Differential evolution

From Wikipedia, the free encyclopedia

In evolutionary computation, **differential evolution** (DE) is a method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. Such methods are commonly known as metaheuristics as they make few or no assumptions about the problem being optimized and can search very large spaces of candidate solutions. However, metaheuristics such as DE do not guarantee an optimal solution is ever found.

DE is used for multidimensional real-valued functions but does not use the gradient of the problem being optimized, which means DE does not require for the optimization problem to be differentiable as is required by classic optimization methods such as gradient descent and quasi-newton methods. DE can therefore also be used on optimization problems that are not even continuous, are noisy, change over time, etc.[1]

DE optimizes a problem by maintaining a population of candidate solutions and creating new candidate solutions by combining existing ones according to its simple formulae, and then keeping whichever candidate solution has the best score or fitness on the optimization problem at hand. In this way the optimization problem is treated as a black box that merely provides a measure of quality given a candidate solution and the gradient is therefore not needed.

DE is originally due to Storn and Price.[2][3] Books have been published on theoretical and practical aspects of using DE in parallel computing, multiobjective optimization, constrained optimization, and the books also contain surveys of application areas.[4][5][6]

Contents [hide]

Algorithm [edit]

A basic variant of the DE algorithm works by having a population of candidate solutions (called agents). These agents are moved around in the search-space by using simple mathematical formulae to combine the positions of existing agents from the population. If the new position of an agent is an improvement it is accepted and forms part of the population, otherwise the new position is simply discarded. The process is repeated and by doing so it is hoped, but not guaranteed, that a satisfactory solution will eventually be discovered.

Formally, let $f : \mathbb{R}^n \to \mathbb{R}$ be the cost function which must be minimized or fitness function which must be maximized. The function takes a candidate solution as argument in the form of a vector of real numbers and produces a real number as output which indicates the fitness of the given candidate solution. The gradient of f is not known. The goal is to find a solution m for which $f(m) \leq f(p)$ for all p in the search-space, which would mean m is the global minimum. Maximization can be performed by considering the function $h := -f$ instead.

Let $\mathbf{x} \in \mathbb{R}^n$ designate a candidate solution (agent) in the population. The basic DE algorithm can then be described as follows:

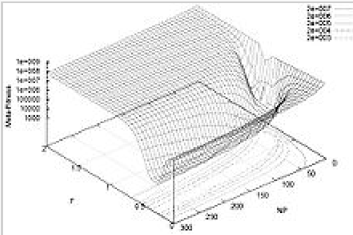
- Initialize all agents $\mathbf{x}$ with random positions in the search-space.
- Until a termination criterion is met (e.g. number of iterations performed, or adequate fitness reached), repeat the following:
  - For each agent $\mathbf{x}$ in the population do:
    - Pick three agents $\mathbf{a}, \mathbf{b}$, and $\mathbf{c}$ from the population at random, they must be distinct from each other as well as from agent $\mathbf{x}$
    - Pick a random index $R \in \{1, \ldots, n\}$ (n being the dimensionality of the problem to be optimized).

- Compute the agent's potentially new position $\mathbf{y} = [y_1, \ldots, y_n]$ as follows:
  - For each i, pick a uniformly distributed number $r_i \equiv U(0,1)$
  - If $r_i < \mathrm{CR}$ or $i = R$ then set $y_i = a_i + F \times (b_i - c_i)$ otherwise set $y_i = x_i$
  - (In essence, the new position is outcome of binary crossover of agent $\mathbf{x}$ with intermediate agent $\mathbf{z} = \mathbf{a} + F \times (\mathbf{b} - \mathbf{c})$.)
- If $f(\mathbf{y}) < f(\mathbf{x})$ then replace the agent in the population with the improved candidate solution, that is, replace $\mathbf{x}$ with $\mathbf{y}$ in the population.

- Pick the agent from the population that has the highest fitness or lowest cost and return it as the best found candidate solution.

Note that $F \in [0,2]$ is called the *differential weight* and $\mathrm{CR} \in [0,1]$ is called the *crossover probability*, both these parameters are selectable by the practitioner along with the population size $\mathrm{NP} \geq 4$ see below.

Parameter selection [edit]



Performance landscape showing how the basic DE performs in aggregate on the Sphere and Rosenbrock benchmark problems when varying the two DE parameters NP and F, and keeping fixed CR =0.9.

