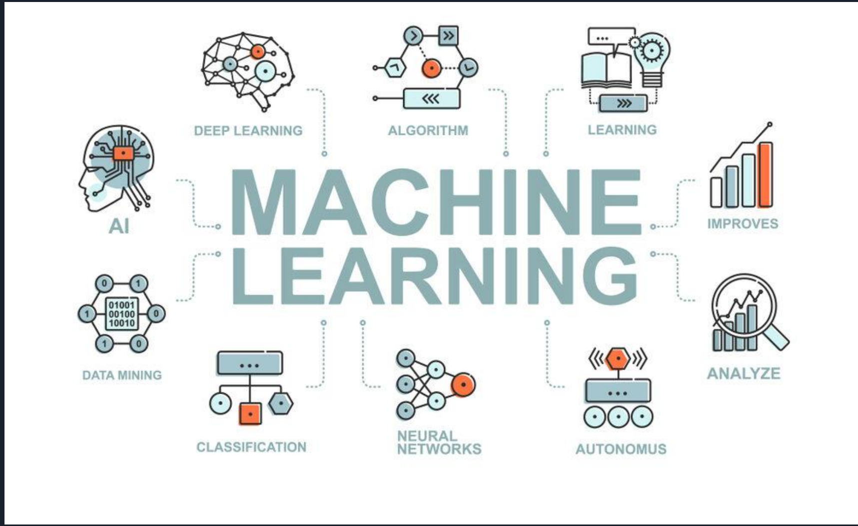
Investigating machine learning algorithms for adapting meta-heuristics parameters

Mokhtar Essaid
Samer El Zant

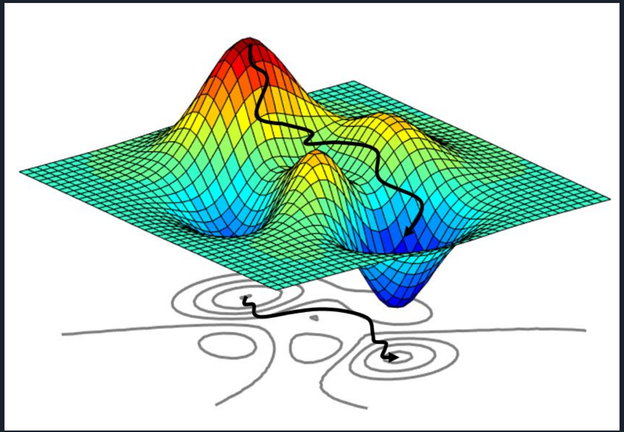
Kanan Mikayilov
Rufat Huseynov

Objective

Exploitation of machine learning capabilities for generating an appropriate set of parameters of the optimization process.



Optimization

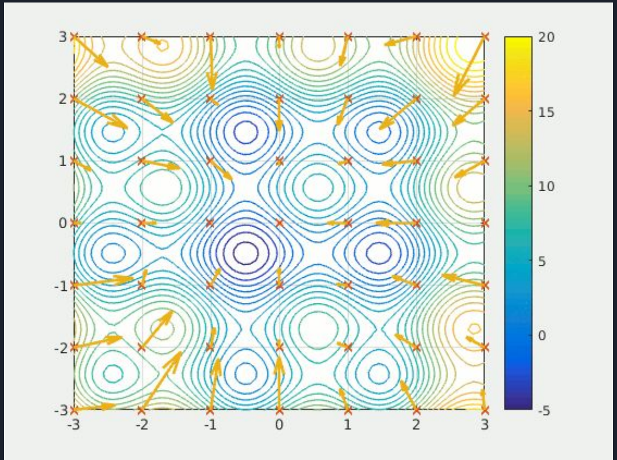


The process of finding the most optimal value of a real mathematical function.

