Reduced Yin-Yang-Pair Optimization and its Performance on the CEC 2016 Expensive Case

Varun Punnathanam
Department of Chemical Engineering
Indian Institute of Technology Guwahati
Guwahati, India

Prakash Kotecha
Department of Chemical Engineering
Indian Institute of Technology Guwahati
Guwahati, India
pkotecha@iitg.ernet.in

***Abstract* — We present a variant of the Yin-Yang-Pair Optimization metaheuristic which accommodates computationally expensive numerical optimization problems more effectively. The variations are directed at simplifying the algorithm as well as enabling it to utilize functional evaluations more efficiently. The performance of Reduced Yin-Yang-Pair Optimization is tested on the functions used for the CEC 2016 Special Session on Bound Constrained Single-Objective Computationally Expensive Numerical Optimization. Additionally, the performance of the algorithm is evaluated based on the criteria specified for the same. The RYYPO is shown to have a significantly low time complexity while simultaneously providing highly competitive performance.**

Keywords* — *Single objective optimization; Yin-Yang-Pair Optimization; Computationally Expensive Numerical Optimization

I. Introduction

Real world optimization problems can often be multi-modal, non-differentiable or even discontinuous, which encourages the usage of metaheuristics and evolutionary techniques. Some of the well-established stochastic techniques are Genetic Algorithm [1], Differential Evolution [2], Particle Swarm Optimization [3] and Ant Colony Optimization [4]. In addition, several metaheuristics have been proposed in recent times such as Virus Colony Search [5], Moth-flame Optimization [6], Stochastic Fractal Search [7] and Grey Wolf Optimizer [8]. These algorithms are primarily derivative-free population-based iterative techniques which attempt to improve the fitness of its population of solutions in successive iterations. In general, these techniques consider population sizes ranging from 10 to 200 (even more in many cases), depending on the problem requirements, and are executed until a specified termination criteria is satisfied, which could potentially lead to requiring a large number of functional evaluations. Problems involving complicated and time consuming objective functions, such as problems involving computational fluid dynamics or Monte Carlo simulations, are frequently encountered in literature. The optimization of such problems could potentially be a highly time consuming process, due to which importance is being given in recent times to algorithms which can provide good results utilizing a very low number of functional evaluations.

The Yin-Yang-Pair Optimizer (YYPO) [9] is a recently developed low complexity metaheuristic which considers two initial points and attempts to explicitly balance exploration and exploitation of the search space. The search is done with the help of two stages, (i) the splitting stage, which is encountered in every iteration and creates new candidate solutions, and (ii) the archive stage, which is encountered at dynamic intervals of iterations and modifies the search parameters. The algorithm was shown to have a very low time complexity [9] and hence provides an attractive option for applications where computational time is a critical constraint. Additionally, YYPO was shown to provide competitive performance on the real parameter single objective CEC 2013 benchmark problems.

In its original form, YYPO has a fixed number of functional evaluations per iteration (four times the number of decision variables). By modifying the splitting stage, the functional evaluations per iteration of Reduced Yin-Yang-Pair Optimization (RYYPO) employed here can vary between 2 functional evaluations to 4 times the number of decision variables, thus enabling the algorithm to potentially have a larger number of iterations before the maximum limit on functional evaluations is encountered.

The paper is organized as follows: The basic YYPO is described in Section II, following which the variations proposed are described in Section III. The experimental settings are presented in Section IV. Subsequently, the results, including the algorithm time complexity, are provided in Section V, following which we conclude the work in Section VI.

II. Yin-Yang-Pair Optimization

The interaction between the explorative and exploitative aspects of an optimization technique largely governs its performance. These two aspects are complimentary yet conflicting in nature, and can be correlated to Yin and Yang in Chinese philosophy. Just as a balance between Yin and Yang is considered to bring harmony to the universe, YYPO attempts to maintain a balance between exploration and exploitation so as to perform well on a problem.

