Using Neural Networks in Diversifying Differential Evolution for Dynamic Optimization

Maryam Hasani-Shoreh, Renato Hermoza Aragonés, and Frank Neumann

University of Adelaide, Australia
firstname.lastname@adelaide.edu.au

Abstract. Dynamic constrained optimization problems occur in a variety of real-world problems. To tackle these problems, evolutionary algorithms have been extensively used due to their effectiveness and minimum design effort. However, to make them prepared for dynamic problems extra mechanisms are required on top of standard evolutionary algorithms. Among them, diversity mechanisms have proven to be effective in handling dynamism, and recently, the use of neural networks have become popular for this purpose. Considering the complexity of using neural networks in the process in comparison with simple diversity mechanisms, we investigate whether they are competitive and the possibility of integrating them to improve the results. For a fair comparison, we need to consider the same time budget for each algorithm. Thus instead of the usual number of fitness evaluations as the measure for the available time between changes, we use wall clock timing.
The results show the extent of improvement in the results when integrating the neural network and diversity mechanisms depends to the type and the frequency of environmental changes. We observe for differential evolution having the proper diversity among population when using neural network plays a key role to the effectiveness of neural network.

Keywords: Dynamic Constrained Optimization · Differential Evolution · Evolutionary Algorithm · Neural Network.

1 Introduction

The dynamism occurs in many real-world problems and it is originated from factors such as variation in the demand market, unpredicted events or variable resources [16, 4]. The goal in these problems is to find the optimum in each instance of the dynamic problem given a limited computational budget. One approach is to apply an independent optimization method to separately solve each problem instance. However, a more efficient approach solves them through an ongoing search, in which the algorithm detects and responds to changes dynamically [21]. Standard evolutionary algorithms (EAs) can be easily modified to handle dynamic environments. The required modifications include change detection and the ability to react to the changes.

Among the many approaches proposed to react to the changes in dynamic environments, diversity mechanisms [6, 10] are the most popular and yet the simplest one. A recent study revealed how common diversity mechanisms can significantly enhance the performance of a baseline differential evolution (DE) for different environmental changes [14]. Other approaches are memory-based approaches [22], multi-population approaches [3] or prediction methods [5].

Previous work on prediction approaches has shown how they can be well suited for dealing with dynamic problems where there is a trend in the environmental changes [18]. For instance, in [23] the Kalman filter is applied to model the movement of the optimum and predict the possible optimum in future environments. Similarly, in [25] linear regression is used to estimate the time of the next change and Markov chains is adopted to predict new optimum based on the previous times optimum. Likewise, in [26] the center points of Pareto sets in past environments are used as data to simulate the change pattern of the center points by using a regression model. Besides these methods, neural

networks (NNs) have gained increasing attention in recent years [17, 18, 15, 19]. In [19], a temporal convolutional network with Monte Carlo dropout is used to predict the next optimum position. In this work the influence of the prediction is controlled via estimation of the prediction uncertainty. In [18] a recurrent NN is proposed that is best suited for objective functions where the optimum movement follows a recurrent pattern. In other works [17, 15], where the change pattern is not stable, it is proposed to directly construct a transfer model of the solutions/fitness using NNs, considering the correlation and difference between the two consecutive environments.

However despite the previous attempts, there are still some concerns regarding the application of NNs in the evolution process. As integrating them in EAs is more complicated than using standard diversity mechanisms, the question arises whether they enhance the results to an extent that pays off for their complexity. In addition, previous work mainly has compared prediction based methods with baseline algorithm and other peer prediction methods[18, 13]. The only work that considers other mechanisms for the dynamic handling is a recent work [19]. In this work, however, the timing for NN has not been accounted. We believe to do a fair comparison with other standard methods, their time consumption needs to be accounted. This is because NN may create a noticeable overhead in the optimization process due to data collection, training and the prediction of new solutions.

To evaluate the effectiveness of NN in the described setting, in this work we compare common diversity mechanisms using a DE algorithm with and without NN through an empirical study. We select DE for our baseline as is a competitive algorithm in constrained and dynamic optimization for continuous spaces [1]. The experiments are repeated for different frequencies of change. In this study, we focus on answering the following questions. Taking into account the time spent on NNs:

- How is it's comparison with other simpler mechanisms for diversifying DE to handle dynamic environments?
- Does diversity of population in DE has influence on the effectiveness of NN?
- Does different frequencies of change impact the suitability of NNs in diversifying DE?

To account for the timing used by NN, we apply the method presented in [13]. In this method, a change happens after an actual running time of the algorithm. This is not the usual way, as in the literature of dynamic problems, a change is often designed to happen after a number of fitness evaluations or generations. But in that way the time spent by NN is not accounted. The results show considering the timing spent for using NN, yet there is significant enhancement in the algorithms when used on top of diversity mechanisms. In addition, for DE in particular, we observe the influence of having a sound diversity among population on the effectiveness of NN. The remainder of the paper is as follows. Section 2 introduces preliminaries on the topic. Our experimental methodology is presented in Section 3. In section 4, a comparison across all the methods are presented. In section 5, we carry out detailed experimental investigations on the use of NNs in different variants of diversifying DE. Finally, we finish with some conclusions and point out some directions for future work.

2 Preliminaries

In this section, an overview of the adapted differential evolution (DE) algorithm to solve dynamic problems, diversity mechanisms and the design of the NN are presented.

2.1 Problem statement

Generally, a dynamic constrained optimization problem (DCOP) is considered as a kind of problem that its fitness function and feasible region will change by

time [21]. Mathematically, a DCOP is defined as follows: Find $\vec{x}$, at each time t, which:

$$\min_{\vec{x} \in F_t \subseteq [L,U]} f(\vec{x}, t) \tag{1}$$

where $t \in N^+$ is the current time,

$$[L,U] = \{\vec{x} = (x_1, x_2, ..., x_D) \mid L_i \leq x_i \leq U_i, \\ i = 1 \dots D\} \tag{2}$$

is the search space, subject to:

$$F_t = \{\vec{x} \mid \vec{x} \in [L,U], g_i(\vec{x}, t) \leq 0, i = 1, \dots, m, \\ h_j(\vec{x}, t) = 0, j = 1, \dots, p\} \tag{3}$$

is called the feasible region at time t.

$\forall \vec{x} \in F_t$ if there exists a solution $\vec{x}^* \in F_t$ such that $f(\vec{x}^*, t) \leq f(\vec{x}, t)$, then $\vec{x}^*$ is called a feasible optimum solution and $f(\vec{x}^*, t)$ is called the feasible optimum value at time t.

2.2 Differential evolution

Differential evolution (DE) is a stochastic search algorithm that is simple, reliable and fast which showed competitive results in constrained and dynamic optimization [1]. Each vector $\vec{x}_{i,G}$ in the current population (called as target vector at the moment of the reproduction) generates one trial vector $\vec{u}_{i,G}$ by using a mutant vector $\vec{v}_{i,G}$. The mutant vector is created applying $\vec{v}_{i,G} = \vec{x}_{r0,G} + F(\vec{x}_{r1,G} - \vec{x}_{r2,G})$, where $\vec{x}_{r0,G}$, $\vec{x}_{r1,G}$, and $\vec{x}_{r2,G}$ are vectors chosen at random from the current population ($r0 \neq r1 \neq r2 \neq i$); $\vec{x}_{r0,G}$ is known as the base vector and $\vec{x}_{r1,G}$, and $\vec{x}_{r2,G}$ are the difference vectors and $F > 0$ is a parameter called scale factor. The trial vector is created by the recombination of the target vector and mutant vector using a crossover probability $CR \in [0, 1]$. In this paper, a simple version of DE called DE/rand/1/bin variant is chosen; where "rand" indicates how the base vector is chosen, "1" represents how many vector pairs will contribute in differential mutation, and "bin" is the type of crossover (binomial in our case). Feasibility rules [8] is applied for the constraint handling.

