Evolutionary Algorithms Notes

Ngoc Bui
Hanoi University of Science and Technology
ngocjr7@gmail.com

April 2020

Evolutionary algorithms (EAs) are optimization metaheuristics that work on Darwinian principles of Natural Selection or Survival of the Fittest.

1 Genetic Algorithms (GA)

2 Particle Swarm Optimization (PSO)

2.1 Particle

2.2 Velocity

3 Differential Evoluation (DE)

4 Non-dominated Sorting Genetic Algorithm (NSGA-II)

This section summaries NSGA-II[1].

4.1 Multiobjective Optimization (MOO)

The presence of multiple objectives in a problem, in principle, gives rise to set of optimal solutions (largely known as Pareto-optimal solutions). In the absence of any further information, one of these Pareto-optimal solutions cannot be said to be better than the other. This demands a user to find as many Pareto-optimal solutions as possible.

Classical optimization methods (including the multi-criterion decision-making methods) suggest converting the multiobjective optimization problem to a single-objective optimization problem by emphasizing one particular Pareto-optimal solution at a time. When such a method is to be used for finding multiple solutions, it has to be applied many times, hopefully finding a different solution at each simulation run.

NOTE: The reason for preserving the diversity of optimal-solution is to prevent local optimal and explore the different areas in the search space.

Multiobjective evolutionary algorithms (MOEAs): Old approaches:

- NSGA
- Pareto-archived evolution strategy (PAES)
- Strength-Pareto EA (SPEA)

Contributions (NSGA-II):

- Solving NSGA problem: include computation ($O(M \times N^3) \rightarrow O(M \times N^2)$)
- Ensuring diversity of optimal-solution without sharing parameter. The traditional approach in NSGA used a shared parameter to get a wide variety of equivalent solutions that have relied mostly on the concept of sharing.
- Applying elitism from recent results.
- Proposing Constrained Multiobjective optimization and handling constraint with NSGA-II

4.2 Elitist Multiobjective Optimization Evolutionary Algorithms

Algorithms demonstrated the necessary additional operators for converting a simple EA to an MOEA. Two common features of those algorithms are:

1. assigning fitness to population members based on nondominated sorting
2. preserving diversity among solutions of the same nondominated front

4.2.1 Strength-Pareto EA (SPEA)

- Authors suggested maintaining an external population at every generation storing all nondominated solutions discovered so far beginning from the initial population.
- This external population participates in all genetic operations. At each generation, a combined population with the external and the current population is first constructed
- All nondominated solutions in the combined population are assigned a fitness based on the number of solutions they dominate and dominated solutions are assigned fitness worse than the worst fitness of any nondominated solution
- This assignment of fitness makes sure that the search is directed toward the nondominated solutions. A deterministic clustering technique is used to ensure diversity among nondominated solutions.

4.2.2 Pareto-archived evolution strategy (PAES)

- A simple MOEA using a single-parent single-offspring EA similar to (1+1)-evolution strategy. Instead of using real parameters, binary strings were used and bitwise mutations were employed to create offsprings.
- In their PAES, with one parent and one offspring, the offspring are compared with respect to the parent. If the offspring dominates the parent, the offspring is accepted as the next parent and the iteration continues. On the other hand, if the parent dominates the offspring, the offspring is discarded and a new mutated solution (a new offspring) is found.

4.3 Elitist Nondominated Sorting Genetic Algorithm (NSGA-II)

NSGA-II introduces two main techniques, first is nondominated sorting, where authors propose a fast algorithm $O(M \times N^2)$ which can find all Pareto-front. The second is *crowding distance* which is used to preserve the diversity of optimal-solutions.

4.3.1 Nondominated Sorting

Notation M is the number of objectives and N is the number of solutions in the population.

Goal:
finding all pareto-front of population.

Naive Approach:
1. create a pool that includes all current solution
2. find the first Pareto-front by finding all nondominated solutions $O(M \times N^2)$
3. remove all nondominated solutions from the pool
4. repeat 2 while pool not empty.

```
pool = list of all solution in population
paretos = list of list
while pool not empty:
    current_pareto = list()
    for solution in population:
        d[solution] = number of other_solution that dominate solution (O(MxN))
        if d[solution] == 0:
            current_pareto.append(solution)
            pool.remove(solution)
    paretos.append(current_pareto)
```