YYPO employs two points (*P1* and *P2*), where *P1* is designed to focus on exploitation and *P2* is designed to focus on exploration, and attempts to improve their fitness in successive iterations. These points act as centers and explore the hypersphere volume defined by search radii δ_1 and δ_2 respectively. The search radii are self-adapting parameters

978-1-5090-0623-6/16/$31.00

which are reduced/increased to simulate contracting/expanding search volumes. Considering a small search radii leads to the creation of new points in the close neighborhood of the parent point, thus the algorithm focuses on exploration. Conversely, having a large search radii leads to creation of new points which are more spaced apart, thus allowing the algorithm to explore a larger volume of the search space.

Modelling the algorithm

YYPO handles all the decision variables in their normalized form (between 0 and 1) which are linearly scaled to its original bounds for functional evaluations with the aid of the lower and upper limits of the variables. Two points are generated randomly (between 0 and 1) and are evaluated for their fitness values, with the fitter point being *P1* and the other *P2*. The user defined parameters corresponding to minimum and maximum number of archive updates (I_{min} and I_{max}) and the expansion/contraction factor (α) are to be specified and the search radii δ_1 and δ_2 are set at an initial value of 0.5 each. The number of archive updates (I) is then randomly generated between the limits I_{min} and I_{max}, following which the iteration loop is initiated. Each iteration starts with a comparison of the fitness of the two points (*P1* and *P2*) which are interchanged if needed so that *P1* denotes the fitter point and *P2* the other. This step is to ensure that the algorithm focuses its exploitation around *P1*. It should be noted that if *P1* and *P2* are interchanged, their corresponding search radii δ_1 and δ_2 are interchanged as well. Subsequently, both the points (as well as their fitness) are stored in an archive (this corresponds to an archive update) and the algorithm proceeds to the splitting stage. The points *P1* and *P2* individually undergo the splitting stage along with their corresponding search radii δ_1 and δ_2.

1. *Splitting stage*

This stage is designed to explore the variable space around a specific point *P* and defined by the corresponding search radii δ. In this stage, 2*D* (where *D* denotes the number of decision variables) new points are generated around *P* and their closeness to *P* is determined by δ. The new points are generated by one of the following two methods with equal probability.

a. *One-way splitting:* 2*D* copies of the point *P* are stored as *NP* Subsequently, one variable of each point in *NP* is modified using the following equations.

$$NP_j^j = NP^j + r\delta \quad and$$
$$NP_{D+j}^j = NP^j - r\delta, \quad where\ j = 1,2,3\ldots D \tag{1}$$

Here, superscript denotes the variable number, the subscript denotes the point number and *r* is a uniformly generated random number between 0 and 1 and generated uniquely for each point in *NP*.

b. *D-way splitting:* While only one variable of each point is modified in the one-way splitting, all variables of each point are varied in the D-way splitting method. Initially *2D* copies of the point *P* are made and stored as *NP,* same as one-way splitting. An additional binary matrix *B* of size 2*D* x *D* is generated in this case, such that the binary string corresponding to each row is unique among the 2*D* strings. Each point in *NP* is subsequently modified using the following equation.

$$NP_k^j = P^j + r\left(\delta/\sqrt{2}\right) \quad if\ B(k,j) = 1,$$
$$NP_k^j = P^j - r\left(\delta/\sqrt{2}\right) \quad else. \tag{2}$$
$$where\ k = 1,2,3\ldots 2D\ and\ j = 1,2,3\ldots D$$

In this case, a unique random number *r* is generated for each variable of each point.