In addition to constraint handling, the algorithms need a mechanism to detect the changes. In the literature, re-evaluation of the solutions is the most common change-detection approach [21]. The algorithm regularly re-evaluates specific solutions (in this work the first and the middle individual of the population) to detect changes in their function values or/and the constraints. If a change is detected, then the change reaction approach will be activated. In this work, two approaches are considered as base of our algorithms and at top of them diversity mechanisms are considered that will be explained in Section 2.3. In the first approach, called noNN, the whole population is re-evaluated. In the second approach (called as NN), a couple of worst individuals in the population will be replaced with the predicted solutions and the rest of the individuals are re-evaluated.

2.3 Diversity mechanisms

We applied the most common diversity mechanisms. For a recent survey regarding the effect of diversity mechanisms in dynamic constrained optimization see [14]. Following is a brief description of each method. For further detail, we refer to the original papers for each method.

Crowding: Among the many niching methods [1], we choose the standard crowding method [24]. In this method, similar individuals in the population are avoided, creating genotypic diversity [2]. Instead of competition between the offspring and the parent, the offspring competes with its closest individual in

[1] Niching techniques are the extension of standard EAs to multi-modal domains

[2] Diversity can be defined at distinct levels; genotypic level refers to differences among individuals over $\vec{x}$ values

terms of Euclidean distance. As our problem dimension is high and due to the selected crossover rate for DE, often the parent is the closest individual. Thus, we modified the method such that offspring competes with N closest individuals (denoted by CwN).

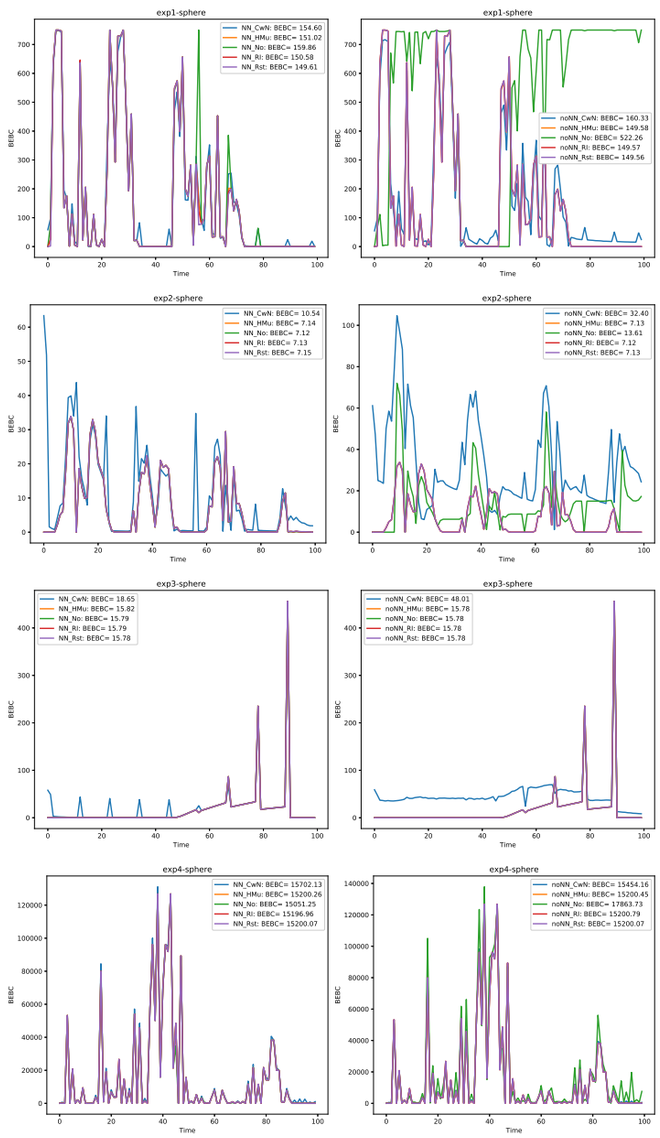

Fig. 1: Best error before change values over time, $\tau = 10$.

Random immigrants: This method (denoted as RI) replaces certain number of individuals (defined by a parameter called replacement rate) with random solutions in the population to assure continuous exploration [11]. In the original paper in every generation random immigrants are inserted in the population. For our version however, as we consider wall clock timing between each change, if we insert solutions at each generation, there is not sufficient time for the evolution process and the results are affected adversely. Thus, random solutions are inserted only when a change is detected.

Restart population: In this method (denoted by Rst), the population is re-started by random individuals. This is an extreme case of RI by considering the replacement rate to the population size.

Hyper-mutation: This method was first proposed by using an adaptive mutation operator in genetic algorithms to solve dynamic constrained opti-

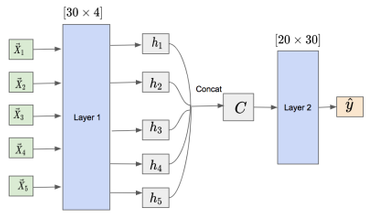

Fig. 2: Structure of neural network

mization problems [7]. Later, it was used for DE in [2]. After a change detection, some of the DE parameters (CR and F) change for a number of generations defined empirically (dependent on frequencies of change) to favor larger movements. However for DE, other mechanisms are needed, since when is converged, is not able to promote diversity. Indeed, DE is dependant to the population diversity to enhance diversity (see mutant vector Equation in Section 2.2). For our version denoted by HMu, in addition to changes of the DE parameters, we insert a couple of random individuals to the population.

2.4 Structure of the neural network

Neural network (NN) is used to predict the future optimum position. To do so, the best solutions of the previous change periods achieved by DE are required to build a time series. Using them NN will go through a training process to learn the change pattern of the optimum position. Among many structures proposed for NN, we use a simple multi-layer feed forward NN in this work. We consider a procedure proposed in [13] for sample collection. In this work to expedite sample collection, it is proposed to use k-best individuals of each time for a couple of previous times ($n_t = 5$). Then a combination of all possibilities (k^{n_t}) to build training data is considered. However, the number of samples collected are limited by choosing a random subset of the aforementioned combination. As otherwise, the time spent for training data exponentially increases due to the large number of collected samples. Also, as in this way enough number of samples are collected, the NN can be limited to use the samples from n_w previous changes only. In this way, the focus is in collecting data from recent previous changes that may have more similar pattern.

As for the first environment changes a small number of samples are existed, hence, it is difficult for the NN to generalize from these data. To avoid this, a lag is defined until a minimum amount of samples are collected (defined as min_batch size, and empirically is assigned to 20).

