached by almost all executions between generations $5000 - 6000$, but we can have a very approximated value around generation $300 - 400$. This proves that if we want more speed for a less accurate value, we can limit the number of generations, and still get a very acceptable value. This is the reason that in the second problem (*rcsp10*) we lowered the maximum number of generations to 10000. It is still a high number, but this way we assure better results.

The results for both problems for all executions are resumed in Table 1. First of, we can see even for a very lower population (value of NP), we reach good results. Usually in EA, NP is always greater that D, in particular in DE, Storn says that good values for NP are between $5 * D$ and $10 * D$ [10]. As can be seen, we tried as low as $NP = D$, and still get acceptable results. Remember that the lower the population, the quickest the algorithm will execute. The average results from the 10 executions of all problems have also a very decent accuracy, up to 9% in the worst case.

Problem	D	NP	Best	Average	St.Dev.	Optimum	Accuracy
rcsp2	100	100	79	86.0	11.0	79	9%
rcsp2		200	79	82.5	3.2		4%
rcsp10	200	200	175	185.5	17.4	175	6%
rcsp10		400	175	181.3	11.9		4%

Table 1: Results of our algorithm in two different problems, with different values for NP

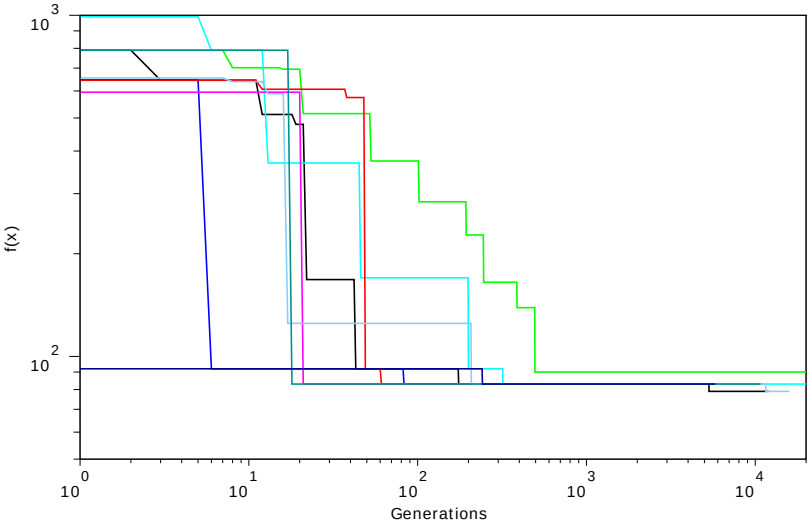

Figure 1: Evolution of the best solution in rcsp2, ($NP = 200$), in 10 executions.

5 Conclusion

Our propose is to confirm that DE is a valid algorithm for the SP, a combinatorial optimization problem. This problem has in interesting issue in that the length of each individual can vary in each generation, which is addressed here for the first time, as far as we know. We have functionally solved this issue, and proved that the DE is a valid algorithm for solving the proposed problem, reaching optimum or near optimum results, albeit in quite more time than Dijkstra, for instance.

For future work, we intent to refine the algorithm, making it more efficient and also to make it multi-objective, for solving Multi-Objective Shortest Path problems (MOSP), that is really the long term objective of this approach.

References

[1] J. E. Beasley. Or-library (http://people.brunel.ac.uk/˜ mastjjb/jeb/info.html), 2010.

[2] Richard Bellman. On a routing problem. *Quart. Appl. Math.*, 16:87–90, 1958.

[3] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. 10.1007/BF01386390.

[4] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100 –107, July 1968.

[5] Jouni A. Lampinen and Ivan Zelinka. Mixed integer-discrete-continuous optimization, by differential evolution, part 1: the optimization method. In Pavel Oaamera, editor, *Proceedings of MENDEL'99, 5th International Mendel Conference on Soft Computing*, pages 71–76, Brno, Czech Republic, June 1999. University of Technology, Faculty of Mechanical Engineering, Institute of Automation and Computer Science.

[6] Zbigniew Michalewicz and David B. Fogel. *How to Solve It: Modern Heuristics*. Springer, enlarged 2nd edition, December 2004.

[7] Godfrey C. Onwubolu and Donald Davendra. *Differential Evolution: A Handbook for Global Permutation-Based Combinatorial Optimization*, volume 175 of *Studies in Computational Intelligence*. CRC Press, Fev 2009.

[8] Kenneth Price, Rainer Storn, and Jouni A. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization*, volume 1 of *Natural Computing Series*. Springer, March 2005.

[9] Rainer Storn and Kenneth Price. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, ICSI, March 1995.

[10] Rainer Storn and Kenneth Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.