At this stage, the modified variable values can potentially violate the variable bounds, in which case the variable is assigned a randomly generated value (between 0 and 1) instead. Either of the two methods generates 2*D* new points, which are subsequently evaluated. Thus, the number of functional evaluations per iteration in YYPO is dependent on the dimension of the problem (number of decision variables). The best point among the 2*D* points in *NP*, irrespective of whether or not it is fitter than the point *P*, replaces the point *P*. The previous point *P* is not discarded, as it was stored in an archive before it underwent the splitting stage. Thus, the splitting stage concludes with both the points *P1* and *P2* individually being replaced with new points. At the current state, if the number of archive updates is equal to *I*, the algorithm proceeds to the archive stage. Otherwise, this completes the iteration and, if the termination criteria has not been satisfied, the algorithm proceeds to the subsequent iteration.

2. *Archive stage*

Initialization of the archive stage signifies that *I* archive updates have been made. Additionally, as each update corresponds to two points being added, the archive presently contains 2*I* points. The best point in the archive, if better than the point *P1*, is interchanged with *P1* (i.e. the previous *P1* is now in the archive). Subsequently, the best point in the current archive (which may contain the previous *P1*), if better than point *P2*, replaces *P2*. Thus, the best two points discovered by the algorithm always survives successive iterations. The search radii δ_1 and δ_2 are subsequently updated using the following equations.

$$\delta_1 = \delta_1 - \left(\delta_1/\alpha\right)$$
$$\delta_2 = \delta_2 + \left(\delta_2/\alpha\right) \tag{3}$$

Reduction of δ_1 leads to the contraction of the volume searched around *P1* (in the splitting stage) while increasing δ_2 leads to the expansion of the volume searched around *P2*. As a high value of the search radii could lead to an ineffective search, the value of δ_2 is limited to a maximum value of 0.75. Subsequently, the archive is emptied and a new value of *I* is generated between its limits (I_{min} and I_{max}). This completes the archive stage and thus the current iteration.

III. RYYPO

In order to fully utilize each functional evaluation being executed by the algorithm, we employ a variant of the base YYPO for the computationally expensive problems. In particular, two major variations are made to the pre-existing framework of the splitting stage. The pseudo-code for the modified splitting function is provided in Algorithm 1 and the variations are described below.

Algorithm 1. Pseudo-code for RYYPO splitting function
Input: Point P and its corresponding δ.
Output: Updated point P along with its fitness.

```
1:  Generate 2D copies of P and store it in NP.
2:  Modify each point in NP using eq. (1).
3:  If any variable in NP is lesser than 0 or greater than 1,
    replace it with a randomly generated number between
    0 and 1.
4:  Generate a random permutation of integers between 1
    to 2D and store it in ord
5:  for j = 1 to 2D
6:    Evaluate fitness of point ord(j) in NP
7:    if point ord(j) in NP is fitter than point P
8:      Point ord(j) in NP replaces P
9:      break for-loop
10:   else
11:     count = count + 1, end
12: end for-loop
13: if count equals 2D
14:   fittest point in NP replaces point P, end
```

A. Updation of P in the splitting stage

In the base YYPO, the splitting stage creates *2D* new points corresponding to a point *P*, and evaluates their respective fitness. Hence, considering points *P1* and *P2*, the algorithm utilized a total of 4*D* functional evaluations per iteration. While this allows for a more thorough search of the variable space, it was observed to be excessive for cases with a very low number of maximum permissible functional evaluations. Obtaining a point fitter than the point which entered the splitting stage is the priority in this case and sufficient condition for terminating the splitting stage. Hence, in the variant employed, we evaluate a new point if and only if the previous point was worse than the parent point (*P*). The 2*D* points in *NP* are evaluated one at a time (with the points taken in a random order), until a point fitter than *P* is encountered. In this case, the fitter point replaces *P* and the splitting stage is terminated. On the other hand, if none of the newly generated point is fitter than *P*, the fittest among the newly generated point replaces the point *P*. This case is exactly similar to the splitting stage in the base YYPO. Thus, the variant splitting stage can employ anywhere between 2 to 4*D* functional evaluations per iteration instead of the fixed 4*D* functional evaluations per iteration utilized by the base splitting stage.

