Teaching-Learning Based Optimization Implementation

Complications when optimizing large scale problems, such as multimodality, dimensionality, and differentiability, tend to cause traditional methods of optimization to fail [1]. These methods, such as steepest decent, linear programming, and dynamic programming, will generally not work when handling large problems, particularly if they contain non-linear objective functions [1]. Generally speaking, algorithms utilizing these techniques require precise tuning of parameters specific to each of the algorithms. Selecting the correct parameters is essential in making these algorithms, such as Genetic Algorithm, Particle Swarm Algorithm, and other nature-inspired algorithms, function properly [1]. Teaching-Learning Based Optimization Algorithm (TLBO) is an algorithm created out of the necessity of needing an algorithm that lacks the usual algorithm-specific tuning steps most other optimization algorithms possess [2]. Its inspiration comes from the effects a teacher has on the students (learners) of a classroom setting. The way TLBO works is through two different phases, the teacher phase and the learner phase. The implantation of TLBO below is done in Python3, and as such will be broken up into its constituent phases when discussing the algorithm.

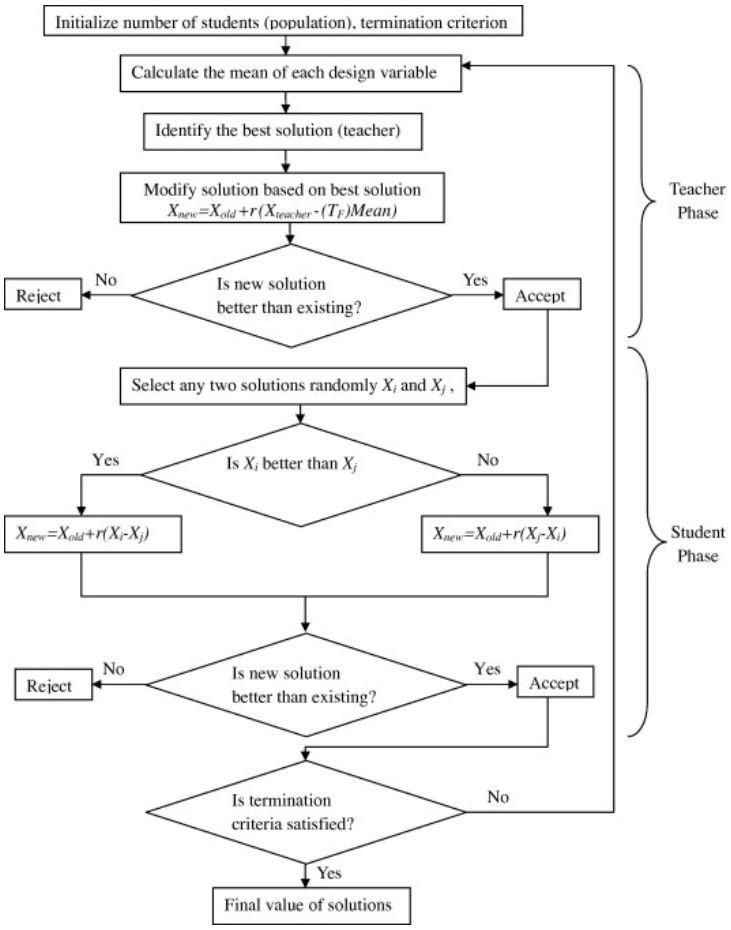

Figure 1: TLBO Implementation

Initialization of TLBO requires two major elements, the termination criterion and the population. The termination criteria will be problem

specific where the most common condition for termination will be tied to the number of iterations of TLBO to be run when finding the global optima defined by the problem given to the algorithm. Population refers to the number of "students" inside of the learning phase of the algorithm. When a population value is given, this directly affects the matrix produced, where each member inside of the total population is given their own unique row inside the matrix, containing values related to the variables given in the columns for use inside of the learning phase. As such, the learning matrix's size is determined by the size of the population multiplied by the number of variables present inside of any given problem set. For our implementation of TLBO, the benchmark function (fitness function) as well as the boundary conditions for the accepted range of values were also instantiated at this time. Should a constraint be violated, a penalty will be issued to the student via the violating constraint being multiplied by a value to cause its respective fitness score to be much worse in respect to the other scores present. Finally, a random state is initiated. The random values used inside of random state is pseudo-random in nature, as a specific randomized state can be seeded by setting the random state equal to an integer value. Otherwise, setting the random state to a value of 'None' will cause the seeded random state to be selected at random.