A brief introduction to the architecture of NN is as follows. For more detail we refer to [13]. We applied a simple version of NN structure that has two hidden layers. The structure is presented in Figure 2. The first layer takes an individual position $\vec{x_i}$ with d dimensions as an input and outputs a hidden representation h_i of the individual with 4 dimensions. As the network uses the last 5 times best individuals to predict a next one, the first layer is applied to each of these 5 individuals $\vec{x_1}, ..., \vec{x_5}$ independently. As a result, we obtain 5 hidden representation with 4 dimensions $h_1, ..., h_5$; to aggregate their information, we choose to concatenate them into a variable H with 4×5 dimensions. The second layer takes H as input and then outputs a prediction with d dimensions, representing the next best individual. The layer one has rectified linear units (ReLU) activation function and the second layer has a linear output without activation function. To train the network, we use mean squared error as a loss function. A couple of neighboring positions of the predicted solution (created by adding noise to the original predicted solution) then are replaced by worst individuals of the population in DE to intensify the search in that region of the solution space.

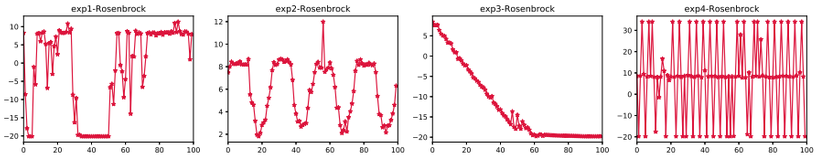

Fig. 3: PCA plot of best_known positions for each experiment over time

3 Experimental Methodology

In this section, the test problems, the algorithms' parameter settings, and the performance metrics are summarized.

3.1 Test problems and parameter settings

To test our algorithms we designed the environmental changes as follows. Dynamic environments are created in two general cases for common functions in literature (sphere (uni-modal), rosenbrock (non-separable) and rastrigin (multi-modal)). In the first two experiments, objective function is constant while the constraints change, and for the last two, the problem is unconstrained with dynamic objective function. Details of the dynamism in each experiment are presented in Table 1. In the first two experiments, the changes are targeted on b values (constraint boundary) of one linear constraint in the the form of $a_i x_i \leq b$ [12]. In the last two experiments the optimum position is transformed based on specific patterns. Figure 3 shows the pattern in which the position

Table 1: Designed test problems

exp1	Uniformly random changes on the boundaries of one linear constraint	$b[t+1] = b[t] + \mathcal{U}(lk, uk)$
exp2	Patterned sinusoidal changes on the boundaries of one linear constraint	$b[t+1] = p \cdot sin(b[t]) + \mathcal{N}(0.5)$
exp3	Linear transformation of the optimum position	$\vec{X_{t+1}} = \vec{X_t} + 0.1t$
exp4	Transformation of the optimum position in sinusoidal pattern with random amplitudes	$\vec{X_{t+1}} = \vec{X_t} + p[t]sin(\frac{\pi}{2}t)$

of optimum changes for rosenbrock fucntion in each experiment [3]. To achieve this plot, the principal component analysis (PCA) method is used to map the thirty dimension to one dimension scale.

The frequency of change, denoted by τ, represents the width that each time lasts. We indicate that when referring to higher frequencies of change, we mean lower values for τ, since higher frequencies of change happens when there is shorter time interval between each change (τ). Different frequencies of change: 1, 5, 10, 20, and 30 will be tested. As mentioned, in this work wall clock time is considered, so the values above represent time in seconds between the two consecutive changes. To have an idea, these values represent the following number of fitness evaluations for the baseline algorithm: $1 \approx 2000$, $5 \approx 11000$, $10 \approx 22000$, $20 \approx 45000$. Undoubtedly, these numbers are not constant for all test cases due to different time-complexity of each function and stochastic nature of DE.

The other parameters are: $d = 30$, runs=20 and the number of changes or times=100. Parameters of DE are chosen as $NP = 20$, $CR = 0.3$, F is a random number in $[0.2, 0.8]$, and rand/1/bin is the chosen variant of DE [1]. The solution space is within $[-5, 5]^d$. Parameters of NN has been selected in a set of preliminary experiments [13]: $k = 3$: epochs=4, $n_w = 5$, batch_size=4, min_batch=20 and $n_p = 5$. All the experiments were run on a cluster, allocating 1 core (2.4GHz) and 4GB of RAM. Our code is publicly available on GitHub: $https://github.com/renato145/DENN$.

[3] The results belong to best_known solutions of each time retrieved by executing 100,000 runs of a baseline DE algorithm.

3.2 Metrics

We applied common performance metrics in the literature of DCOPs as follows:

Modified offline error (MOF) represents the average of the sum of errors in each generation divided by the total generations [20].

$$MOF = \frac{1}{G_{max}} \sum_{G=1}^{G_{max}} (|f(\vec{x}^*, t) - f(\vec{x}_{best,G}, t)|) \tag{4}$$

Where G_{max} is the maximum generation, $f(\vec{x}^*, t)$ is the global optimum at current time t, and $f(\vec{x}_{best,G}, t)$ represents the best solution found so far at generation G at current time t. Only feasible solutions are considered to calculate the best errors at every generation. If there were no feasible solution at a particular generation, the worst possible value that a feasible particle can have would be taken.

Best error before change (BEBC) is another common measure that considers the behaviour of algorithm only in the last solution achieved before next change happens.

$$BEBC = \frac{1}{T_{max}} \sum_{t=1}^{T_{max}} (|f(\vec{x}^*, t) - f(\vec{x}_{best}, t)|) \tag{5}$$

Absolute recovery rate introduced in [20] is used to analyze the convergence behaviour of the algorithms in dynamic environments. This measure infers how quick an algorithm starts converging to the global optimum before the next change occurs. An important observation about this metric is that it reports the speed of convergence relatively to the first achieved solution [18].

$$ARR = \frac{1}{T_{max}} \sum_{t=1}^{T_{max}} (\frac{\sum_{G=1}^{G_{max}(t)} |f_{best}(t, G) - f_{best}(t, 1)|}{G_{max}(t)[f^*(t) - f_{best}(t, 1)]}) \tag{6}$$

Success rate (SR) calculates in how many times (over all times) each algorithm is successful to reach to ϵ-precision from the optimum before reaching to the next change.

NN-Error reports the Euclidean distance of the predicted solution and the best_known. Lower values are preferred.

NN-time reports the percentage of the time spent to train and use NN per overall optimization time.

Table 2: Pairwise comparison of methods on MOF values for $\tau = 1$ and 20 (mean of 20 runs)