B. Only one-way splitting employed

The second variation to the base YYPO splitting function is directed at further simplifying the algorithm. Since the computing resources utilized and time constraints are of paramount importance in the computationally expensive case of the CEC 2016 problems [10], we shed the significant amount of computational weight which is required by the D-way splitting aspect of the base YYPO. The justification of this removal is given below.

- As opposed to the one-way splitting (which modifies a single variable in each point), the D-way splitting requires every variable in every point to be modified. This corresponds to $(D - 1)$ additional changes to be incorporated for each new point.
- The modification of each variable corresponds to the generation of a unique random number (between 0 and 1) which is to be multiplied to the search radii δ. This in turn translates to the D-way splitting requiring the generation of $(D - 1)$ additional random numbers as compared to the one-way splitting.
- D-way splitting requires an additional step of generating a binary matrix B of size $2D$ x D such that each row forms a unique binary string. This is a computationally involved step which was done by generating $2D$ unique integers between 0 and $2^D - 1$, and subsequently converting them to binary strings. The binary string is required to define the direction in which each new point would be generated by the D-way splitting, and is not required for one-way splitting.

Thus, employing only the one-way splitting leads to a significantly faster search algorithm.

IV. Experimental settings

The performance of RYYPO is evaluated on the benchmark problems specified in the computationally expensive case of the CEC 2016 Competition on Real-Parameter Single Objective Optimization [10] using the evaluation criteria specified for this competition. It should be noted that the CEC 2016 competition considers the same problems as the CEC 2015 expensive case competition. The fifteen benchmark functions as well as their global optima values are as follows:

- Unimodal functions
  - f_1: Rotated Bent Cigar Function (100).
  - f_2: Rotated Discus Function (200).
- Simple multimodal functions
  - f_3: Shifted and Rotated Weierstrass Function (300).
  - f_4: Shifted and Rotated Schwefel's Function (400).
  - f_5: Shifted and Rotated Katsuura (500).
  - f_6: Shifted and Rotated HappyCat (600).
  - f_7: Shifted and Rotated HGBat Function (700).
  - f_8: Shifted and Rotated Expanded Griewank's plus Rosenbrock's Function (800).
  - f_9: Shifted and Rotated Expanded Scaffer's F6 Function (900).

- Hybrid functions
  - f_{10}: Hybrid Function 1 (N=3) (1000).
  - f_{11}: Hybrid Function 2 (N=4) (1100).
  - f_{12}: Hybrid Function 3 (N=5) (1200).
- Composition functions
  - f_{13}: Composition Function 1 (N=5) (1300).
  - f_{14}: Composition Function 2 (N=3) (1400).
  - f_{15}: Composition Function 3 (N=5) (1500).

The termination criteria was fixed at a maximum of 50*D* functional evaluations. The fifteen functions were evaluated at dimensions of D = 10 and 30. The algorithm parameters were not rigorously tuned. They were empirically tested (using trial and error) over 10 complete runs (each run consisting of all the fifteen functions, 20 random instances and the two dimension settings). Each complete run required around 24 seconds of computing time, hence the complete tuning process required a total of about 4 minutes of computing time. The score corresponding to each trial run was calculated using the mechanism provided in the CEC competition report [10] and are provided in Table I. The following parameter values: (i) I_{min} = 2, (ii) I_{max} =3 and (iii) α = 4 corresponds to the least error obtained from the 10 trials, and hence is utilized for generating the results in this work. All the instances were executed on a computer system with the Intel i7 3.40 GHz processor, 6 GB RAM and the Windows 7 operating system. The RYYPO was coded and evaluated benchmark functions using MATLAB R2015a.