The first stage of TLBO is the teaching phase, which is to represent the influence a teacher has on a classroom of students. The primary goal of this stage is for the teacher to "teach" the class to the best of their capabilities on the subject manner. For this, the teacher attempts to cause the mean result of the classroom (the students) to increase in value by teaching the students of the classroom to the best of their abilities. As such, the number of subjects a teacher must teach is related to the number of variables present in the problem to be optimized and the number of students needing to be taught is represented by the population size given during the initialization stage of the algorithm. From here, the best student is identified, having the best mean results (fitness) across all the subjects (variables) taken together relative to the rest of the classroom, which will be denoted in a column containing all of the fitness scores. The teacher will then act upon the class, attempting to move the mean closer to its own performance. This is calculated by the function [2]:

$$fitness_{diff} = r_i\ (fitness_{Teacher} - T_F\ fitness_{old})\ ;\ r_i = (0,1)\ ;\ T_F = \text{round}(1+\text{rand}[0,1]\{2-1\})$$

T_F is the teaching factor and can be a value of either 1 or 2 with an equal probability and is meant to determine the value for the mean to be changed while r_i is simply a random value inside of the

range of [0,1]. The solution generated by this equation is used to modify the old solution generated for the mean: $fitness_{new} = fitness_{old} + fitness_{diff}$

```
# teaching phase
for i in self.index:
    T_F = 1+rs.randint(0,2)  # teaching factor is randomly 1 or 2.
    r_i = rs.rand(self.dimension)  # multiplier also random. 1 row * 13 columns

    new_solution = self.population[i] + (r_i * (teacher - (T_F * mean))) # new solution 1 row * 13 columns
    new_solution = np.minimum(np.maximum(new_solution, self.lower_bound), self.upper_bound)

    new_fitness = self.function(new_solution)

    if new_fitness < self.fitness[i]:
        self.population[i] = new_solution
        self.fitness[i] = new_fitness
```

Figure 2: TLBO teaching phase in Python3

From here, the solution of the new mean versus the old mean are compared, and the better one for the problem at hand is kept. This concludes the teaching phase of the algorithm.

The learning phase of the algorithm considers the other way learners in the classroom, that being by learning from each other among the population of learners. After the teacher phase has modified the learners values, two of the learners are selected randomly such that their values are not equal to each other. The way the modification to their fitness values is calculated is based on whether the problem being solved is a maximization problem or a minimization problem. The benchmark functions implemented by our project are all minimization type problems so as such those formulas will be included. The way a minimization problem is solved inside of the learning phase is through the following equations [2]:

```
# learning phase
for i in self.index:

    j = rs.choice(self.index[:i] + self.index[(i + 1):], 1)  # pick another random i!=j
    r_i = rs.rand(self.dimension)

    if self.fitness[i] < self.fitness[j]:
        new_solution = self.population[i] + r_i * (self.population[i] - self.population[j]).flatten()
    else:
        new_solution = self.population[i] + r_i * (self.population[j] - self.population[i]).flatten()

    new_solution = np.minimum(np.maximum(new_solution, self.lower_bound), self.upper_bound)
    new_fitness = self.function(new_solution)

    if new_fitness < self.fitness[i]:
        self.population[i] = new_solution
        self.fitness[i] = new_fitness
```

Figure 3: TLBO learning phase in Python

If ($Student_{p,old} < Student_{Q,old}$)

$$Student_{P,new} = Student_{P,old} + r_i\,(Student_{P,old} - Student_{Q,old});\ r_i = (0,1)$$

If ($Student_{p,old} < Student_{Q,old}$)

$$Student_{P,new} = Student_{P,old} + r_i\,(Student_{Q,old} - Student_{P,old});\ r_i = (0,1)$$

Just like the end of the teaching phase, these new values are compared against the original values, and whichever has the better fitness score is kept. If the termination conditions are not met at this point in time, the algorithm returns to the teaching phase to restart this process as a new iteration. If the termination condition is met, the best value is reported as the final solution the algorithm found through its calculations.

