Using Neural Networks in Diversifying Differential Evolution for Dynamic Constrained Optimization

Track: GECH - General Evolutionary Computation and Hybrids

ABSTRACT

Frank: Change evolutionary algorithms to differential evolution where you refer to your appproaches Dynamic constrained optimisation 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, extra mechanisms are needed on top of proposed evolutionary algorithms for static domains to make them prepared for dynamic problems. Diversity mechanisms have been used to do this since long time ago. However, some approaches become popular recently like the integration of neural networks into the evolutionary process. In this work, we explore the differences of using neural networks and diversity mechanisms and the possibility of integrating them into the evolutionary optimisation process. Given the complexity and several stages of using neural networks in the process, for a fair comparison, we use wall clock timing instead of the usual number of fitness evaluations as the measure for the available time between dynamic changes. The results reveal that in environments with low frequencies of change, using neural networks can significantly improve the results even for diversifying evolutionary algorithms. **Frank: I edited the abstract please check. better to avoid abbreviations like NN here.**

KEYWORDS

Dynamic Constrained Optimization, Evolutionary Algorithm, Differential Evolution, Neural Network

ACM Reference Format:
Track: GECH - General Evolutionary Computation and Hybrids. 2020. Using Neural Networks in Diversifying Differential Evolution for Dynamic Constrained Optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference 2020 (GECCO '20).* ACM, New York, NY, USA, 8 pages. https://doi.org/

1 INTRODUCTION

Many real-world problems are in the category of dynamic constrained optimization problems (DCOPs) [17]. The dynamism is originated from factors such as variation in the demand market, unpredicted events, variable resources that may change over time [4]. In these problems, the goal 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 the changes dynamically [21]. To tackle these problems evolutionary algorithms (EAs) from the static domain are commonly used with some adaptations [21]. The modifications include mechanisms like change detection and the ability to react to the changes. Without the adaptations for EAs, premature convergence may happen, in that case, they cannot adapt well to the changes. Indeed, maintaining a diverse set of solutions in population helps to ensure the algorithm caters for changes in a dynamic environment.

Among the many approaches used to handle dynamic environments, diversity mechanisms [6, 10] **Frank: I put this into one cite command** 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 EA for different environmental changes [14]. Other approaches are memory-based approaches [23], multi-population approaches [3] or prediction methods [5]. Among many prediction methods, neural networks (NNs) [15, 24, 26, 27] have gained increasing attention in recent years [16, 18–20]. Previous work has shown that the use of neural networks can be well suited for dealing with DCOPs, especially if there is a trend in the environmental changes [19]. However, there are still some concerns regarding the application of NNs in EAs. Firstly integrating them in EAs is more complicated than using standard diversity mechanisms. The question arises whether they enhance the results to an extent that makes them worth to be used in combination with EA to solve these problems. Secondly, the consumed time by a NN needs to be taken into account for a fair comparison. This is due to the fact that 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 the use of NN in the described setting, in this work we compare common diversity mechanisms with an EA algorithm with and without NN through an empirical study. In this study we try to answer the following questions:

- Does the use of neural networks enhance the performance when used in diversifying evolutionary algorithms for dynamic constrained optimisation?
- Taking into account the time spent on neural networks, does different frequencies of change impact the suitability of neural networks in diversifying evoluitonary algorithms?

For our study, we choose differential evolution (DE) as our baseline algorithm which is a competitive algorithm in constrained and dynamic optimization for continuous spaces [1]. The experiments are repeated for different frequencies of change. 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 in DCOPs, a change is often designed to happen after a number of fitness evaluations or generations. But in that way the timing of NN is not accounted. The results reveal considering the timing spent for using NN, yet there is significant enhancement in the algorithms using NN. Moreover, for the algorithm without any other mechanism, the enhancement of the results are even to a greater extent.

The remainder of the paper is as follows. Section 2 introduces preliminaries on the topic. Our experimental setup is presented in Section 3, and in **??** we carry out detailed experimental investigations on the use of neural networks in different variants of diversifying evolutionary algorithms. 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 DCOPs, diversity mechanisms and the design of the NN are presented.

2.1 Problem statement

Generally, a 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,

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

is the search space, subject to:

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

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 optimal 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 in DCOPs 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 DCOPs 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, we choose the standard crowding method [25]. In this method, similar individuals in the population are avoided, creating genotypic diversity[1]. Instead of competition between the offspring and the parent, the offspring competes with the individual with lowest Euclidean distance. As our problem dimension is high, if we consider the only closest individual, usually the parent is chosen as the closest one. Thus, we modified the method in a way that offspring competes with N closest individuals (denoted by CwN).

