Using Evolutionary Algorithms for Finding Optimal Initial Conditions in a Zero-Player Game

Rohit Priyanka Nandwani
Department of Physics
Indian Institute of Technology Guwahati
Guwahati, Assam, India - 781039
nandwani@iitg.ac.in

***Abstract*—A game that evolves as determined by its initial state, requiring no further input from humans is considered a zero-player game. In this report, we use several evolutionary algorithms to solve a physics-based zero-player maze game. The goal is to reach a region of space, called the "win-zone" by choosing appropriate initial components of velocity, as quickly as possible.**

***Index Terms*—Evolutionary Algorithms, Zero-Player Game, Initial Conditions, Physics**

I. Introduction

In this report, we will be working with two versions of the same problem. In each problem, there is a "massy" point-particle (called "ball") which can move around in space, following Newton's laws of motion. The "environment" in which the *ball* exists is filled with multiple fixed elastic walls and a "bucket" with walls as its boundary. The *environment* is also laden with zones, called "end-zones". If the ball enters an *end-zone*, the simulation ends.

The objective of the game is to reach a special region of *end-zone*, called the "win-zone". *Win-zone* is the *lid* at the top of the bucket, passing through which guarantees entry inside the *bucket*.

- In the first problem, the objective is to reach the win-zone, *without any consideration of the time-taken* to do so.
- In the second problem, the objective is to *reach the win-zone in the least amount of time* possible.

To solve the problem using metaheuristic techniques, the objective function to be minimized is introduced in the next section, along with the exact problem statement.

II. Problem Statement

A. Setup and Data Points

As mentioned in the introduction, the first problem consists of walls with coefficient of restitution, $e = 1$. There are 12 walls (*line segments*) spread throughout the region, with the starting and ending coordinates of line segments given in TABLE I. Also, see Fig. 1 to see the orientation of x and y axes, and the position of origin.
The ball's starting position is fixed, and given in TABLE II.

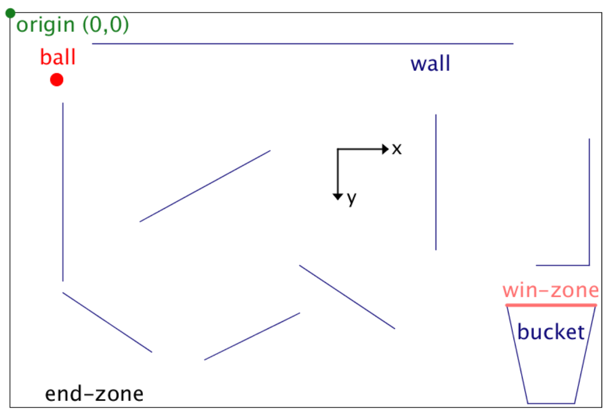

Fig. 1. Environment 1 : Elastic Walls

TABLE I
Coordinates of Walls in Environment-1

Wall	Point 1 (x,y) (in %)	Point 2 (x,y) (in %)
1	$(9.\bar{3}, 23)$	$(9.\bar{3}, 68)$
2	$(9.\bar{3}, 71)$	$(24, 86)$
3	$(22, 53)$	$(44, 35)$
4	$(14.\bar{6}, 8)$	$(85.\bar{3}, 8)$
5	$(33.\bar{3}, 88)$	$(49.\bar{3}, 76)$
6	$(49.\bar{3}, 64)$	$(65.\bar{3}, 80)$
7	$(72, 26)$	$(72, 60)$
8	$(84, 74)$	$(87.5, 99)$
9	$(99, 74)$	$(95.5, 99)$
10	$(89.\bar{3}, 64)$	$(98.\bar{6}, 64)$
11	$(98.\bar{6}, 32)$	$(98.\bar{6}, 64)$
12	$(87.5, 99)$	$(95.5, 99)$

TABLE II
Initial Coordinates of Ball in Environment 1

x-coordinate (in %)	y-coordinate (in %)
8	17

$$\text{width of the outer end-zone} = 760 \text{ units}$$
$$\text{height of the outer end-zone} = 506.\bar{6} \text{ units}$$

B. Objective Function - Problem 1

In all of the problems, the **decision variables** are the components of initial velocity imparted,

$$\mathbf{u} = (u_x, u_y)$$

These values are given as the initial conditions for the *simulation*. After the simulation ends (when the ball reaches end-zone, which includes the win-zone), the last coordinate of the ball is recorded.