The benchmark functions implemented are all minimization type problems, each with their own set of design variables and constraints. The first of these benchmark functions is a quadratic minimization problem, containing 13 variables and 9 inequality constraints [2]. The equation to be minimized is:

$$f(x) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i$$

The optimum solution for this equation is located at the vector (1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1) with a fitness value equal to -15. The constraints on this function are as follows:

$$g_1(x) = 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \le 0$$
$$g_2(x) = 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \le 0$$
$$g_3(x) = 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \le 0$$
$$g_4(x) = -8x_1 + x_{10} \le 0$$
$$g_5(x) = -8x_2 + x_{11} \le 0$$
$$g_6(x) = -8x_3 + x_{12} \le 0$$
$$g_7(x) = -2x_4 - x_5 + x_{10} \le 0$$
$$g_8(x) = -2x_6 - x_7 + x_{11} \le 0$$
$$g_9(x) = -2x_8 - x_9 + x_{12} \le 0$$

where x_i is bound in the following way:

$$0 \leq x_i \leq 1, i = 1, 2, 3 \ldots 9$$
$$0 \leq x_i \leq 100, i = 10, 11, 12$$
$$0 \leq x_i \leq 1, i = 13$$

The third benchmark function would be the second function implemented in the code. This function is a nonlinear minimization problem containing 7 variables and 4 nonlinear inequality constraints [2]. The function to be minimized is as follows:

$$f(x) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7$$

The optimum solution for this problem is located at the vector (2.330499, 1.951372, −0.4775414, 4.365726, −0.6244870, 1.1038131, 1.594227) and its respective fitness value will be equal to 680.6300573. The inequality constraints on the problem are as follows:

$$g_1(x) = -127 + 2x_1^2 + 3x_2^4 + x_3 + 4x_4^2 + 5x_5 \leq 0$$
$$g_2(x) = -282 + 7x_1 + 3x_2 + 10x_3^2 + x_4 + x_5 \leq 0$$
$$g_3(x) = -196 + 23x_1 + x_2^2 + 6x_6^2 - 8x_7 \leq 0$$
$$g_4(x) = 4x_1^2 + x_2^2 - 3x_1x_2 + 2x_3^2 + 5x_6 - 11x_7 \leq 0$$

The variables for this function are all bound in this way:

$$-10 \leq x_i \leq 10;\ i = 1,2,\ldots,7$$

The final minimization problem implemented would be the design of a pressure vessel. This is a constrained mechanical design benchmark problem. The primary goal of this minimization problem is to minimize the total cost of a pressure vessel in terms of the cost of the materials, formation of the vessel, and the welds used in the vessel. The function for this equation is nonlinear with 3 linear inequality constraints, 1 nonlinear inequality constraint, 2 discrete variables, and two continuous variables [2]. The equation for this function is as follows:

$$f(x) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3$$

Where the constraints on the function are:

$$g_1(x) = -x_1 + 0.0193x_3 \leq 0$$
$$g_2(x) = -x_2 + 0.00954x_3 \leq 0$$
$$g_3(x) = -\pi x_3^2 x_4 - \frac{4}{3}\pi x_3^3 + 1296000 \leq 0$$
$$g_4(x) = x_4 - 240 \leq 0$$

Where the bounds are:

$$0 \leq x_1 \leq 99$$
$$0 \leq x_2 \leq 99$$
$$10 \leq x_3 \leq 200$$
$$10 \leq x_4 \leq 200$$

The implementation of TLBO the authors was done using MATLAB. Our implementation was done using Python, so some variance in results may be due to the differences between how TLBO is implemented in the two languages. The results we achieved with the functions we implemented resulted in being able to replicate the results of the paper's implementation of TLBO using MATLAB with our implementation of TLBO using Python for the first benchmark function.

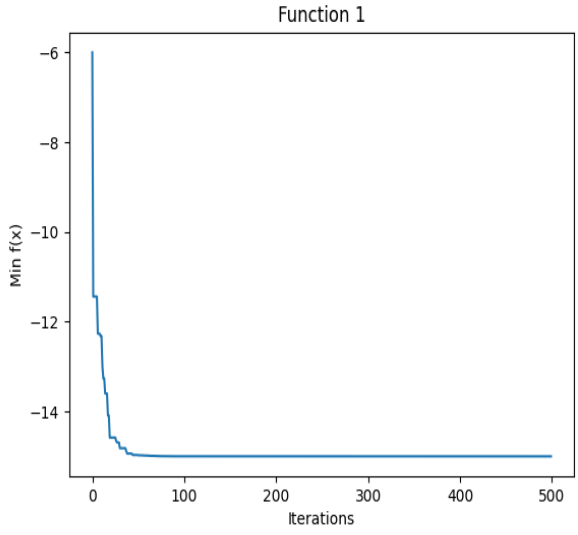

Figure 4: Graph representing the convergence of the fitness value towards its solution via TLBO implemented in Python for the first benchmark function.