Algorithm categories:

- Direct algorithms
- Stochastic algorithms
- Population algorithms

Population Algorithms

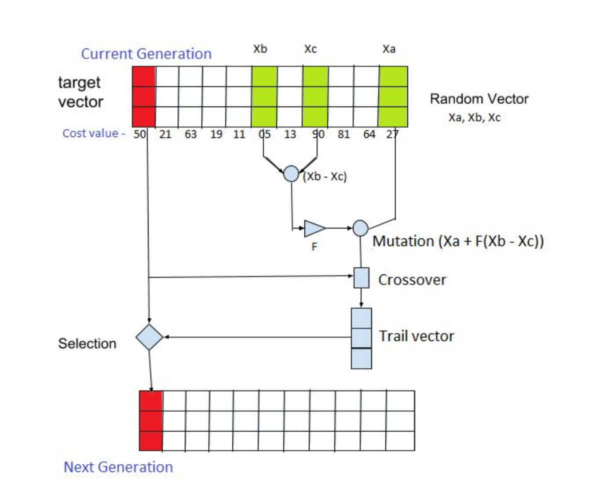


Possible solutions, together form a population.

Algorithms:

- Genetic algorithm (GA)
- Differential Evolution (DE)
- Particle Swarm Optimization (PSO)

Differential Evolution Algorithm



An evolutionary algorithm (EA), based on natural selection

Stages:

- Initialization
- Selection 1 (Optional)
- Mutation
- Crossover
- Selection 2 (Optional)
- Evaluation

Variants:

- Standard DE (SDE)
- Composite DE (CODE)
- Self-adaptive DE (JDE)
- JADE

Generation scheme

Mutation and crossover are the main stages of DE algorithms

Schemes:

- DE/rand/1
- DE/best/1
- DE/current-to-best/1
- rand/1/bin

$$V_i = x_{R1} + F * (x_{R2} - x_{R3})$$

$$V_i = x_i + F * (x_{pbest} - x_i) + F * (x_{R1} - x_{R2})$$

Standard Differential Evolution (SDE)

Data:
NP: population size, *F*: mutation factor, *CR*: crossover probability, *MAXFES*: maximum number of functions evaluations

INITIALIZATION $G = 0$; Initialize all *NP* individuals with random positions in the search space;

while $FES < MAXFES$ **do**

  for $i \leftarrow 1$ **to** *NP* **do**

    GENERATE three individuals x_{r1}, x_{r1}, x_{r1} from the current population randomly. These must be distinct from each other and also from individual x_i, i.e. $r_1 \neq r_2 \neq r_3 \neq i$

    MUTATION Form the donor vector using the formula: $\mathbf{v}_i = \mathbf{x}_{r_1} + F(\mathbf{x}_{r_2} - \mathbf{x}_{r_3})$

    CROSSOVER The trial vector $\mathbf{u}_i$ is developed either from the elements of the target vector $\mathbf{x}_i$ or the elements of the donor vector $\mathbf{v}_i$ as follows:

$$u_{i,j} = \begin{cases} \mathrm{v}_{i,j}, & \text{if } r_{i,j} \leq CR \text{ or } j = j_{rand} \\ x_{i,j}, & \text{otherwise} \end{cases}$$

    where $i = \{1, \ldots, NP\}$, $j = \{1, \ldots, D\}$, $r_{i,j} \sim \cup(0, 1)$ is a uniformly distributed random number which is generated for each j and $j_{rand} \in \{1, \ldots, D\}$ is a random integer used to ensure that $\mathbf{u}_i \neq \mathbf{x}_i$ in all cases

    EVALUATE If $f(\mathbf{u}_i) \leq f(\mathbf{x}_i)$ then replace the individual $\mathbf{x}_i$ in the population with the trial vector $\mathbf{u}_i$

    $FES = FES + NP$

  end

  $G = G + 1$;

end

Standard Differential Evolution (SDE)