Let

$\mathbf{r} \ = (x, y)$ be the last coordinate of the ball.

$\mathbf{r_c} = (x_c, y_c)$ be the mid-point of the win-zone.

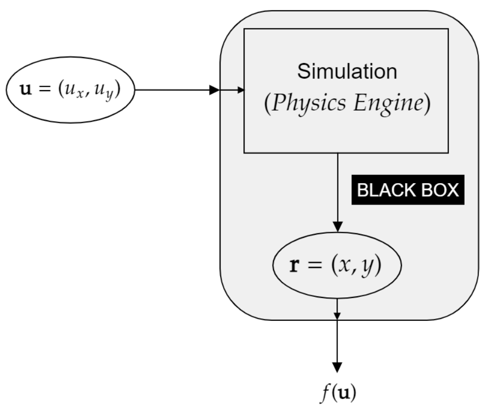

Fig. 2. Objective Function

TABLE III
COORDINATES OF MID-POINT OF WIN-ZONE

x-coordinate, x_c (in %)	y-coordinate, y_c (in %)
91.5	74

The objective function $f(\mathbf{u})$ is defined as follows:

$$f(\mathbf{u}) = \begin{cases} \mathrm{d}\left(\mathbf{r}, \mathbf{r}_c\right) & , \mathbf{r} \in \text{win-zone} \\ \lambda \cdot \mathrm{d}\left(\mathbf{r}, \mathbf{r}_c\right) & , \mathbf{r} \notin \text{win-zone} \end{cases}$$

,where the factor of $\lambda (= 10)$ is introduced to create a bias towards the win-zone, in contrast to the remaining end-zone.

C. Brief Explanation of the Physics Engine

What follows is the pseudocode for finding the objective function value ($f(\mathbf{u})$) for a given input value ($\mathbf{u}$).

```
u = [u_x, u_y]  // decision variables value
death = false
while death_is_false :
    update ball's position
    is the ball's new position on/outside the end-zone?
        YES :
            f(u) = λ· d(r, r_c)
            death = true
        NO :
            has the ball "crossed" the win-zone?
                YES :
                    f(u) = d(r, r_c)
                    death = true
                NO :
                    do nothing
    Check collision of the ball with each wall one-by-one,
    and update it's position and velocity accordingly.

return f(u)
```

Updating the ball's position simply involves the following :

$$x \rightarrow x + u_x$$
$$y \rightarrow y + u_y$$

The non-trivial part of the implementation entails two aspects :

- Checking for collision between a wall and a ball.

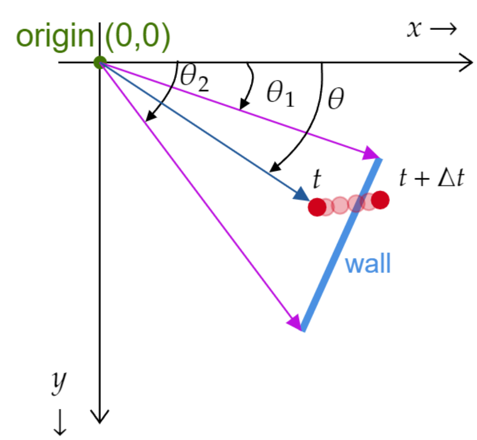

Fig. 3. Collision Detection with the Wall

A ball which is "going to" collide with a wall must satisfy two conditions:

(1) **Crossing**

The ball can be either on the wall or on one of its sides.