Experiment	Function	noNN_RI	NN_RI	noNN_HMu	NN_HMu	noNN_No	NN_No	noNN_CwN	NN_CwN	noNN_Rst	NN_Rst
$\tau = 1$											
1	Rastrigin	**100.59**	103.36	**105.7**	113.28	517.9	**489.26**	**222.97**	376.18	144.39	**140.48**
	Rosenbrock	**73601.01**	84413.69	**64690.02**	68315.59	1958799.29	**73601.56**	624341.74	**315775.79**	**188898.12**	206130.53
	Sphere	**21.15**	24.38	**20.45**	21.5	546.55	**25.67**	173.99	**96.84**	46.46	**41.03**
2	Rastrigin	84.23	**59.59**	96.73	**82.08**	**30.42**	30.76	196.99	**74.28**	157.8	**147**
	Rosenbrock	3495.21	**2386.98**	8014.1	**3814.8**	2143.16	**619.47**	73201.31	**3838.93**	17171.11	**10804.96**
	Sphere	4.25	**3.12**	8.31	**4.86**	9.98	**1.29**	68.38	**5.59**	16.55	**10.7**
3	Rastrigin	**21.06**	33.74	**24.87**	42.19	472.29	**318.94**	**247.64**	333.28	**30.98**	34.38
	Rosenbrock	**109.23**	146.53	**122.73**	159.75	1787.22	**128.55**	699754.07	**1722.27**	399.02	**307.96**
	Sphere	**0.09**	0.11	**0.09**	0.11	**0.08**	0.1	133.52	**2.03**	0.36	**0.31**
4	Rastrigin	**841.9**	871.99	772.83	**688.72**	3832	**1139.29**	**1748.45**	2848.3	754.38	**716.17**
	Rosenbrock	223395465.4	**151705461.9**	189535874	**108888429.1**	957940673.4	**205570867.9**	**477280317**	945221760	175355577.9	**127501241**
	Sphere	733.49	**609.08**	625.98	**428.99**	3541.08	**780.85**	**1431.8**	2833.45	549.18	**492.19**
$\tau = 20$											
1	Rastrigin	34.84	**33.88**	**40.16**	46.77	516.94	**483.62**	**159.09**	263.95	**46.86**	47.12
	Rosenbrock	**9210.76**	11159.41	11588.09	**10445.52**	1988246.32	**12044.2**	567717.09	**183479.33**	**19508.38**	21891.08
	Sphere	**2.86**	3.44	3.56	**3.26**	545.61	**3.46**	143.29	**43.39**	**5.4**	5.97
2	Rastrigin	20.78	**19.82**	25.47	**23.21**	42.58	**20.83**	121.64	**21.71**	46.56	**40.38**
	Rosenbrock	445.51	**290.08**	1555.23	**624.17**	105.43	**88.32**	24211.28	**1307.21**	2762.26	**1549.64**
	Sphere	0.51	**0.41**	1.38	**0.77**	12.87	**0.19**	35.24	**2.38**	2.17	**1.41**
3	Rastrigin	**11.54**	51.33	**31.99**	118.2	874.81	**802.81**	**86.23**	430.54	10.41	**8.69**
	Rosenbrock	22.3	**16.06**	21.73	**17.1**	74.6	**17.49**	204135.11	**274.62**	**11.34**	16.79
	Sphere	0.04	**0.03**	**0.03**	0.07	0.05	**0.03**	35.14	**0.52**	**0.02**	**0.02**
4	Rastrigin	129.34	**118.55**	131.28	**107.88**	3930.78	**320.03**	**717.04**	1472.18	111.38	**103.62**
	Rosenbrock	23445061.73	**16785682.65**	20298120.95	**12073567.99**	1322886269	**34616064.68**	**152205349.1**	391706791.9	16297615.87	**8864684.95**
	Sphere	74.36	**57.94**	68.51	**56.91**	4256.18	**129.66**	**588.71**	1209.84	53.91	**42.82**

4 Cross Comparison of Approaches

Figure 8 presents a boxplot for MOF values distribution based of each frequency for 20 runs. For this plot, we omitted worse performing algorithms to

have a better resolution for analyzing other methods. The results emphasize how with increasing τ from 1 to 10 the MOF values decrease. As the algorithms have more timing budget within each change to evolve the solutions and achieve closer values to optimum. The biggest difference is for $\tau = 1$ in comparison to other τ values. However, there are some exceptions from this general trend. For instance the behaviour of NN_No in exp3 for rastrigin function is an anomaly. According to Figure 8, when increasing τ from 1 to 20, the MOF values also increase. Looking to the plot for optimum position changes in Figure 3, there is a linearly decreasing trend in the first half of the time scale and a constant optimum position in the second half. As NN does not have the correct new optimum (based of specific characteristic of this experiment), then it is not helpful to DE. In addition, with rastrigin function there are chances of getting stuck in local optimum, as it has a multimodal attribute. This intensifies in higher τ values as the population is more converged. However, the algorithms with diversity variants could avoid the local optimum by promoting diversity. This shows the importance of diversity variants in case of wrong predictions. NN relies in the previous time solutions achieved by DE, if the solutions are far from the optimum, the resulting training data will have poor quality and therefore not useful to DE anymore. Furthermore, the poor performance of NN_No compared to its diversity variants counterparts in exp1 and exp3 for rastrigin function, shows how we can improve the results of NN by using diversity mechanisms properly. In addition, Table 2 show the MOF values allowing pairwise comparison of diversity variants with and without NN for $\tau = 1$ and $\tau = 20$ respectively. In each set of columns the better performing algorithm is highlighted. We clearly see for small τ, methods with NN are not competitive with their counterparts in each set of diversity variant. Albeit, for large τ value the trend changes and NN variants outperform. First, as for this NN architecture, we train the NN with three best solutions of each time, thus when τ is smaller, the algorithm is not converged and the best solutions have greater distance between each other and may not represent the optimum region properly. Furthermore, in higher τ values, the NN time expenditure is negligible compared to the whole evolution process (as we see later in Table 3). In addition, NN gives direction to the diversity mechanisms that have more random nature. Therefore integration of NN and diversity variant improve the algorithm in comparison to its baseline diversity variant.

For overall comparison of the algorithms a heatmap with mean rankings of MOF values is presented in Figure 4. To achieve these grades, first we rank every method grouped by function, experiment and frequency. Afterwards, we calculate the mean of the ranks among all the frequencies. This figure helps to analyze the performance of the algorithms over each set of experiment and function compared to each other. For instance, an evident observation is that methods using CwN are not competent in most functions and experiments.

However, this heatmap is not able to define the severity of differences among methods. To the purpose of relativity analysis, we propose Figure 7 that presents the results of the overall performance of methods in each experiment. To achieve standard values (denoted as MOF_norm) in each set of function and experiment, the values are divided by the minimum value among all methods. So the method with lowest MOF value has MOF_norm value assigned to one and the others are proportionally calculated. As an example, in this figure, we can observe CwN and Rst, in experiment 2 and 3, are considerably worse than others. To achieve a better resolution in methods' comparison, we limit the y-scale, at the cost of missing some data from the worse performing algorithms. Other detectable observation from this figure, is that variants of RI and HMu has better performances overall.

To validate the results of MOF values, the 95%-confidence Kruskal-Wallis statistical test and the Bonferroni post hoc test, as suggested in [9] are presented (see Figure 9). Nonparametric tests were adopted because the samples of runs did not fit a normal distribution based on the Kolmogorov-Smirnov test. In this heatmap, as the legend represents, the squares with brightest color show the methods with not-significantly different (NS) results, and the squares in the spectrum of the purple colors show the significantly different

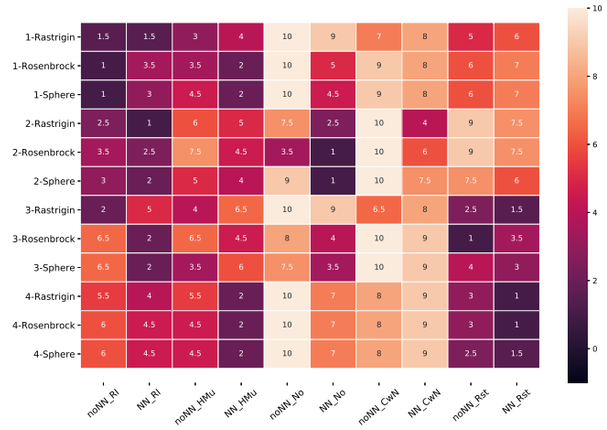

Fig. 4: Heatmap of methods' rank over MOF values for each function and experiment, considering $\tau = 5$, and 10. Lower ranks represent better performances. Numbers in Y-axis label show experiment number.