TABLE I. SCORES OBTAINED ON TRIALS

Trial	I_{min}	I_{max}	α	Score
1	2	2	10	1.870E+09
2	2	3	10	2.009E+09
3	2	4	10	2.175E+09
4	1	2	10	1.645E+09
5	1	3	10	2.038E+09
6	1	4	10	2.055E+09
7	2	3	15	2.448E+09
8	2	3	5	1.271E+09
9	**2**	**3**	**4**	**9.569E+08**
10	2	3	3	3.770E+09

V. RESULTS AND DISCUSSION

The base YYPO and its current variant RYYPO was developed with emphasis on creating a very low complexity optimzation tool. Hence, the motivation here is to provide a performance competitive to current state-of-the-art techniques, while being superior with respect to the algorithm time complexity. For the current set of 15 benchmark functions and the evaluation criteria employed, the RYYPO was observed to perform better than the base YYPO. The error values obtained by RYYPO for the 10*D* and 30*D* cases as well as the time complexity of the algorthm are discussed below.

A. Error values obtained

Table II and Table III provide the best, worst, median, mean and standard deviations corresponding to each of the fifteen benchmark functions and the two problem dimensions (Table II corresponds to the 10*D* case while Table III corresponds to the 30*D* case). Here, the terms best, worst, median and mean refer to the best, worst, median and mean of the error values obtained for each function and dimension over the 20 runs (error value is the difference between the obtained fitness and the global optimum of the specific function). The following are a few observations which can be inferred based on these tables:

- The results presented in Table II and Table III show that the algorithm provides a reliable and consistent performance for most of the functions on both dimensions.
- For many functions ($f_5 - f_7$, f_9, f_{11}, f_{13} and f_{14}), the mean fitness obtained by the algrothm remains largely unaffected with increase in the problem dimension from 10*D* to 30*D*.
- The median values obtained for many functions are very close to the best value obtained for the corresponding function. This is a trend which is common for both dimensions on the functions f_3, f_6, f_9, f_{13} and f_{14}, implying that RYYPO is relatively consistent in its performance on these functions and remains largely unaffected by the random seed.
- The standard deviation obtained for many functions (f_3, f_5, f_6, f_7, f_8, f_9 and f_{11}) is significantly low, which is another indication of consistency on these functions.
- The lower value of the mean error was obtained for the functions f_6 and f_7 on both the dimensions. These two functions correspond to the least best, worst and median error values as well.
- The significant exceptions are functions f_1, f_2 and f_{10}, on which the algorithm is not as effective. This includes both the unimodal funtions in the set (f_1 and f_2). As RYYPO emphasizes on speed rather than accuracy, the overhead communication and computation is maintained at a minimum, which could potentially be the reason for it being unable to perform well on the two rotated unimodal problems.
- The error values obtained for the remaining hybrid functions and composite functions (f_{11} to f_{15}) are relatively high as well. This is not surprising as these functions are reported to be very difficult.
- The best performance among the hybrid and composite functions is observed for f_{11}.

TABLE II. ERROR VALUES FOR 10*D* CASE

F	Best	Worst	Median	Mean	Std. Dev.
f_1	6.80E+06	5.62E+07	1.78E+07	2.41E+07	1.72E+07
f_2	1.28E+04	7.63E+04	4.43E+04	4.74E+04	1.88E+04
f_3	5.58E+00	1.30E+01	8.96E+00	9.01E+00	2.01E+00
f_4	5.04E+01	3.92E+02	2.22E+02	2.29E+02	9.02E+01
f_5	9.86E-01	3.13E+00	1.90E+00	2.03E+00	6.02E-01
f_6	4.14E-01	1.17E+00	6.90E-01	6.98E-01	1.91E-01
f_7	2.36E-01	1.64E+00	7.25E-01	7.62E-01	4.21E-01
f_8	3.91E+00	1.05E+01	6.41E+00	6.42E+00	1.56E+00
f_9	3.10E+00	4.50E+00	4.01E+00	3.95E+00	3.55E-01
f_{10}	5.01E+03	7.06E+06	3.25E+05	1.32E+06	2.07E+06
f_{11}	4.82E+00	1.83E+01	1.01E+01	1.04E+01	3.21E+00
f_{12}	3.82E+01	4.07E+02	2.27E+02	2.25E+02	1.06E+02
f_{13}	3.22E+02	3.94E+02	3.44E+02	3.50E+02	1.91E+01
f_{14}	1.91E+02	2.61E+02	2.09E+02	2.14E+02	1.48E+01
f_{15}	2.01E+01	6.12E+02	4.44E+02	4.53E+02	1.22E+02