Fast algorithm:
1. Compare all pairs of solutions and each solution stores 2 information:
   - n_p the number of solutions that dominate the solution p.
   - S_p set of solution that solution p dominate.
2. Set of solutions having $n_p = 0$ become first pareto-front.
3. Remove solutions in pareto-front from pool.
   - For each solution p in pareto-front, for all solution p' in S_p and reduce the number $n_{p'}$.
4. Repeat 2. for until pool not empty

```
for p1 in solutions
    for p2 in solutions:
        if p1 dominate p2 for all objective:
            n[p2] += 1
            S[p1].append(p2)

pool = solutions
paretos = list()
while not pool.empty():
    cur_paretos = list()
    for p in solutions:
        if n[p] == 0:
            cur_paretos.append(p)
            n[p1] -= 1 for all p1 in S[p]
            pool.remove(p)
    paretos.append(cur_paretos)
```

4.3.2 Crowding distance - Diversity preserving

The old version (NSGA) use a shared parameter to maintain sustainable diversity in a population. This parameter is related to the distance metric chosen to calculate the proximity measure between two population members. The parameter denotes the largest value of that distance metric within which any two solutions share each other's fitness. This approach has several disadvantages:

- Performance depends on the chosen parameter.
- Complexity reaches to $O(N^2)$ since we have to compute all pairs of solutions.

Proposed approach:

The authors proposed a new metric called *crowding-distance* which is an estimate of the perimeter of the cuboid formed by using the nearest neighbors as the vertices. Essentially, crowding-distance measures how close a solution is to other solutions in the same pareto-front. If a solution has high crowding-distance, it means that solution covers the higher area in pareto-front. It is more meaningful when we want to preserve the diversity of the solution set.

$$distance[i] = \begin{cases} \infty & \text{if } f_m(i) = f_m^{max} | f_m^{min} \\ \sum_{m \in M} \frac{f_m(i+1) - f_m(i-1)}{f_m^{max} - f_m^{min}} & \text{if other cases} \end{cases}$$

Where f_m is objective function value of m^{th} objective. $f_m(i+1)$ is nearest solution that have objective bigger than $f_m(i)$, similar to $f_m(i-1)$.

1. The crowding-distance computation requires sorting the population according to each objective function value in ascending order of magnitude.
2. Thereafter, for each objective function, the boundary solutions (solutions with the smallest and largest function values) are assigned an infinite distance value.
3. All other intermediate solutions are assigned a distance value equal to the absolute normalized difference in the function values of two adjacent solutions. This calculation is continued with other objective functions.
4. The overall crowding-distance value is calculated as the sum of individual distance values corresponding to each objective. Each objective function is normalized before calculating the crowding distance

`pseudocode`

Crowded-Comparison Operator

Assume that every individual i in the population has two attributes:

- nondomination rank (i_r)
- crowding distance (i_d)

Authors define a partial order as:

$$i < j \text{ if } (i_r < j_r) \text{ or } ((i_r == j_r) \text{ and } (i_d > j_d))$$

4.3.3 Evoluationary loop

Figure 1 shows the general structure of nsga-ii.

$R_t = P_t \cup Q_t$	combine parent and offspring population
$\mathcal{F} =$ `fast-non-dominated-sort`(R_t)	$\mathcal{F} = (\mathcal{F}_1, \mathcal{F}_2, \ldots)$, all nondominated fronts of R_t
$P_{t+1} = \emptyset$ and $i = 1$	
until $\lvert P_{t+1}\rvert + \lvert\mathcal{F}_i\rvert \leq N$	until the parent population is filled
`crowding-distance-assignment`$(\mathcal{F}_i)$	calculate crowding-distance in $\mathcal{F}_i$
$P_{t+1} = P_{t+1} \cup \mathcal{F}_i$	include ith nondominated front in the parent pop
$i = i + 1$	check the next front for inclusion
Sort$(\mathcal{F}_i, \prec_n)$	sort in descending order using $\prec_n$
$P_{t+1} = P_{t+1} \cup \mathcal{F}_i[1 : (N - \lvert P_{t+1}\rvert)]$	choose the first $(N - \lvert P_{t+1}\rvert)$ elements of $\mathcal{F}_i$
$Q_{t+1} =$ `make-new-pop`(P_{t+1})	use selection, crossover and mutation to create a new population Q_{t+1}
$t = t + 1$	increment the generation counter

Figure 1: a general structure

5 Multiobjective Evolutionary Algorithm Based on Decomposition (MOEA/D)

6 Multifactorial Evolutionary Algorithm (MFEA)

This section summaries this paper [2]

6.1 Multifactorial Optimization (MFO)

The purpose of MFO is not to find optimum trade-offs among the constitutive[1] objective functions. Rather, the goal is to fully and concurrently optimize each task, aided by the implicit parallelism of population-based search.