Initialization

```python
""" initialization method, creates the population with randomly generated individuals, finds the best individual with its fitness value"""
def initialization(self):

  # create the population
  self.p = []

  for i in range(Config.get_population_size()):
    self.p.append(Vector())

  # sort the population by fitness value asc
  self.p = sorted(self.p, key=lambda individual: individual.get_fitness())

  # get the best individual
  self.BestPosition = self.p[0].get_position()

  # get the best fitness value
  self.BestFitness = self.p[0].get_fitness()
  self.PrevBestFitness = self.BestFitness

  # an arrays to store best values of each generation, and the average values of each generation, if best vector is updated
  self.BestResults = []
  self.AverageResults = []
```

Standard Differential Evolution (SDE)

Generation + Mutation

```
""" generation method serves to find random individuals from population except the current individual"""
def generation(self):
  self.x_i = self.p[self.cur_index]
  self.random_vectors = super().get_random_individuals(Config.get_population_size(), self.cur_index, 3)

"""mutation method serves to generate the mutant vector by using the generation scheme"""
def mutation(self):
  [a, b, c] = self.random_vectors

  self.v_i = Vector()
  self.v_i.set_position( a.get_position() + self.f * ( b.get_position() - c.get_position() ) )
```

Standard Differential Evolution (SDE)

Crossover

```
"""crossover method serves to generate trial vector checking the condition to set either mutant vector cell or current individual cell to certain cell"""
def crossover(self):
  u_i = Vector()

  dimension = Config.get_dimension()

  # generates random value from 0 to dimension size value
  j_rand = rn.randrange(0, dimension)

  for j in range(dimension):
    if self.x_i.position[j] < self.cr or j == j_rand:
      u_i.position[j] = self.v_i.position[j]
    else:
      u_i.position[j] = self.x_i.position[j]

  self.u_i = u_i
```

Standard Differential Evolution (SDE)

Evaluation

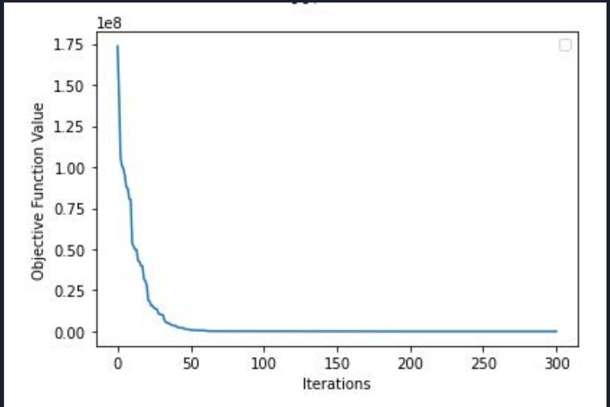
```python
"""evaluation method serves to update the population in case the trial vector has a better (less) fitness value than current individual"""
def evaluation(self):
    function = Config.get_function()

    TargetFitness = function.compute(self.x_i.get_position())
    TrialFitness = function.compute(self.u_i.get_position())

    if TrialFitness < TargetFitness:
        self.p[self.cur_index].set_position(self.u_i.get_position())
        self.p[self.cur_index].set_fitness(TrialFitness)
```

Standard Differential Evolution (SDE)