TABLE III. ERROR VALUES FOR THE 30*D* CASE

F	Best	Worst	Median	Mean	Std. Dev.
f_1	3.05E+07	1.77E+08	5.91E+07	7.94E+07	4.65E+07
f_2	9.60E+04	2.08E+05	1.58E+05	1.53E+05	3.30E+04
f_3	3.32E+01	4.37E+01	3.81E+01	3.86E+01	2.95E+00
f_4	2.98E+02	1.12E+03	5.04E+02	5.96E+02	2.18E+02
f_5	1.15E+00	4.58E+00	2.61E+00	2.58E+00	8.29E-01
f_6	3.45E-01	8.14E-01	5.65E-01	5.69E-01	1.31E-01
f_7	2.79E-01	1.09E+00	4.40E-01	5.52E-01	2.43E-01
f_8	3.57E+01	1.20E+03	2.45E+02	3.47E+02	3.68E+02
f_9	1.30E+01	1.44E+01	1.36E+01	1.37E+01	3.60E-01
f_{10}	4.45E+06	3.34E+07	9.40E+06	1.30E+07	8.08E+06
f_{11}	2.31E+01	1.90E+02	8.86E+01	8.70E+01	6.10E+01
f_{12}	3.96E+02	1.32E+03	1.00E+03	9.44E+02	2.74E+02
f_{13}	3.46E+02	4.32E+02	3.82E+02	3.84E+02	2.44E+01
f_{14}	2.31E+02	3.70E+02	2.80E+02	2.90E+02	4.02E+01
f_{15}	5.32E+02	1.52E+03	1.17E+03	1.07E+03	3.40E+02

The convergence curves corresponding to the mean for 10D as well as 30D are provided in Figure 1. The mean of the best fitness obtained by the algorithm for each of 15 functions at specific functional evaluations over 20 runs are plotted against the fraction of maximum number of functional evaluations. Note that the semilog convergence curves are provided for a few functions (f_1, f_2, f_8, f_{10} and f_{12}) for a better understanding. The following are a few observations based on the convergence curves:

- The convergence is appreciably fast for most functions with the exceptions of f_1, f_3 and f_9.
- Although the objective function value reported by the algorithm for f_1 and f_{10} are poor, it can be observed that the algorithm has not converged for either of the functions. This suggests that RYYPO could potentially perform better on these functions with additional functional evaluations.
- On the other hand, the algorithm is observed to have converged to a solution for majority of the problems (all except f_1, f_4, f_5 f_8 and f_{10}).
- The convergence observed in the 10D case is relatively faster than 30*D* for most cases with the exception of f_2 and f_6.
- The mean error values for most functions at half the maximum functional evaluations is very near to the final error values. This suggests that the algorithm could potentially provide a similar level of performance with a much lower number of functional evaluations.

From Table I, it can be observed that RYYPO obtains a score of 9.569E+08 using the mechanism provided in the CEC2016 competition report. The base YYPO on the other hand is observed to obtain a score of 5.698E+10 by utilizing the same experimental settings as RYYPO. Thus, it can be observed that the modifications employed significantly improve the performance of the existing YYPO framework on the expensive case CEC 2015 benchmark problems.