Within the aforementioned scope, consider a situation wherein K optimization tasks are to be performed simultaneously. Without loss of generality, all tasks are assumed to be minimization problems. The j^{th} task, denoted T_j, is considered to have a search space X_j on which the objective function is defined as $f_j : X_j \rightarrow R$

MFO is the problem that aim of finding $\{x_1, x_2, \ldots, x_{K-1}, x_K\} = argmin f_1(x), f_2(x), \ldots, f_{K-1}(x), f_K(x)$. Herein, each f_j is treated as an additional factor influencing the evolution of a single population of individuals. For this reason, the composite problem is referred to hereafter as a K-factorial problem.

Notations:

- P: population, a set of properties for every individual p_i, where $i \in 1, 2, \ldots, |P|$.
- p_i: an individual, encoded in a unified search space encompassing $X_1, \ldots, X_K$ and can be decoded into a task-specific solution representation with respect to each of the K optimization tasks.
- Ψ_j^i: *Factorial cost* (similar to fitness) of individual p_i in task T_j. $\Psi_j^i = \lambda * \delta_j^i + f_j^i$ where λ is a large penalizing multiplier, f_j^i and δ_j^i are the objective value and the total constraint violation, respectively, of pi with respect to T.

[1] (adj) forming a part of something; components

- r_j^i: *Factorial rank* of p_i on task T_j the index of p_i in the list of population members sorted in ascending order with respect to Ψ_j^i.
- $\varphi_i = 1/min_{j \in 1..K}\{r_j^i\}$: *Scalar fitness*, the best rank over all task of individual p_i.
- $\tau_i = argmin_j\{r_j^i\}$: *Skill factor* of individual p_i is the one task, amongst all other tasks in MFO, on which the individual is most effective.

p_a is considered to dominate p_b ($p_a >> p_b$) in a multifactorial sense simply if $\varphi_a > \varphi_b$. If an individual p_* maps to the global optimum of any task, then, $\varphi^* \geq \varphi_i$ for all $i \in \{1, 2, \ldots, |P|\}$. Therefore, it can be said that the proposed technique is indeed consistent with the ensuing definition of multifactorial optimality. An individual p_*, with a list of objective values $f_1, f_2, \ldots, f_K$, is considered optimum in multifactorial sense if $\exists j \in \{1, 2, \ldots, K\}$ such that $f_j^* \leq f_j(x_j)$, for all feasible $x_j \in X_j$.

6.2 Multifactorial Evolutionary Algorithm (MFEA)

The basic structure of the MFEA (most similar to GA's structure, the differents are on algorithm 2 and 3)

```
#. Generate an initial population of individuals and store it in current-pop (P).
#. Evaluate every individual with respect to every optimization task in the multitasking environme
#. Computetheskillfactor($\tau$)of each individual.
#. while (stopping conditions are not satisfied) do
- Apply genetic operators on current-pop to generate an offspring-pop (C). Refer to Algorithm 2.
- Evaluate the individuals in offspring-pop for selected optimization tasks only (see Algorithm 3)
- Concatenate offspring-pop and current-pop to form an intermediate-pop $(P C)$.
- Update the scalar fitness ($\varphi$) and skill factor ($\tau$) of every individual in intermedi
- Select the fittest individuals from intermediate-pop to form the next current-pop (P).
#. end while
```

6.2.1 Individual Encoding

- Assume that that in K optimization tasks to be performed simultaneously, and we encode the individuals by the dimensionality of the j^{th} task is given by D_j.
- Dimensionality of unified search space $D_{multitask} = max_{j=1...k}\{D_j\}$
- The i_{th} dimension of the unified search space is represented by a random-key y_i, and the fixed range represents the box-constraint of the unified space. While addressing task T_j, we simply refer to the first D_j random-keys of the chromosome.

Summary:
- assume that we encode the individuals by the dimensionality of the j^{th} task is given by D_j.
- the dimensionality of unified search space

$$D_m = max_{j=1...k} D_j$$

- Unified space represented by $Y = y; |; |y| = D_m$.
  - $y = \{y_1, \ldots y_{D_m}\}$ can understand as encoded chromosome(individual) in Y
  - $y_i \in [0, 1]$ is called random-key (value).
- the first D_j random-keys $\{y_1, \ldots y_{D_j}\}$ is representation of individual y in task j^{th}.

6.2.2 Individual Decoding

Consider the ith variable (x_i) of the actual problem to be bounded within the range $[L_i, U_i]$. If the corresponding random-key of an individual takes a value y_i, then it's mapping into the search space of the actual problem is given by $x_i = L_i + (U_i - L_i) \times y_i$. In contrast, for the case of discrete optimization, the chromosome decoding scheme is usually problem dependent

Summary

Decoding $\{y_1, \dots y_{D_j}\}$ for task T_j: * $x^{(j)} = x_1^{(j)}, \dots x_{D_j}^{(j)} \in X_j$: is the representation of individual in actual space X_j of the problem. * $x_i^{(j)} \in [L_i^{(j)}, U_i^{(j)}]$ is domain of $x_i^{(j)}$. * For continuous problem:

$$\rightarrow x_i = L_i + (U_i - L_i) \times y_i$$

* For discrete problem: depend on each problem.