Assume the equation of the line, of which the wall is a segment (a line segment) is as follows:

$$LHS(x, y) = y - mx - c = 0$$

So, in general,

$$LHS(x(t), y(t)) < 0$$

or

$$LHS(x(t), y(t)) > 0$$

or

$$LHS(x(t), y(t)) = 0$$

At the moment of crossing, the condition which must then be satisfied is :

$$LHS(x(t), y(t)) \cdot \; LHS(x(t + \Delta t), y(t + \Delta t)) \leq 0$$

(2) **Range**

The crossing checks the condition of crossing across the infinitely stretching line, of which the wall is a part. To wean out the false line-segment crossings, we ensure that the angle subtended by the ball on the $x - axis$ is actually within the angles θ_1 and θ_2 shown in Fig. 3.

$$\begin{aligned}
&\theta \in [\theta_{min}, \theta_{max}] \\
&\theta = \arctan\left(\tfrac{y}{x}\right) \\
&\theta_1 = \arctan\left(\frac{y_1}{x_1}\right) \\
&\theta_2 = \arctan\left(\frac{y_2}{x_2}\right) \\
&\theta_{min} = \min(\theta_1, \theta_2) \\
&\theta_{max} = \max(\theta_1, \theta_2)
\end{aligned}$$

- Updating the ball's velocity after collision.

To update the velocity, using conservation of linear momentum, projections on the normal vector and a little bit of vector algebra, we end up with the following:

If wall is not vertical :
$$\begin{aligned}
&\text{change} = -2\tfrac{(-u_x \cdot m + u_y)}{(m^2+1)}, \text{ where } m \rightarrow \text{slope} \\
&u_x \rightarrow u_x - \text{change} \cdot m \\
&u_y \rightarrow u_y + \text{change}
\end{aligned}$$

If wall is vertical :
$$u_x \rightarrow -u_x$$

III. Results - Problem 1

A. Brute Force Shallow Search

Despite the fact of Problem 1 (*elastic walls*) being a relatively simple problem amongst these class of problems; the time required, number of computations performed, and plainly the number of objective function calls is significant.

Still a shallow search for "solutions" (velocity values that lead to particle ending in win-zone) was performed on the interval:

$$\begin{aligned}
u_x &\in [0, 5] \\
u_y &\in [-5, 5] \\
\text{step-size} &= 0.01
\end{aligned}$$

This will help us analyse the behaviour of metaheuristic techniques in this relatively-simpler search space.

*Note:

"Solutions" have objective function values less than or equal to half the width of win-zone.

$$\implies \mathbf{u} \text{ is a solution } \iff f(\mathbf{u}) \leq \frac{\text{width of win-zone}}{2} = 57$$

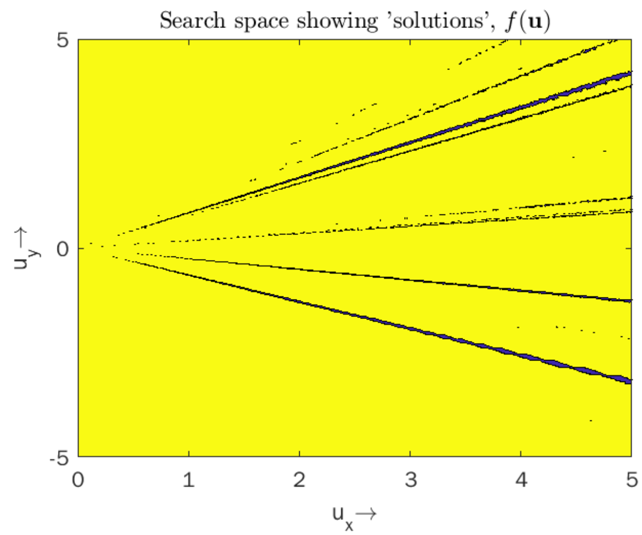

Fig. 4. Brute Force Shallow Search Results

This simple shallow search took 10 *to* 12 *hours* because of the complexity of the Objective function. Almost all solutions lie on linear *bands* passing through origin. The relatively sharp behaviour introduced due to the λ factor can be seen in Fig. 5 and Fig. 6.

B. Comparison of Algorithms

Considering the sharp behaviour of objective function, the algorithms (*and their respective parameters*), for which the performance was compared are given in TABLE IV.