The choice of DE parameters F, CR and NP can have a large impact on optimization performance. Selecting the DE parameters that yield good performance has therefore been the subject of much research. Rules of thumb for parameter selection were devised by Storn et al.[3][4] and Liu and Lampinen.[7] Mathematical convergence analysis regarding parameter selection was done by Zaharie.[8] Meta-optimization of the DE parameters was done by Pedersen [9][10] and Zhang et al.[11]

Variants [edit]

Variants of the DE algorithm are continually being developed in an effort to improve optimization performance. Many different schemes for performing crossover and mutation of agents are possible in the basic algorithm given above, see e.g.[3] More advanced DE variants are also being developed with a popular research trend being to perturb or adapt the DE parameters during optimization, see e.g. Price et al.,[4] Liu and Lampinen,[12] Qin and Suganthan,[13] Civicioglu [14] and Brest et al.[15] There are also some work in making a hybrid optimization method using DE combined with other optimizers. [16]

Sample code [edit]

The following is a specific pseudocode implementation of differential evolution, written similar to the Java language. For more generalized pseudocode, please see the listing in the Algorithm section above.

```
//definition of one individual in population
class Individual {
 //normally DifferentialEvolution uses floating point variables
 var float data1, data2
 //but using integers is possible too
 var integer data3
}

class DifferentialEvolution {
 //Variables
 //linked list that has our population inside
 var LinkedList<Individual> population=new LinkedList<Individual>()
 //New instance of Random number generator
 var Random random=new Random()
 var integer PopulationSize=20

 //differential weight [0,2]
 var float F=1
 //crossover probability [0,1]
 var float CR=0.5
 //dimensionality of problem, means how many variables problem has. this case 3
(data1,data2,data3)
 var integer N=3;
```

```
 //This function tells how well given individual performs at given problem.
 function float fitnessFunction(Individual in) {
  ...
  return fitness
 }

 //this is main function of program
 function void Main() {
  //Initialize population whit individuals that have been initialized whit uniform
random noise
  //uniform noise means random value inside your search space
  var i=0
  while(i<populationSize) {
   var Individual individual= new Individual()
   individual.data1=random.UniformNoise()
   individual.data2=random.UniformNoise()
   //integers cant take floating point values and they need to be either rounded
   individual.data3=Math.Floor( random.UniformNoise())
   population.add(individual)
   i++
  }

  i=0
  var j
  //main loop of evolution.
  while (!StoppingCriteria) {
   i++
   j=0
   while (j<populationSize) {
    //calculate new candidate solution

    //pick random point from population
    var integer x=Math.floor(random.UniformNoise()%(population.size()-1))
    var integer a,b,c

    //pick three different random points from population
    do{
     a=Math.floor(random.UniformNoise()%(population.size()-1))
    }while(a==x);
    do{
     b=Math.floor(random.UniformNoise()%(population.size()-1))
    }while(b==x| b==a);
    do{
     c=Math.floor(random.UniformNoise()%(population.size()-1))
    }while(c==x | c==a | c==b);

    // Pick a random index [0-Dimensionality]
    var integer R=rand.nextInt()%N;

    //Compute the agent's new position
    var Individual original=population.get(x)
    var Individual candidate=original.clone()

    var Individual individual1=population.get(a)
    var Individual individual2=population.get(b)
    var Individual individual3=population.get(c)

    //if(i==R | i<CR)
     //candidate=a+f*(b-c)
    //else
     //candidate=x
    if( Math.floor((random.UniformNoise()%N)==R | random.UniformNoise()%1<CR){
     candidate.data1=individual1.data1+F*(individual2.data1-individual3.data1)
    }// else isn't needed because we cloned original to candidate
    if( Math.floor((random.UniformNoise()%N)==R | random.UniformNoise()%1<CR){
     candidate.data2=individual1.data2+F*(individual2.data2-individual3.data2)
    }
    //integer work same as floating points but they need to be rounded
    if( Math.floor((random.UniformNoise()%N)==R | random.UniformNoise()%1<CR){

candidate.data3=Math.floor(individual1.data3+F*(individual2.data3-individual3.data3))
```

```
      }

      //see if is better than original, if so replace
      if(fitnessFunction(original)<fitnessFunction(candidate)){
       population.remove(original)
       population.add(candidate)
      }
      j++
     }
    }

    //find best candidate solution
    i=0
    var Individual bestFitness=new Individual()
    while (i<populationSize) {
     var Individual individual=population.get(i)
     if(fitnessFunction(bestFitness)<fitnessFunction(individual)){
      bestFitness=individual
     }
     i++
    }

    //your solution
    return bestFitness
   }
  }
```