TLBO Solution: [1,1,1,1,1,1,1,1,1,3,3,3,1]

Fitness Value: -15.0

Population: 50

Iterations: 500

Within the third benchmark function, we achieved a near match in terms of fitness values between the paper's implementation (fitness value = 680.6300) in MATLAB compared to our implementation of TLBO in Python (fitness value = 680.6925). This result can potentially be caused by the way Python deals with randomness or just not hitting a seed which gives an equivalent result through testing.

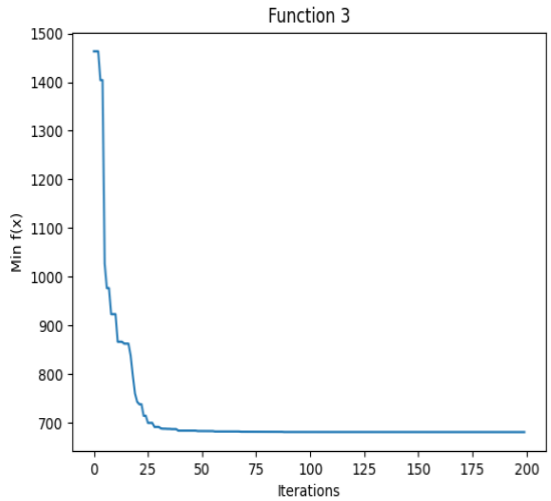

Figure 5: Graph representing the convergence of the fitness value towards its solution via TLBO implemented in Python for the third benchmark function.

TLBO Solution: [2.319, 1.956, -0.4437, 4.3556, -0.62662, 1.1149]
Fitness Value: 680.6925

Population: 50

Iterations: 200

The final function, the pressure vessel function, our implementation using a population of 50 and a maximum iteration total of 200 provided us with a fitness score of 5926. This is an improvement over the author's implantation inside of MATLAB, which has a fitness score of 6059. Similar to the third benchmark function, this could be caused simply by the nature of randomness in the tests and any difference in how randomness is handled between Python and MATLAB.

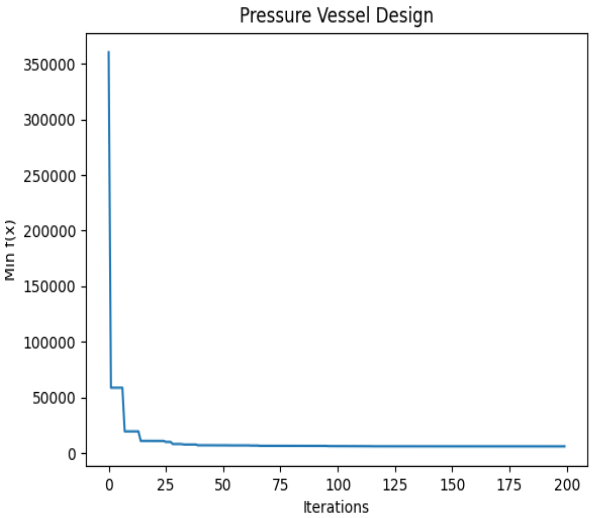

Figure 6: Graph representing the convergence of the fitness value towards its solution via TLBO implemented in Python for the pressure vessel function.

TLBO Solution: [0.78897, 0.39028, 40.78522, 194.03056]

Fitness Value: 5926.700

Population: 50

Iterations: 200

Conclusion: Within the paper, the TLBO algorithm was discussed, displaying what advantages it has over traditional algorithms such as GA or PSO. In addition, the implementation of TLBO in particular was discussed involving the teaching and learning phases it performs to get to a solution without needing fine tuning of particular parameters to achieve these results. The algorithm was also discussed on how it can be used to optimize functions in the branch of mechanical engineering to obtain minimum value of functions. Furthermore, these results are able to be reproduced in other implementations as shown by our implementation in Python compared to the author's implementation of TLBO inside of MATLAB.

References

[1] R. Rao, V. Savsani and D. Vakharia, "Teaching–Learning-Based Optimization: An optimization method for continuous non-linear large scale problems," *Information Sciences ,* vol. 183, pp. 1-15, 2012.

[2] R. Rao, V. Savsani and D. Vakharia, "Teaching–learning-based optimization: A novel method for constrained mechanical design optimization problems," *Computer-Aided Design ,* vol. 43, pp. 303-315, 2011.

[3] R. V. Rao, Teaching Learning Based Optimization Algorithm And Its Engineering Applications, Springer International Publishing , 2015.