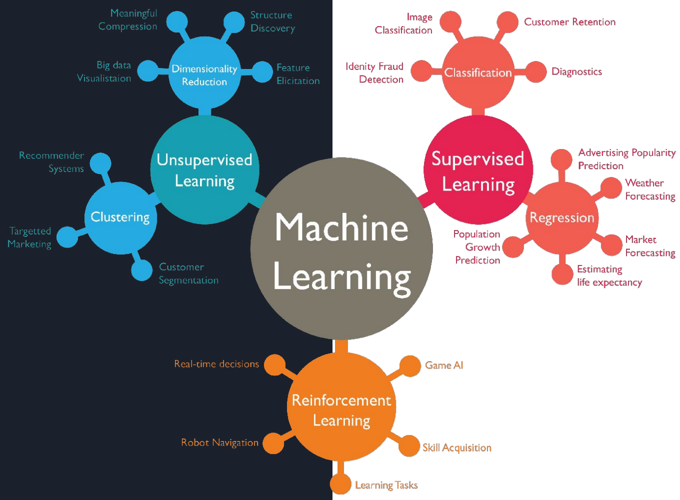
Execution




```
Iteration: 992 --> f([[6.45061 1.02205 1.03085 1.03093 1.05162 1.03862 1.02678 1.0419  1.03081
 1.02532 1.03026 1.04358 1.03018 1.02084 1.02553 1.02995 1.02299 1.03883
 1.03296 1.03736 1.03744 1.03054 1.04633 1.06649 1.10197 1.14887 1.26758
 1.53409 2.29216 5.28683]]) = 6.81190
Iteration: 993 --> f([[6.45061 1.01465 1.03085 1.03093 1.05162 1.03862 1.02678 1.0419  1.03081
 1.02532 1.03026 1.04358 1.03018 1.02084 1.02553 1.02995 1.02299 1.03883
 1.03296 1.03736 1.03744 1.03054 1.04633 1.06649 1.10197 1.14887 1.26758
 1.53409 2.29216 5.28683]]) = 6.79293
```

Parameter tuning

Machine learning is a method of data analysis that automates analytical model building. It is a branch of artificial intelligence (AI) based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention.



Logistic regression is estimating the parameters of a logistic model: a statistical model that models the probability of one event (out of two alternatives) taking place.

ML implementation

Train data

```
"""method to train datasets on logistic regression for F and CR parameters"""
def train_dataset(self):
  # mutation factor
  df =  pd.DataFrame(self.train_mutation_factor)

  x = df.iloc[:, : -1]
  y = df.iloc[:, -1]

  self.f_regression = LogisticRegression(solver='liblinear', random_state=0)
  self.f_regression.fit(x, y)

  # crossover probability
  df =  pd.DataFrame(self.train_crossover_probability)

  x = df.iloc[:, : -1]
  y = df.iloc[:, -1]

  self.cr_regression = LogisticRegression(solver='liblinear', random_state=0)
  self.cr_regression.fit(x, y)
```

Predict data

```
"""method that by using logistic regression, predict the value randomly generated,
def predict(self, test_data, regression):
  test_df = pd.DataFrame(test_data)
  x_test = test_df.iloc[:, :]
  prediction = regression.predict(x_test)

  return prediction
```

Population Reduction

Improve the execution process

$$N_{G+1} = \text{round}\left[\left(\frac{N^{min} - N^{init}}{MAX\_NFE}\right) \cdot NFE + N^{init}\right]$$

Get rid of worst solutions

Benchmark Functions

Ackley Function

$$f(\mathbf{x}) = -a \exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)\right) + a + \exp(1)$$

Rastrigin Function

$$f(\mathbf{x}) = 10d + \sum_{i=1}^{d}\left[x_i^2 - 10\cos(2\pi x_i)\right]$$

Rosenbrock Function

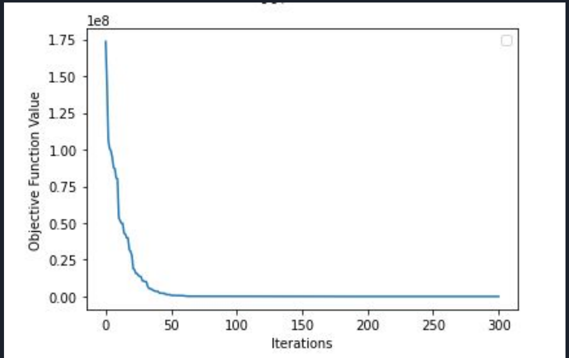
$$f(\mathbf{x}) = \sum_{i=1}^{d-1}\left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$$