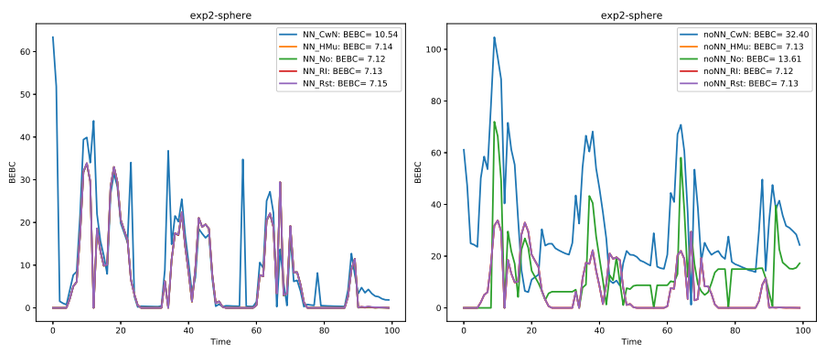

Fig. 5: Best error before change values over time, $\tau = 10$.

methods with the mentioned p-values. Results of the test display, in most test cases, the methods have significant difference among each other.

To compare the algorithms in each separate time look into Figure 5. This figure shows the deviation of the best solution achieved at each time and the best_known solution for that time. Thus it shows how the methods track the best_known along with the changes. The values in legend show the BEBC values for each method, that is the average of what the plots show over all times. Lower values show the methods are capable of closely following the optimum like NN_RI. Overall, most of the methods show similar results only variants of CwN and noNN_No are inferior. We only bring the results of sphere function for $\tau = 10$, but considering other functions and frequencies, overall, when τ is small, this measure show more differences between methods. But for $\tau = 10$ most of the methods could achieve near to optimum solutions no matter the differences among their evolution process. In addition, for rastrigin with multimodal characteristic the differences between methods for this measure are bigger as there are more chances of algorithms with lack of diversity follow a local optimum and become unable to reach near global optimum solutions.

The other figure (Figure 6) show the Euclidean distance of the position of best at first generation after a change and the position of best_known. This plot gives an idea of how far from the best_known position the methods start for the new time. We observe Rst is more far away as we expected since it starts randomly. In this experiment as the position changes are not huge (see Table 1), so noNN_No also is not so far away the best_known. Albeit yet the results for this method for MOF is not satisfying (Table 2) and it is due to lack of a diverse population to allow proper exploration in the search space for new optimum.

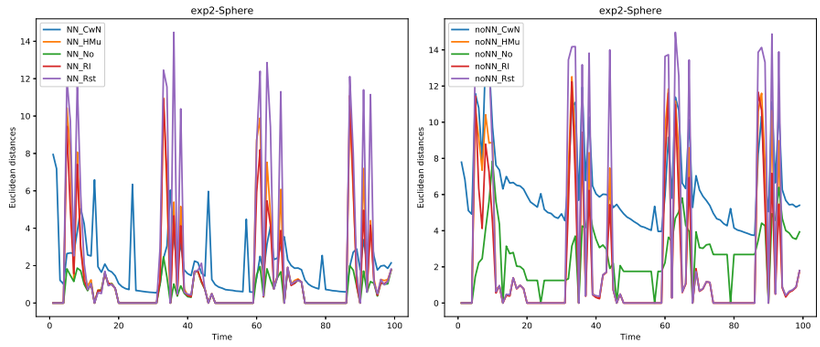

Fig. 6: Euclidean distance between the best of first generation after change and the optimum position for each method, $\tau = 10$.

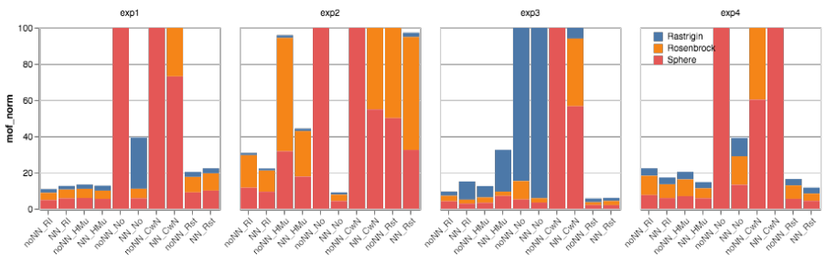

Fig. 7: MOF-norm values considering $\tau = 10$ for each method and experiment, color-coded with functions

In Figure 12, the prediction error, is plotted for each time. To achieve prediction error, as three solutions are sampled around the predicted solution by NN, so we considered the average Euclidean distance of the best_known solution to these three inserted solutions. Notice that NN starts to predict after collecting enough data, so the time scale for this plot is shorter as opposed to 100 times for other plots.

Figure 10 shows a heatmap of the algorithms' ranking based on ARR values for every method considering all frequencies of change. This measure represents which method can recover faster after a change. ARR rank results show the noNN version of Rst, RI and HMu variants have almost the best recovery from their first best solution after a change, compared to other methods in all experiments. As we can interpret from Equation 6, this measure is slightly biased over the first solution achieved. In consequence, according to this measure, algorithms that start with a very poor solution may achieve a higher ARR values than those starting with a better solution. So if the first best solution is drastically changed for next generations, this measure reports better results. This is the reason noNN_Rst is the best based on this measure.

Conversely, the worst results for this measure is for CwN variants (both NN and noNN) and noNN_No. Comparing NN_No and noNN_No, We can conclude how NN improves recovery capabilities of the algorithm, especially for exp1 and exp4 with drastic changes.

The heatmap for SR values (see Figure 11) illustrates satisfying results for almost all the methods; meaning they can reach to an ϵ-precision (=10%) of the optimum for almost all the changes (or times). However, CwN-variants and noNN_No are the exceptions in which SR values are low. In addition, all the methods show difficulty reaching to optimum in exp2 for rastrigin function (low values for SR). Moreover, in other experiments all the methods decrease their performance based of SR values for this function compared to the other two functions. This is attributed by its multimodal characteristic.

Table 3 represents the percentages of the amount of time spent for calling NN unit compared to the overall optimization time. Regardless of the experiment and function, the results show for $\tau = 1$ around 10-11%, $\tau = 5$ around 2%, $\tau = 10$ around 1% and $\tau = 20$ around 0.5%. This shows when τ is higher,

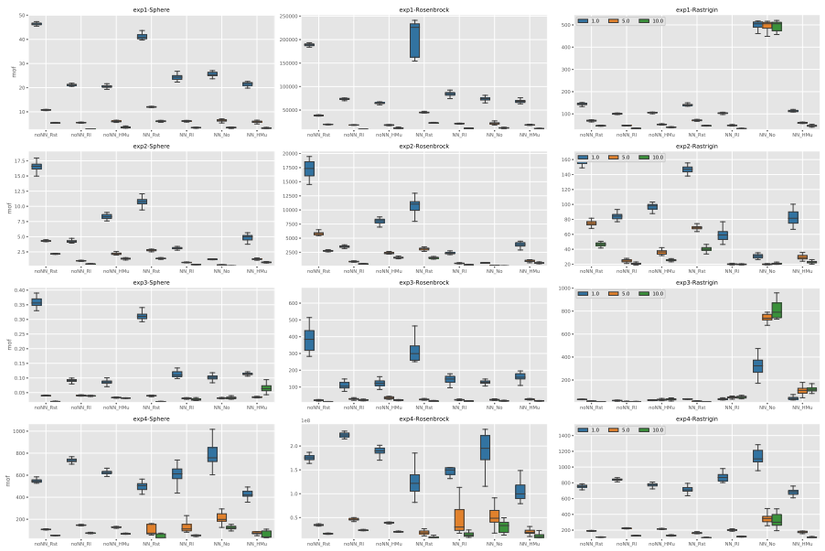

Fig. 8: Distribution of MOF values for 20 runs, different frequencies (τ). Lower values represent better performances

it is less expensive to use NN in terms of the computational cost. As NN time remains constant, when τ is small, the proportion of time for evolution process is lower. Hence, the samples used to train NN do not represent real optimum or near optimum values and the prediction from NN is not exact in consequence.