Random immigrants on change: This technique (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 are considering time for each change if we insert solutions at each generation, there is not sufficient time for the optimization itself and the results are affected adversely. Thus, we only choose to insert random solutions when a change is detected.

Hyper-mutation: This method was first proposed by using an adaptive mutation operator in genetic algorithms to solve DCOPs [7]. Later, it was used in DE in [2]. When the change is detected, 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. In addition, the adaptive version for DE needs another control criteria as if we only consider DE parameters, when is converged, is not able to add diversity. The reason is DE is dependant to diversity of population to create diversity. For our version denoted by HMu, in addition to changes of DE parameters, we insert random individuals to population.

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

2.4 Structure of the neural network

NN is used to make a reliable forecast of the future optimum position. To do so, the best solutions of the previous change periods

[1]diversity can be defined at distinct levels; genotypic level refers to differences among individuals over x-values

achieved by the EA are required to build a time series. To learn the change pattern of the optimum position, NN will go through a training process. To train the network, k-best individuals of each time are collected for a couple of the previous times. We consider a procedure proposed in [13]. 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 above mentioned combination. As otherwise, the time spent for training data exponentially increases due to the large number of samples collected. 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]. The structure of the applied neural network has two hidden layers. 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. The predicted solution or its neighboring positions then can be used by EA to intensify the search in that region of the solution space. The predicted solutions are replaced with the worst individuals of the population.

3 EXPERIMENTAL METHODOLOGY

In this section, the designed experiments in terms of the test problems, the applied performance metrics, and the applied algorithm parameter settings are summarized.

3.1 Test problems and parameters settings

We used the test problem proposed in [13]. Dynamic environments are proposed in two general cases for common functions in literature: Sphere, Rosenbrock and Rastrigin. In the first two experiments, objective function is constant while the constraints change, and for the third and fourth experiments, the problem is unconstrained with dynamic objective function. Details of the designed dynamism in each experiment are: exp1: uniformly random changes on the boundaries of one linear constraint, exp2: patterned sinusoidal changes on the boundaries of one linear constraint, exp3: linear transformation of the optimum position, exp4: transformation of the optimum position in sinusoidal pattern with random amplitudes. In the first two experiments, the changes are targeted on b values (constraint boundary) of one linear constraint in the the form of $a_i x_i \le b$ [12]. We refer to in [13] for the pattern in which the position of optimum changes in each experiment. 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). Three frequencies of change: 4, 10 and 20 will be tested. As mentioned, in this work the real time is considered, so the values above represents time in seconds between the two consecutive changes. To have an idea, these values represent the following number of fitness evaluations: $4 \approx 4000$, $10 \approx 18000$, $20 \approx 40000$. Undoubtedly, these numbers are not constant for all test cases due to different time-complexity of each function and stochastic nature of EA.

The other parameters are: problem dimension=30, runs=30 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]. Variable boundaries are limited in $x_i \in [-5, 5]$. Parameters of NN are: $k = 3$: epochs=3, $n_w = 5$, batch_size=4, min_batch=20 and $n_p = 3$. All the experiments were run on a cluster, allocating 1 core (2.4GHz) and 4GB of RAM.

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 [22].

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

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.

Absolute recovery rate introduced in [22] is used to analyze the convergence behaviour of algorithms in dynamic environments. This measure infers to how quick an algorithm is to start converging to the global optimum before the next change occurs.

$$ARR = \frac{1}{m} \sum_{i=1}^{m} (\frac{\sum_{j=1}^{p(i)} |f_{best}(i,j) - f_{best}(i,1)|}{p(i)[f^*(i) - f_{best}(i,1)]}) \tag{5}$$