Average behaviour of each of these algorithms starting from the same random population can be seen in Fig. 8, Fig. 9 and Fig. 10.

Mean behaviour of RCGA (for the chosen parameters atleast) is the worst. TLBO, PSO and ABC perform the best. One important thing to keep in mind is that these graphs only represent the average behaviour. This means all the outliers and random fluctuations will easily get damped leading to a more representative behaviour.

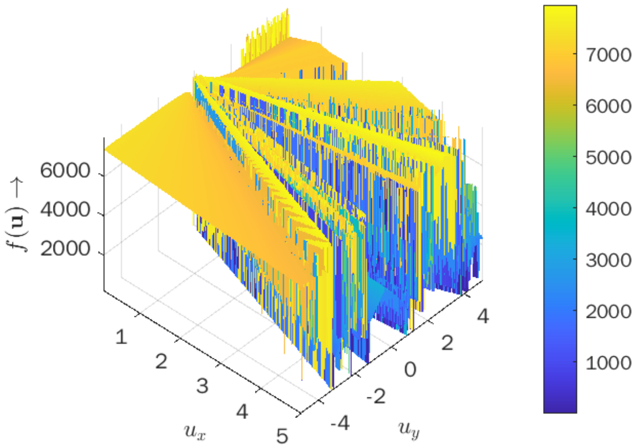

Fig. 5. Objective Function Plot

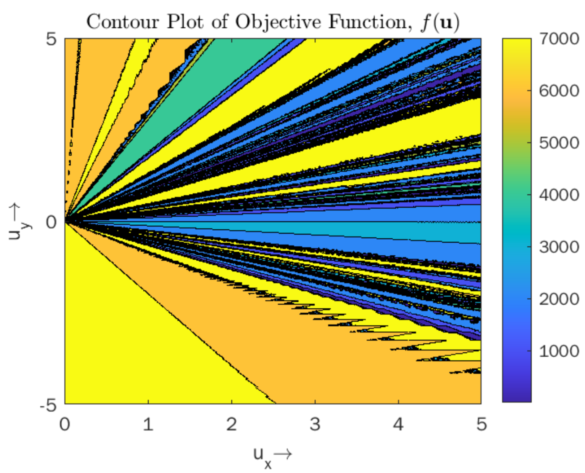

Fig. 6. Nature of the Objective Function

But, it would be unwise to not consider the total number of functional evaluations, as shown in TABLE IV. Clearly, keeping the complexity and time taken for calculating the objective function of a single value in mind, the *"best"* algorithm to choose amongst these 5 would be **Particle Swarm Optimization** (PSO).

Fig. 7 shows the distribution of swarm and global best solution for a single run (5 generations).

The performance of the metaheuristic technique can be seen as an animation at the following url :

https://git.io/JDvsN

IV. Fitness Function and Results - Problem 2

A. Fitness Function

In Problem 2, minimization of the time-taken or number-of-time-steps (# time-steps) is also an "objective" other than minimization of the original objective function [$f(\mathbf{u})$], as defined in II. *B*.

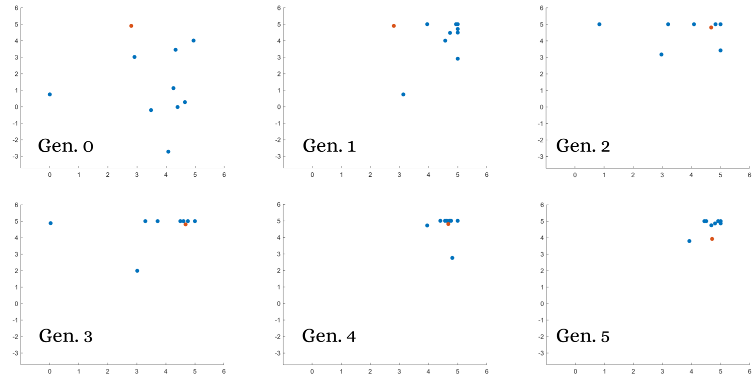

Fig. 7. Particle Swarm Optimization (first five generations)
Global Best and Swarm