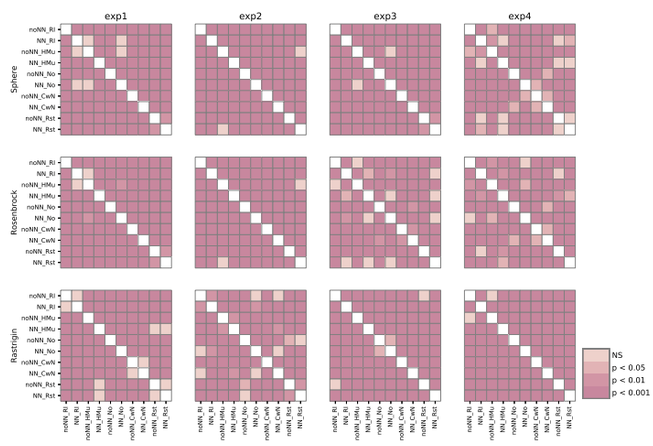

Fig. 9: Kruskal-Wallis statistical test on MOF values for τ=10, NS represents not-significant

5 Detailed Examination of Using Neural Networks

In this section, the methods are compared on the basis of each diversity variant.

5.1 Crowding

From Figure 1, we observe that CwN variants have the biggest difference to the optimum compared to other methods. This explains their poor performance in MOF values looking to the heatmap (see Figure 4). Based of the colors of the heatmap, it is easily noticeable that the variants of this method perform dissatisfying. NN can enhance the results for this method, but still remains inferior compared to other methods. This method behaves particularly poor in exp2

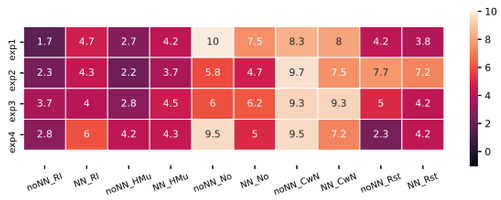

Fig. 10: Heatmap of methods' rank over absolute recovery rate (ARR) values for each experiment, considering $\tau = 5$ and 10. Lower ranks represent better performances

and exp3, even compared to noNN_No. By promoting diversity unnecessarily, CwN adversely affects the convergence of the algorithm to new optimum position, which is not too distant from the previous optimum. In addition, from Figure 10 we can see this variant delays the recovery of the algorithm (low rank for ARR values in NN_CwN and noNN_CwN show inability of the algorithm in recovery after a change).

CwN has been reported as one of the best methods for handling dynamic environments in a recent survey study considering other diversity methods [14]. The reason that CwN is not competitive in our study is due to the following reasons: firstly, in the literature this method showed its best effectiveness for multimodal test problems in which we have several local optima. In such case, CwN helps to diversify solutions by avoiding similar individuals in each sub region of the search space [24]. In addition, CwN showed its thriving performance in problems that include features like disconnected feasible regions and small feasible areas [14]. Secondly, CwN was tested on a benchmark with a low problem dimension (2) on the previous work [14]. As our problem's dimension is 30, and having a crossover rate (CR) of 0.3 leads the offspring to change only in a couple of dimensions. In this condition the closest individual will be the parent, causing the method to act similar to no-diversity mechanism but with an overhead of calculating distances at each iteration. To alleviate this in our CwN version, the offspring will compete with the N=5 closest individuals.

Checking the results of SR-values for noNN_CwN shows this algorithm is barely able to get to the optimum for almost all the changes in exp2 for all functions and exp3 for rosenbrock function. While as we see in Figure 11 , most of the other methods are in 100% for exp2 (rosenbrock and sphere).

Fig. 11: Heatmap of mean values (20 runs) for success rate (SR), considering $\tau = 5$ and 10. Higher values represent better performances. Numbers in Y-axis show experiments

5.2 Random immigrants and restart population

From Figure 1, it is visible that in most functions and experiments, the solutions achieved by NN_RI are the best among others in terms of tracking the optimum. This can explain its best rank amongst all methods, represented in Figure 4. From this figure, we can see its noNN version also has a promising

ranking. To see more clearly, Table 2 show the comparison of NN and noNN version for this method. Comparing them, the results for large τ for NN show better performances than noNN in most of the functions and experiments except for rastrigin with a multimodal characteristic in exp3 and rosenbrock and sphere in experiment 1 with random changes. The reason for the better performance of NN_RI is that in noNN_RI we only insert random solutions, while for NN_RI, we diversify solutions by random immigrants and direct them with the predicted solutions by NN which expedite the convergence to new optimum leading to better MOF values. However, for small τ NN is less able to beat its noNN counterpart. Low timing budget leave the algorithm with poor final solutions at each time. Thus NN is fed with poor quality solutions and is not able to predict the correct future times optimum positions. Although worth to mention, RI is reported to have lower performance results in cases of small feasible areas [14]. The reason is the inserted solutions are discarded by constraint handling mechanism and can not proceed as best solution to guide the search for next generations. In our test problem, we lack such a small feasible area in which RI may show its worst performance. For future work, we will test this method for smaller feasible areas with features of disconnected feasible areas. In addition, another possible future work is to observe the effect of randomness over predicted solutions. In our experiments, we test with inserting 5 predicted solutions and 2 random solutions. However, different combinations of these forces can be experimented based of problem features.

As mentioned before, Rst can be considered as a sever case of RI in which we discard all the previous attempts of the algorithm and start new random solutions. This implies when using NN, diversity of population is of high importance. In case of NN_Rst we have a population scattered around the search space. This is not helpful when NN tries to direct the solution toward the new optimum. In case of NN_RI, however, we have a proper amount of diversity among solutions (5 individuals from NN and 2 individuals from RI), so the results are promising.

In most of the experiments, we can clearly observe the behavior of RI is significantly better than this method. However, noNN_Rst is ranked as the best for exp4 in which the position of the optimum have drastic changes. Conversely, noNN_Rst achieve a low ranking for exp2 and exp3 in which the optimum position changes are not huge and by discarding the previous attempts of the algorithm, the performance degrades. This explains why noNN_No has an inferior performance for exp4 in which for larger changes it can not promote diversity to reach optimum. While for exp2 and exp3 is ranked better since it does not need a drastic change in the position.

ARR rank results show noNN_Rst and RI variants have almost the best recovery after a change compared to other methods for all experiments. Since this measure (see Equation 6) reports better values for those algorithms that can get distant from their first generation best solution faster. For these two methods, the good result for this measure was predictable as their first best solution is randomly created and often more far away than the optimum compared to other methods.