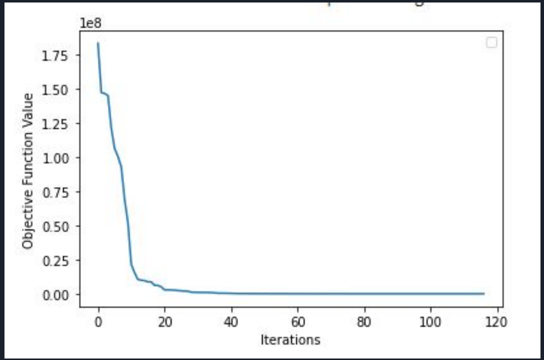
Schwefel Function

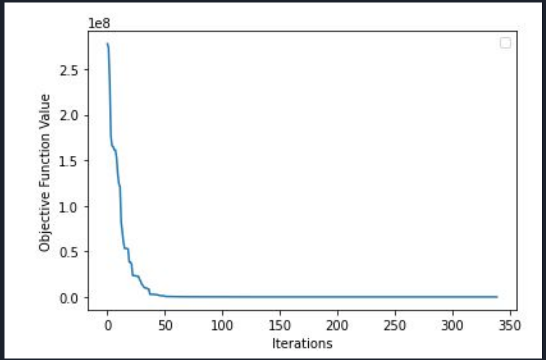
$$f(\mathbf{x}) = 418.9829d - \sum_{i=1}^{d} x_i \sin(\sqrt{|x_i|})$$

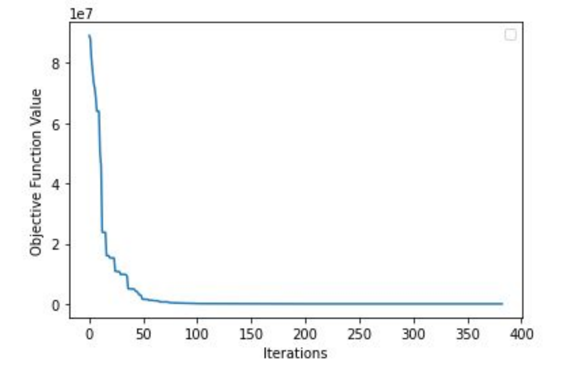
Results

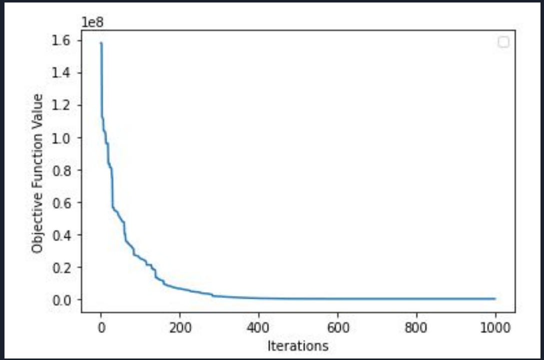
SDE - CODE - JDE - JADE - Cuckoo Search - SDE with ML and population reduction

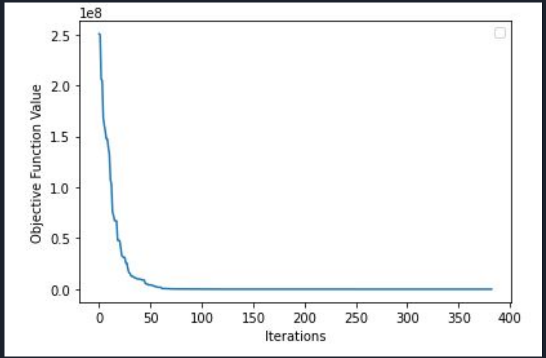












Results

Comparison??

Convergence rate (Best to Worst)

CODE - SDE with ML and population reduction - JDE - JADE - SDE - Cuckoo Search

Execution time (Best to Worst)

SDE- JDE - JADE - Cuckoo Search - CODE - SDE with ML and population reduction

Conclusion

To conclude, we implemented a few meta-heuristic algorithms (CODE. JDE, JADE, SADE) and also SDE including Machine Learning techniques to get more optimal results compared to the previous variants.

To compare the algorithms, and conclude which algorithm was better, we displayed the improvement of optimal values by showing the values in graph.

Thank you for your attention!

Kanan Mikayilov

Rufat Huseynov

12.06.2022