See also [edit]

- CMA-ES
- Artificial bee colony algorithm
- The runner-root algorithm (RRA)
- Evolution strategy
- Genetic algorithm
- Differential search algorithm [14]
- Biogeography-based optimization

References [edit]

1. ^ Rocca, P.; Oliveri, G.; Massa, A. (2011). "Differential Evolution as Applied to Electromagnetics". *IEEE Antennas and Propagation Magazine* **53** (1): 38–49. doi:10.1109/MAP.2011.5773566.
2. ^ Storn, R.; Price, K. (1997). "Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces". *Journal of Global Optimization* **11**: 341–359. doi:10.1023/A:1008202821328.
3. ^ *a b c* Storn, R. (1996). "On the usage of differential evolution for function optimization". *Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS)*. pp. 519–523.
4. ^ *a b c* Price, K.; Storn, R.M.; Lampinen, J.A. (2005). *Differential Evolution: A Practical Approach to Global Optimization*. Springer. ISBN 978-3-540-20950-8.
5. ^ Feoktistov, V. (2006). *Differential Evolution: In Search of Solutions*. Springer. ISBN 978-0-387-36895-5.
6. ^ Chakraborty, U.K., ed. (2008), *Advances in Differential Evolution*, Springer, ISBN 978-3-540-68827-3
7. ^ Liu, J.; Lampinen, J. (2002). "On setting the control parameter of the differential evolution method". *Proceedings of the 8th International Conference on Soft Computing (MENDEL)*. Brno, Czech Republic. pp. 11–18.
8. ^ Zaharie, D. (2002). "Critical values for the control parameters of differential evolution algorithms". *Proceedings of the 8th International Conference on Soft Computing (MENDEL)*. Brno, Czech Republic. pp. 62–67.
9. ^ Pedersen, M.E.H. (2010). *Tuning & Simplifying Heuristical Optimization* (PDF) (PhD thesis). University of Southampton, School of Engineering Sciences, Computational Engineering and Design Group.
10. ^ Pedersen, M.E.H. (2010). "Good parameters for differential evolution" (PDF). *Technical Report HL1002* (Hvass Laboratories).
11. ^ Zhang, X; Jiang, X; Scott, P.J. (2011). "A Minimax Fitting Algorithm for Ultra-Precision Aspheric Surfaces". *The 13th International Conference on Metrology and Properties of Engineering Surfaces*.
12. ^ Liu, J.; Lampinen, J. (2005). "A fuzzy adaptive differential evolution algorithm". *Soft Computing* **9** (6): 448–462. doi:10.1007/s00500-004-0363-x.
13. ^ Qin, A.K.; Suganthan, P.N. (2005). "Self-adaptive differential evolution algorithm for numerical optimization". *Proceedings of the IEEE congress on evolutionary computation (CEC)*. pp. 1785–1791.
14. ^ *a b* Civicioglu, P. (2012). "Transforming geocentric cartesian coordinates to geodetic coordinates by using differential search algorithm". *Computers & Geosciences* **46**: 229–247. doi:10.1016/j.cageo.2011.12.011.
15. ^ Brest, J.; Greiner, S.; Boskovic, B.; Mernik, M.; Zumer, V. (2006). "Self-adapting control parameters in

differential evolution: a comparative study on numerical benchmark functions". *IEEE Transactions on Evolutionary Computation* **10** (6): 646–657. doi:10.1109/tevc.2006.872133.
16. ^ Zhang, Wen-Jun; Xie, Xiao-Feng (2003). DEPSO: hybrid particle swarm with differential evolution operator. *IEEE International Conference on Systems, Man, and Cybernetics* (SMCC), Washington, DC, USA: 3816-3821.

External links [edit]

- Storn's Homepage on DE featuring source-code for several programming languages.
- Fast DE Algorithm A Fast Differential Evolution Algorithm using k-Nearest Neighbour Predictor.
- MODE Application Parameter Estimation of a Pressure Swing Adsorption Model for Air Separation Using Multi-objective Optimisation and Support Vector Regression Model.

v · t · e Major subfields of optimization [show]

Categories: Optimization algorithms and methods | Evolutionary algorithms | Mathematical optimization | Operations research