B. Algorithm complexity

The CEC 2016 expensive case competition requires information related to the computational complexity of the algorithm [10]. The computational complexity table in the required format is provided in Table IV. The value of *T0* was obtained by running the sample block of code provided in the report [10], and was obtained as 0.1055 seconds. $\overline{T}1$ is calculated as the mean of the computing time (with 50*D* functional evaluations as the termination criteria) required by the algorithm for a specific function at each of the two dimension (10*D* and 30*D*) over 20 runs of each dimension. The complexity corresponding to each dimension is given as $\overline{T}1/T0$. It can be observed that the computational time required by the algorithm is exceptionally low, with the complexity increasing with the dimension of the problem.

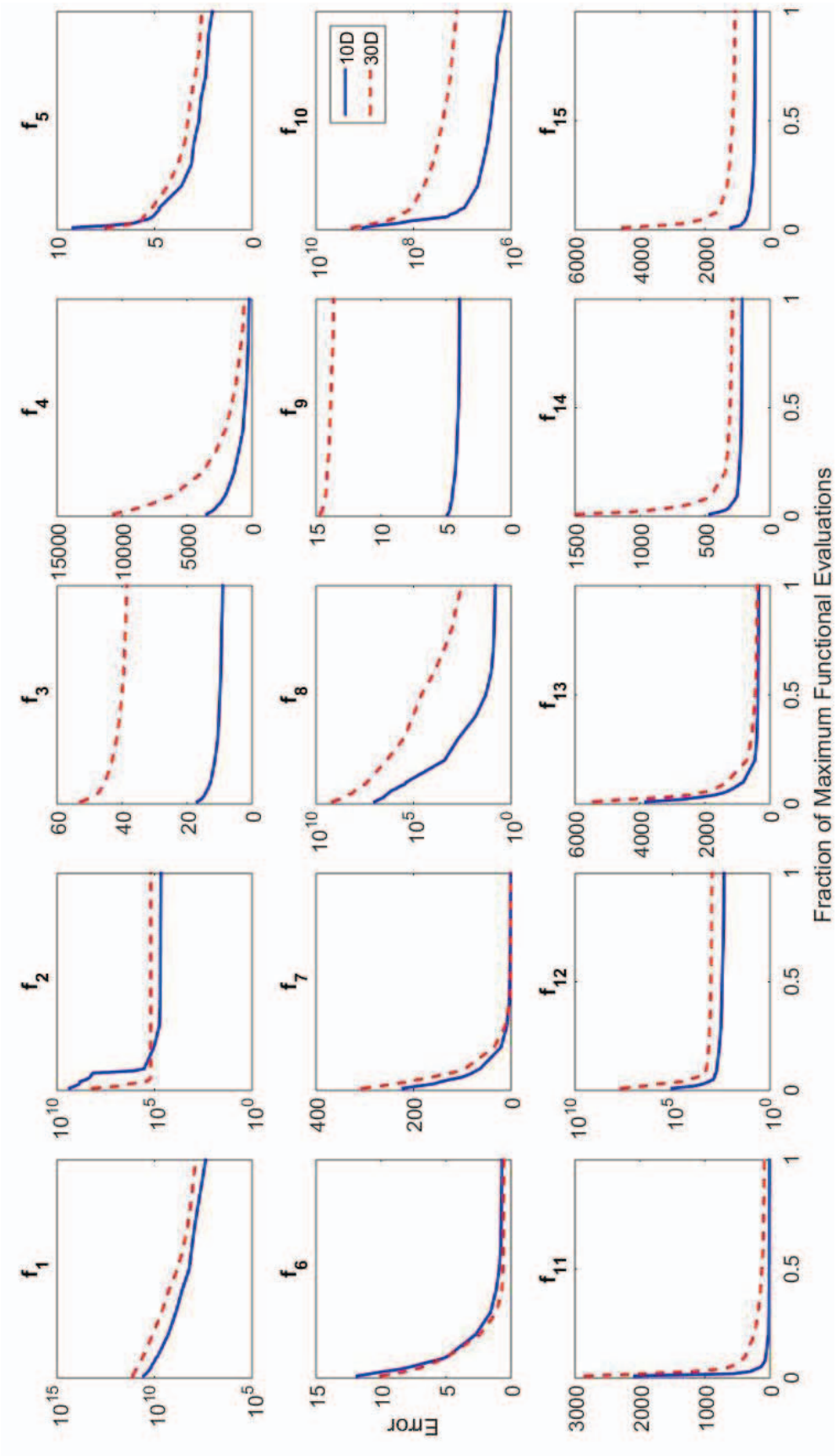

Fig. 1. Convergence curves of means for 10*D* and 30*D* cases

TABLE IV. COMPUTATIONAL COMPLEXITY ON 10D AND 30D CASES

Function	10D	30D
f_1	0.1100	0.3345
f_2	0.0953	0.2679
f_3	0.2353	1.5099
f_4	0.1077	0.3362
f_5	0.1678	0.9154
f_6	0.1027	0.3050
f_7	0.1075	0.3133
f_8	0.1065	0.3597
f_9	0.0947	0.2784
f_{10}	0.0990	0.3326
f_{11}	0.1319	0.5852
f_{12}	0.1075	0.3697
f_{13}	0.1255	0.5519
f_{14}	0.1216	0.4581
f_{15}	0.2654	1.7348

VI. CONCLUSION

A variant of the Yin-Yang-Pair Optimization was proposed in this work to handle problems with a significantly low number of permissible functional evaluations. Two major modifications were proposed for the splitting stage, one to restrict the number of functional evaluations and the other with the intention of speeding up the algorithm. The first modification ensures that the fitness of a newly generated solution in the splitting stage is evaluated only if the fitness of the previously evaluated solution is worse than the parent solution (point *P1* or *P2*). The second modification is the removal of the D-way splitting stage, thus employing only the one-way splitting. This is incorporated due to the fact that the one-way splitting is significantly less computationally intensive.