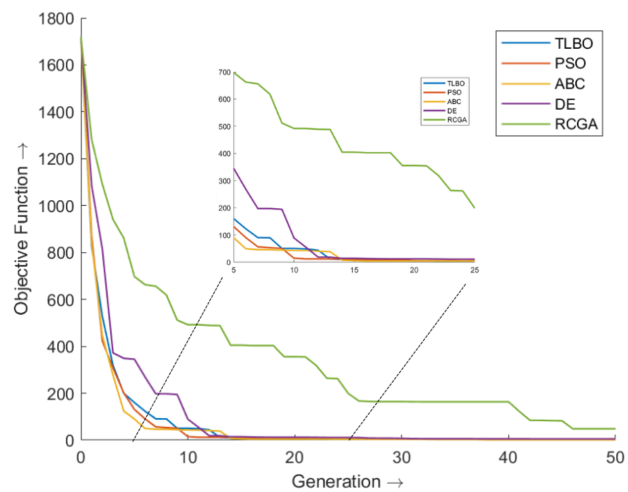

Fig. 8. Mean of Best Objective Function Values across 20 Runs

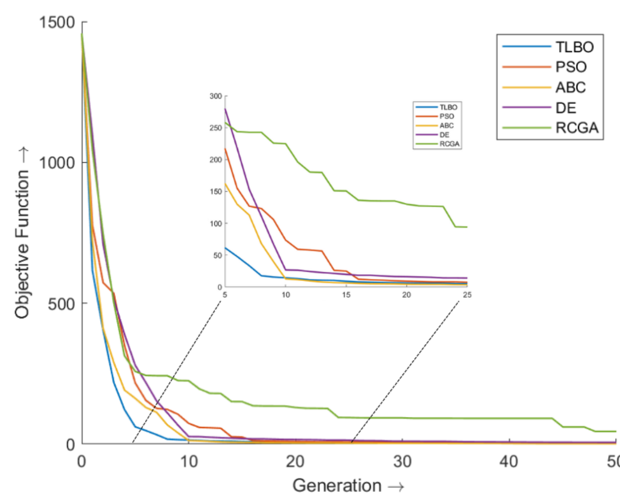

Fig. 9. Mean of Best Objective Function Values across 50 Runs

The fitness function $fit(\mathbf{u})$ is defined as follows:

$$fit(\mathbf{u}) = \begin{cases} \mathrm{d}\left(\mathbf{r},\mathbf{r}_c\right) + \gamma_0 \left(ts\right)^{\beta_0} & , \mathbf{r} \in \text{win-zone} \\ \lambda \cdot \mathrm{d}\left(\mathbf{r},\mathbf{r}_c\right) + \gamma_1 \left(ts\right)^{\beta_1} & , \mathbf{r} \notin \text{win-zone} \end{cases}$$
$$= f(\mathbf{u}) + \boldsymbol{\gamma} \left(ts\right)^{\boldsymbol{\beta}}$$

TABLE IV
LIST OF METAHEURISTIC TECHNIQUES USED

Algorithm	Parameters
TLBO	$N_p = 10$ $T = 50$ $\#FE = 1010$
PSO	$N_p = 10$ $T = 50$ $w = 0.7298$ $c_1 = 1.496$ $c_2 = 1.496$ $\#FE = 510$
DE/rand/1/binomial	$N_p = 10$ $T = 50$ $p_c = 0.8$ $F = 0.8$ $\#FE = 510$
GA (real-coded)	$N_p = 10$ $T = 50$ $\eta_c = 20$ $p_c = 0.8$ $\eta_m = 20$ $p_m = 0.2$ $\max \#FE = 510$
ABC	$N_p = 10$ $T = 50$ $limit = 500$ $(= N_p \cdot T)$ $\max \#FE = 1060$