6.2.3 Population initialization

- Dimensionality of unified search space $D_{multitask} = max_{j=1\dots k}\{D_j\}$. Reasons:
  - circumvent the curse of dimensionality.
  - discovery and implicit transfer of useful genetic material from one task to another in an efficient manner. A single individual in the population may inherit genetic building blocks corresponding to multiple optimization tasks.

6.2.4 Genetic mechanisms (Assortative mating)

A key feature of the MFEA is that certain conditions must be satisfied for two randomly selected parent candidates to undergo crossover. The principle followed is that of nonrandom or assortative[2] mating which states that individuals prefer to mate with those belonging to the same cultural background. In the MFEA, the skill factor (τ) is viewed as a computational representation of an individual's cultural bias.

Algorithm 2: Assortative mating

```
Consider two parent candidates pa and pb randomly selected from current-pop.
1. Generate a random number rand between 0 and 1.
2. if ($\Tau_b$ == $\Tau_b$) or (rand < rmp) then
i. Parents pa and pb crossover to give two offspring individuals ca and cb.
3. else
i. Parent p_a is mutated slightly to give an offspring ca.
ii. Parent p_b is mutated slightly to give an offspring cb. 4. end if
```

The term `($\Tau_b$ == $\Tau_b$) or (rand < rmp)` conversely means if their skill factors differ, crossover only occurs as per a prescribed random mating probability (`rmp`), or else mutation kicks in. the parameter `rmp` is used to balance exploitation and exploration of the search space.

Finally, while choosing crossover and mutation operators, it must be kept in mind that the random-keys presented earlier are always interpreted as continuous variables, even if the underlying optimization problem is discrete. This encourages the use of existing real-coded genetic operators, or the design of new operators for improved navigation of the composite landscapes associated with MFO.

[2] denoting or involving the preferential mating of animals or marring of people with similar

6.2.5 Selective evaluation

A question is posed when reviewing the general structure of MFEA.
Question: Why offsprings are evaluated for only one task?
Evaluating every individual for every problem being solved will often be computationally too expensive, and is therefore undesirable. An important observation we have made is that an individual generated in the MFEA is unlikely to be high performing across all tasks.

The algorithm 3, which is named as *selective imitating*, said that the offspring is evaluated only for one task. If the offspring is inherited from two parents then randomly choosing the parent which the offspring imitate to.

Another question: Why do not evaluate offspring at both parents' tasks?

Algorithm 3: Selective imitating

```
An offspring ‘c’ will either have two parents (pa and pb) or a single parent (pa or pb) - see Algo
1.
if (‘c’ has 2 parents) then
i. Generate a random number rand between 0 and 1.
ii. if (rand < 0.5) then
‘c’ imitates pa $\rightarrow$ The offspring is evaluated only for task $\tau_a$ (the skill factor
iii. else
‘c’ imitates pb $\rightarrow$ The offspring is evaluated only for task $\tau_b$ (the skill factor
iv. end if 2. else
i. ‘c’ imitates its single parent $\rightarrow$ The offspring is evaluated only for that task whic
3. end if
4. Factorial costs of ‘c’ with respect to all unevaluated tasks are
artificially set to infinity (a very large number).
```

6.2.6 Selection Operation

The MFEA follows an elitist strategy which ensures that the best individuals survive through the generations.

6.2.7 The salient features of MFEA

An important question in the whole paper is "Why multifactorial optimization is effective? and Why should we use MFEA?". The author does not specify the theoretical evidence that shows the efficiency of MFEA. The author repeatedly uses "implicit" and "salient" to describe the strongness of MFEA. The author demonstrates by numerical experiments instead.

"Standard EAs typically generate a large population of candidate solutions, all of which are unlikely to be competent for the task at hand. In contrast, in a multitasking environment, it is intuitively more probable that a randomly generated or genetically modified individual is competent for at least one task. The mechanism of the MFEA builds on this observation by effectively splitting the population into different skill groups, each excelling at a different task. More interestingly and important, the genetic material created within a particular group may turn out to be useful for another task as well. Thus, in such situations, the scope for genetic transfer across tasks can potentially lead to the discovery of otherwise hard to find global optima"

7 Multi-Objective Multifactorial Optimization (MO-MFEA)

References

[1] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[2] Abhishek Gupta, Yew-Soon Ong, and Liang Feng. Multifactorial evolution: toward evolutionary multitasking. *IEEE Transactions on Evolutionary Computation*, 20(3):343–357, 2015.