The performance of RYYPO is evaluated as per the criteria specified for the CEC 2016 Special Session on Bound Constrained Single-Objective Computationally Expensive Numerical Optimization. The algorithm is shown to give a highly competitive performance while having exceptionally low computational complexity. Additionally, the convergence of the algorithm was observed to be appreciably rapid. Future works would be directed at improving the algorithms performance on rotated unimodal problems without adversely affecting its computational complexity.

REFERENCES

[1] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*: MIT press, 1992.

[2] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization,* vol. 11, pp. 341-359, 1997.

[3] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *IEEE International Conference on Neural Networks - Conference Proceedings*, Piscataway, NJ, 1995, pp. 1942-1948.

[4] M. Dorigo, M. Birattari, and T. Stützle, "Ant colony optimization," *Computational Intelligence Magazine, IEEE,* vol. 1, pp. 28-39, 2006.

[5] M. D. Li, H. Zhao, X. W. Weng, and T. Han, "A novel nature-inspired algorithm for optimization: Virus colony search," *Advances in Engineering Software,* vol. 92, pp. 65-88, 2016.

[6] S. Mirjalili, "Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm," *Knowledge-Based Systems,* vol. 89, pp. 228-249, 2015.

[7] H. Salimi, "Stochastic Fractal Search: A powerful metaheuristic algorithm," *Knowledge-Based Systems,* vol. 75, pp. 1-18, 2015.

[8] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey Wolf Optimizer," *Advances in Engineering Software,* vol. 69, pp. 46-61, 2014.

[9] Punnathanam V., Kotecha K., https://sites.google.com/site/yinyangpairoptimization/manuscript, Last accessed on April 12, 2016.

[10] Q. Chen, B. Liu, Q. Zhang, J. Liang, P. Suganthan, and B. Qu, "Problem Definitions and Evaluation Criteria for CEC 2015 Special Session on Bound Constrained Single-Objective Computationally Expensive Numerical Optimization," Technical Report, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Technical Report, Nanyang Technological University, Singapore, 2014.