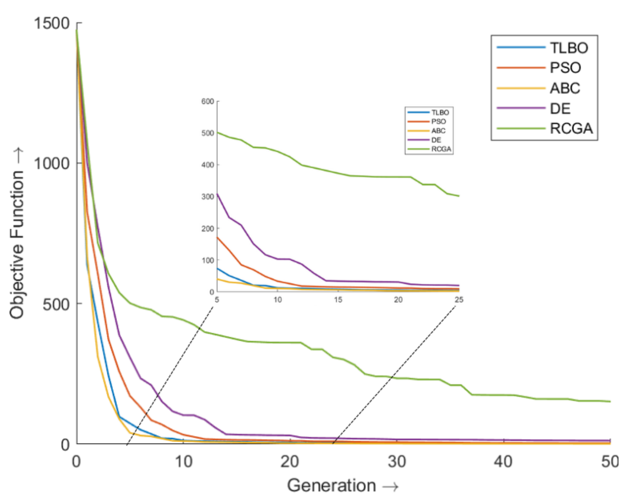

Fig. 10. Mean of Best Objective Function Values across 100 Runs

where

$$ts = \text{number-of-time-steps}$$

$$\boldsymbol{\gamma} = \begin{pmatrix}\gamma_0\\ \gamma_1\end{pmatrix} = \begin{pmatrix}4\\ 1\end{pmatrix}$$

$$\boldsymbol{\beta} = \begin{pmatrix}\beta_0\\ \beta_1\end{pmatrix} = \begin{pmatrix}1\\ 1\end{pmatrix}$$

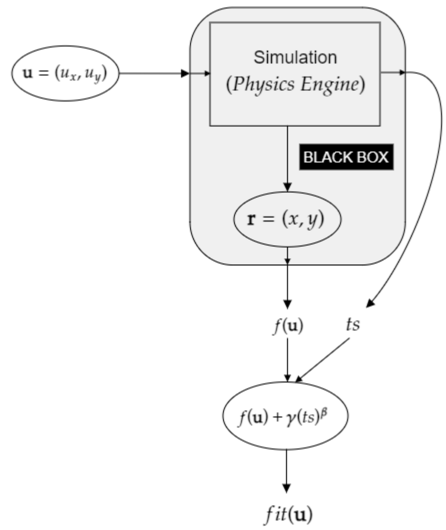

Fig. 11. Fitness Function

The above values of $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ were chosen empirically so as to favour minimization of objective function more than the time-steps. At the same time, minimization of the time-steps is not completely neglected.

B. Comparison of Algorithms

The results of all the listed metaheuristic techniques can be seen in Fig. 12, Fig. 13 and Fig. 14.