According to SR measure, noNN_Rst show medium results. Although the performance of this algorithm drops based of MOF values for exp2 and exp3, but the results for SR values show only drop of performance in exp2 for rastrigin. This discrepancy is due to the fact that MOF values consider the performance of algorithms over all generations while SR cares only about last solution achieved by each algorithm.

5.3 Hyper-mutation

HMu's results are quite similar to RI variant with slightly better performances for exp1 and exp4 and worse performances for exp2 and exp3. The rankings in the heatmap 4, clarify this observation. The reason lies on the shape of the changes in the environment for different experiments. Exp1 and exp4 have bigger changes, in which HMu with larger scale factor after a change converge faster to the new optimum. While for exp2 and exp3 RI is more efficiently handle the smaller change. Since considering hyper mutation factor in HMu, many

Table 3: NN-time; % time spent for training and using NN in proportion to overall optimization time (mean for 20 runs)

experiment		exp1				exp2				exp3				exp4			
τ		1.0	5.0	10.0	20.0	1.0	5.0	10.0	20.0	1.0	5.0	10.0	20.0	1.0	5.0	10.0	20.0
Rastrigin	NN_CwN	10.763	2.154	1.124	0.554	10.941	2.144	1.129	0.551	11.046	2.182	1.191	0.555	10.611	2.103	1.079	0.728
	NN_HMu	11.106	2.227	1.117	0.579	10.928	2.224	1.150	0.589	11.081	2.272	1.149	0.757	10.608	2.144	1.074	0.540
	NN_No	10.755	2.136	1.085	0.548	10.951	2.190	1.097	0.564	11.009	2.190	1.090	0.561	10.628	2.139	1.056	0.553
	NN_RI	11.070	2.163	1.117	0.555	10.896	2.189	1.117	0.564	11.174	2.225	1.124	0.550	10.517	2.124	1.076	0.549
	NN_Rst	10.970	2.197	1.081	0.544	10.817	2.166	1.073	0.544	11.644	2.323	1.137	0.556	10.719	2.147	1.072	0.539
Rosenbrock	NN_CwN	10.658	2.122	1.079	0.539	10.772	2.167	1.140	0.543	10.859	2.193	1.194	0.555	10.803	2.113	1.082	0.529
	NN_HMu	10.826	2.197	1.106	0.558	10.878	2.216	1.163	0.572	10.839	2.194	1.108	0.572	10.674	2.152	1.107	0.541
	NN_No	10.862	2.206	1.091	0.545	10.823	2.173	1.101	0.547	10.754	2.175	1.092	0.569	10.554	2.163	1.108	0.547
	NN_RI	10.921	2.173	1.087	0.561	10.982	2.186	1.104	0.558	11.087	2.166	1.082	0.547	10.598	2.233	1.102	0.545
	NN_Rst	11.020	2.182	1.095	0.550	10.896	2.208	1.085	0.558	11.429	2.248	1.090	0.548	11.575	2.197	1.076	0.547
Sphere	NN_CwN	10.757	2.136	1.067	0.535	10.827	2.164	1.171	0.540	10.875	2.210	1.182	0.545	10.519	2.100	1.139	0.530
	NN_HMu	10.814	2.220	1.100	0.559	10.752	2.183	1.139	0.576	10.717	2.162	1.125	0.572	10.610	2.209	1.181	0.557
	NN_No	10.952	2.211	1.128	0.560	10.830	2.988	1.098	0.539	10.682	2.170	1.088	0.552	10.643	2.152	1.074	0.540
	NN_RI	10.981	2.179	1.111	0.562	11.229	2.186	1.504	0.548	10.903	2.136	1.082	0.544	10.653	2.188	1.101	0.550
	NN_Rst	10.932	2.213	1.110	0.555	11.031	2.188	1.082	0.544	10.988	2.213	1.075	0.545	11.523	2.318	1.091	0.546

of the individuals go through a change and convergence is delayed. Whilst, in these two experiments the optimum position changes are minor. Due to the same reason, NN_No also outperforms NN_HMu for exp2 and exp3. This means diversity mechanisms do not always improve algorithm performance when used on top of NN. In the proposed version of HMu for DE in [2], it is proposed to change DE variant from DE/rand/1/bin to DE/best/1/bin (see Section 2.2), when hyper parameters of DE are activated. The best is chosen from the current time or a memory including previous times best. The test problem in that work has a problem's dimension size of two. For larger problem sizes like ours, this method is not able to promote diversity, since in a DE promoting diversity needs a minimum level of diversity among solutions. Thus, we insert randomly created solutions (7 individuals for the case without NN and 2 with the case using NN). However, the proper selection of the number of inserted individuals can be experimented in a future study.

Based on the statistical test results, this method does not have significant difference in some cases with RI (for exp1 sphere and rosenbrock with RI) and in some cases with Rst (exp1 rastrigin and exp2 sphere with Rst).

6 Conclusions and future directions

In this work, a neural network was considered to solve dynamic problems together with differential evolution. Given the complexity of integrating neural network into the evolution process and considering the time spent to train it, we investigated weather it can be competitive compared to standard diversity mechanisms. We empirically studied the possibility of integrating them to extract the best of each to improve the results. We observed diversity of population is essential when using neural network for DE. DE evolution process depends on the diversity of population; if we use only neural network, as the inserted solutions are distributed around the predicted value, then without other diversity mechanism is slower to explore the search space leading to lower MOF values. In addition, in some cases due to multi-modality of the function, neural network may stuck in local optimum and not having correct data to feed the network will result in amplifying the false effect of prediction in evolution process for future times. In this case diversity mechanisms can help. On the other hand, we observed how neural network can improve the results of simple diversity mechanisms especially for low environmental changes. It can direct the search toward the next optimum besides random nature of diversity mechanisms.

The presented results in this work, belonged to one simple feed-forward neural network. For future work, we encourage application of other designs in the context of dynamic optimization. Considering several proposed structures of neural networks in the machine learning community, their application to handle dynamic optimization problems is in its infancy yet. A through study can show the effect of applying a combination of different neural network structures based of a range of problem types and decides which structure is preferred considering features of the problem.

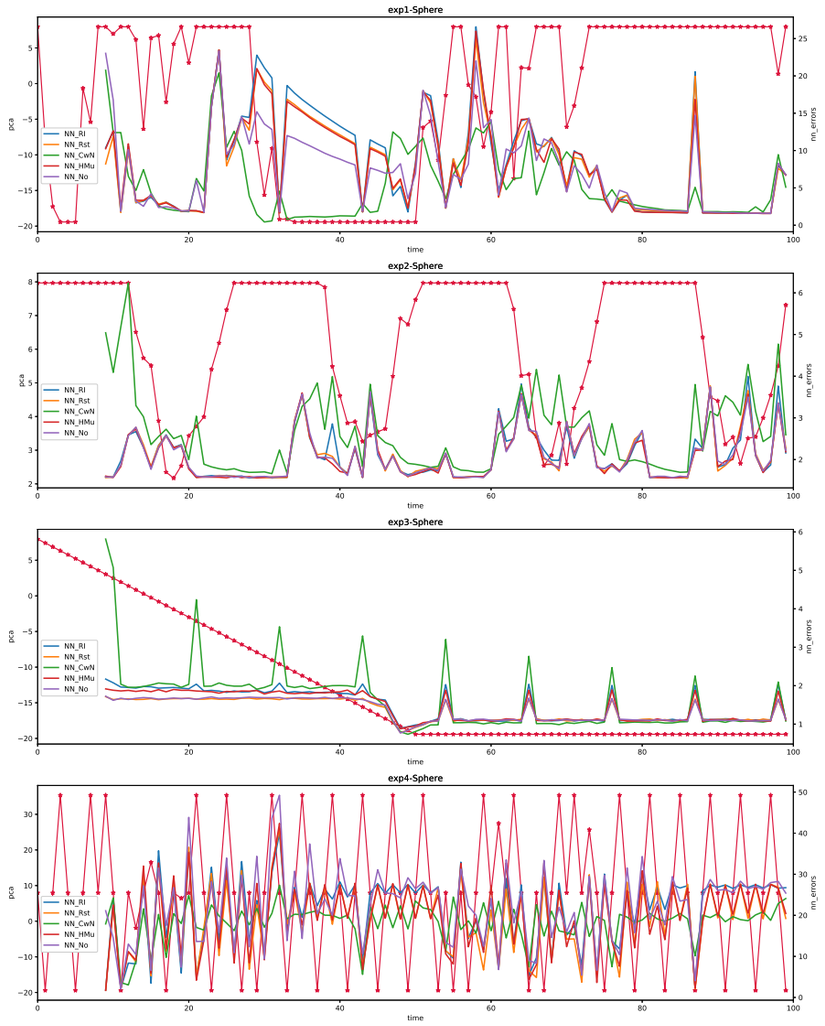

Fig. 12: Error of NN plus PCA for τ=10, right y-axis shows nn_errors and left y-axis is for pca scale

References

1. Ameca-Alducin, M.Y., Hasani-Shoreh, M., Blaikie, W., Neumann, F., Mezura-Montes, E.: A comparison of constraint handling techniques for dynamic constrained optimization problems. In: 2018 IEEE Congress on Evolutionary Computation (CEC). pp. 1–8. IEEE (2018)
2. Ameca-Alducin, M.Y., Mezura-Montes, E., Cruz-Ramirez, N.: Differential evolution with combined variants for dynamic constrained optimization. In: Evolutionary computation (CEC), 2014 IEEE congress on. pp. 975–982. IEEE (2014). https://doi.org/10.1109/CEC.2014.6900629
3. Branke, J., Kaußler, T., Smidt, C., Schmeck, H.: A multi-population approach to dynamic optimization problems. In: Evolutionary design and manufacture, pp. 299–307. Springer (2000)
4. Branke, J., Schmeck, H.: Designing evolutionary algorithms for dynamic optimization problems. In: Advances in evolutionary computing, pp. 239–262. Springer (2003)
5. Bu, C., Luo, W., Yue, L.: Continuous dynamic constrained optimization with ensemble of locating and tracking feasible regions strategies. IEEE Transactions on Evolutionary Computation **PP**(99), 1–1 (2016). https://doi.org/10.1109/TEVC.2016.2567644
6. Bui, L.T., Abbass, H.A., Branke, J.: Multiobjective optimization for dynamic environments. In: 2005 IEEE Congress on Evolutionary Computation. vol. 3, pp. 2349–2356 Vol. 3 (Sept 2005). https://doi.org/10.1109/CEC.2005.1554987
7. Cobb, H.G.: An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments (1990)
8. Deb, K.: An efficient constraint handling method for genetic algorithms. Computer methods in applied mechanics and engineering **186**(2), 311–338 (2000)
9. Derrac, J., García, S., Molina, D., Herrera, F.: A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm and Evolutionary Computation **1**(1), 3–18 (2011). https://doi.org/http://dx.doi.org/10.1016/j.swevo.2011.02.002
10. Goh, C.K., Tan, K.C.: A competitive-cooperative coevolutionary paradigm for dynamic multiobjective optimization. IEEE Transactions on Evolutionary Computation **13**(1), 103–127 (Feb 2009). https://doi.org/10.1109/TEVC.2008.920671
11. Grefenstette, J.J., et al.: Genetic algorithms for changing environments. In: PPSN. vol. 2, pp. 137–144 (1992)
12. Hasani-Shoreh, M., Ameca-Alducin, M.Y., Blaikie, W., Schoenauer, M.: On the behaviour of differential evolution for problems with dynamic linear constraints. In: 2019 IEEE Congress on Evolutionary Computation (CEC). pp. 3045–3052. IEEE (2019)
13. Hasani-Shoreh, M., Aragonés, R.H., Neumann, F.: Neural networks in evolutionary dynamic constrained optimization: Computational cost and benefits (2020)
14. Hasani-Shoreh, M., Neumann, F.: On the use of diversity mechanisms in dynamic constrained continuous optimization. In: International Conference on Neural Information Processing. pp. 644–657. Springer (2019)
15. Jiang, M., Huang, Z., Qiu, L., Huang, W., Yen, G.G.: Transfer learning-based dynamic multiobjective optimization algorithms. IEEE Transactions on Evolutionary Computation **22**(4), 501–514 (2017)
16. Liu, L., Zechman, E.M., Brill, Jr, E.D., Mahinthakumar, G., Ranjithan, S., Uber, J.: Adaptive contamination source identification in water distribution systems using an evolutionary algorithm-based dynamic optimization procedure. In: Water Distribution Systems Analysis Symposium 2006. pp. 1–9 (2008)
17. Liu, X.F., Zhan, Z.H., Gu, T.L., Kwong, S., Lu, Z., Duh, H.B.L., Zhang, J.: Neural network-based information transfer for dynamic optimization. IEEE transactions on neural networks and learning systems (2019)
18. Meier, A., Kramer, O.: Prediction with recurrent neural networks in evolutionary dynamic optimization. In: International Conference on the Applications of Evolutionary Computation. pp. 848–863. Springer (2018)
19. Meier, A., Kramer, O.: Predictive uncertainty estimation with temporal convolutional networks for dynamic evolutionary optimization. In: International Conference on Artificial Neural Networks. pp. 409–421. Springer (2019)
20. Nguyen, T.T., Yao, X.: Continuous dynamic constrained optimization—the challenges. IEEE Transactions on Evolutionary Computation **16**(6), 769–786 (2012)
21. Nguyen, T., Yang, S., Branke, J.: Evolutionary dynamic optimization: A survey of the state of the art. Swarm and Evolutionary Computation **6**(0), 1 – 24 (2012). https://doi.org/http://dx.doi.org/10.1016/j.swevo.2012.05.001

22. Richter, H.: Evolutionary Computation for Dynamic Optimization Problems, chap. Dynamic Fitness Landscape Analysis, pp. 269–297. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
23. Rossi, C., Abderrahim, M., Díaz, J.C.: Tracking moving optima using kalman-based predictions. Evolutionary computation **16**(1), 1–30 (2008)
24. Sareni, B., Krahenbuhl, L.: Fitness sharing and niching methods revisited. IEEE transactions on Evolutionary Computation **2**(3), 97–106 (1998)
25. Simões, A., Costa, E.: Evolutionary algorithms for dynamic environments: prediction using linear regression and markov chains. In: International Conference on Parallel Problem Solving from Nature. pp. 306–315. Springer (2008)
26. Zhou, A., Jin, Y., Zhang, Q.: A population prediction strategy for evolutionary dynamic multiobjective optimization. IEEE transactions on cybernetics **44**(1), 40–53 (2013)