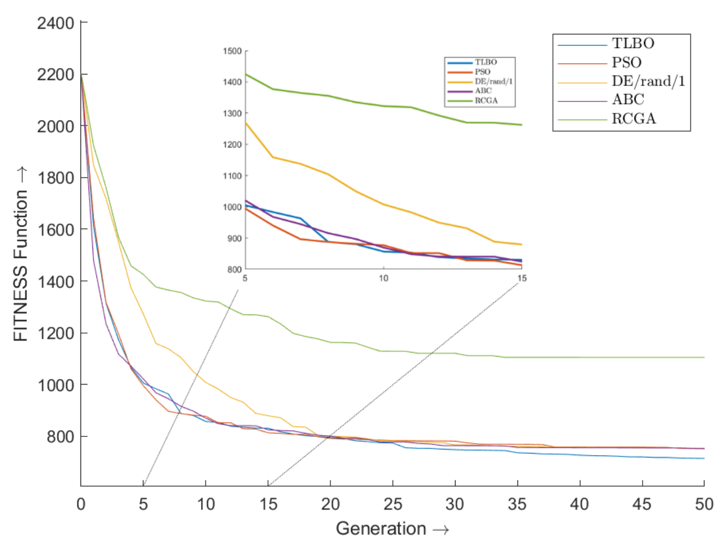

Fig. 12. Mean of Best Fitness Function Values across 50 Runs

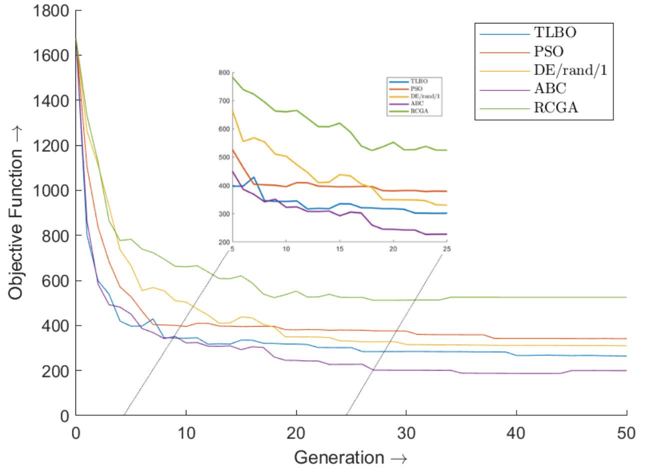

Fig. 13. Mean of Corresponding Objective Function Values across 50 Runs

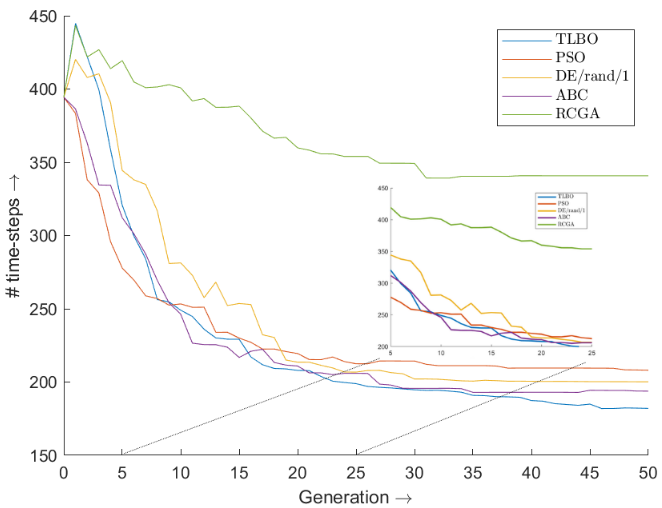

Fig. 14. Mean of Corresponding "time-step" Values across 50 Runs

In this adapted fitness function definition, once again RCGA performs the worst on average. Performance of ABC, PSO and TLBO is equivalent when it comes to the minimization of fitness function $fit(\mathbf{u})$. But with respect to objective function $f(\mathbf{u})$, performance of ABC is better; and with respect to #time-steps, performance of TLBO is better as generations progress.

V. Conclusion

We assigned an objective function to an abstract and complicated physics-based zero-player game. Using this definition of objective function, we applied the existing metaheuristic techniques. The algorithms performed very well and the "solutions" (ball landing up inside the bucket) were attained within a few generations. We also compared the various algorithms with each other, choosing the parameters required for the algorithms using empirical analysis.

Then, we added an additional objective of minimization of the time taken for ball to end up in the end-zone. Using the weighted sum method for multi-objective optimization and choosing the weights (representing relative importance of objectives), we were successful in getting "solutions" as well as minimizing the time taken to complete the first objective (inside the simulation).

Originally, we had planned to take this problem up a notch, by working with inelastic walls, massy finite-sized "planets" with gravitational fields, massless finite-size "charges" with electrostatic field and viscous mediums (where the ball experiences a resistive force too). We were unable to do so in this report. In future, we might try applying the metaheuristic techniques for same.

A sincere thank you to Prof. Prakash Kotecha, IITG for teaching us a memorable and interesting course on Computer-Aided Applied Optimization Techniques.

References

[1] L. Zadeh, "Optimality and non-scalar-valued performance criteria," in IEEE Transactions on Automatic Control, vol. 8, no. 1, pp. 59-60, January 1963, doi: 10.1109/TAC.1963.1105511.
[2] Martin Gardner (October 1970), "Mathematical games: The fantastic combinations of John Conway's new solitaire game 'Life'", Scientific American.
[3] Reas, C. and Fry, B. Processing: programming for the media arts (2006). Journal AI Society, volume 20(4), pages 526-538, Springer.