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

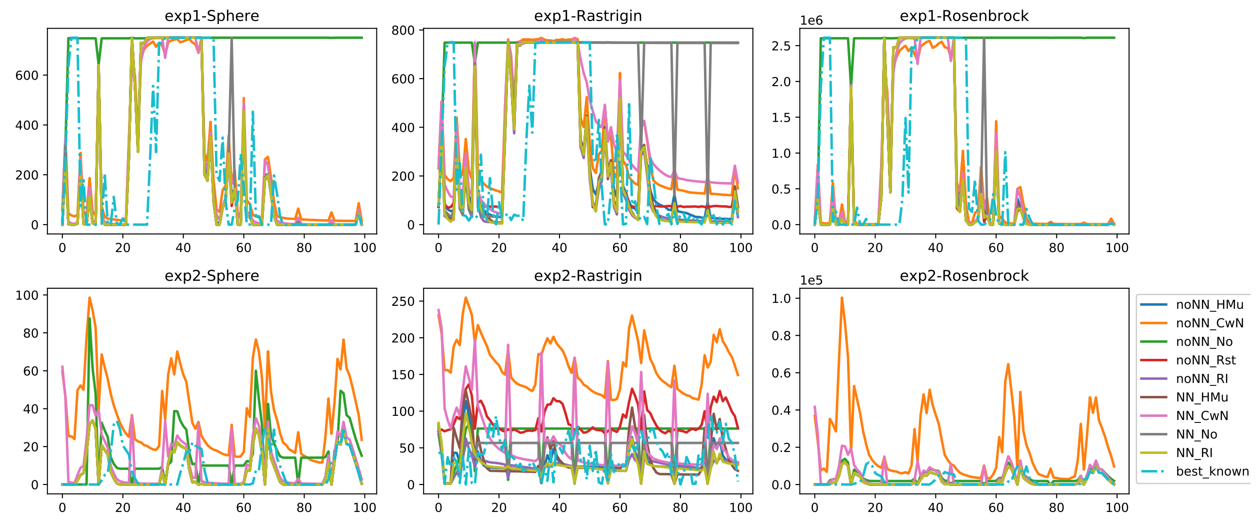


Figure 1: Fitness values of different functions for exp1 and exp2, $\tau = 10$, color-coded with each method over time

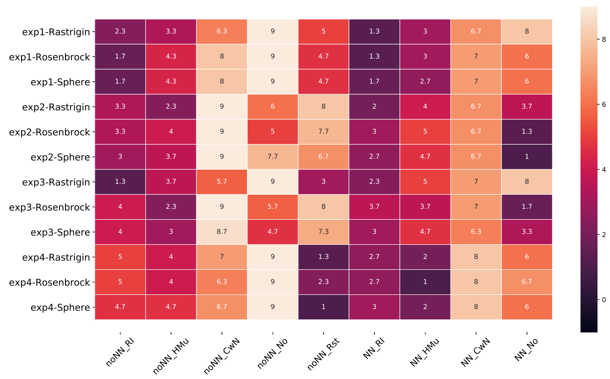


Figure 2: Heatmap of methods' rank over MOF values for each function and experiment. Lower ranks represents better performances

4 CROSS COMPARISON OF APPROACHES

The first method to illustrate the results is the best fitness values achieved by each method over time presented in Figure 1. This figure shows how the methods track the best_known along with the changes. Representing the results in this figure based of time, helps to compare the algorithms in each separate time. However, for an overall comparison of methods, we need other ways. For this, the heatmap of mean rankings of MOF values are presented in Figure 2. To achieve this rankings, first we rank each method in each set of function, experiment and for each frequency separately. Then we take the average of the means considering all the frequencies. This figure helps to investigate the performance of algorithms over each set of experiment and function.

This heatmap helps to see the comparison of methods to each other. For instance, based on the colors of heatmap, it is noticeable that NN methods compared to noNN methods have better rankings overall. Another example is with this heatmap easily can conclude that the methods including CwN are not effective in most functions and experiments. However, this heatmap is not able to show the severity of differences of methods. To the purpose of relativity analysis, we present the results of overall performance of methods in each experiment in Figure 6. To achieve standard values 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 equal 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 worse than others to a greater extent. In addition, we can observe MOF-norm values show slight difference between methods in rastrigin function with a multimodal characteristic with a lot of local optima. The major differences between methods belong to sphere and rosenbrock. To achieve a better resolution in methods comparison, we limit the y-scale; however in the cost of missing some data from the worse performing algorithms. In addition, in this figure, we can observe there is this general trend that the same variant of NN has better performance compared to its noNN counterpart. However there is an exception, for exp2 and exp3 NN is worse than noNN for HMu. In addition, severity of improvement in results created by NN is different. For instance, the improvement of the results for the algorithms with diversity mechanisms are lower compared to the algorithm without any other mechanism (noNN_No).

To clearly observe the behaviour of methods for each frequency Figure 3 is suggested. It presents a boxplot for distribution of MOF values based of each frequency for 30 runs. The results show how with increasing frequency from $\tau = 4$ to $\tau = 20$ the MOF values decrease. As the algorithms have more time to evaluate solutions and reach to better values.

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. Nonparametric tests were adopted because the samples of runs did not fit to a normal distribution based on the Kolmogorov-Smirnov test. Results of the test show in most test cases, the methods have significant difference to each other.

Figure 4 shows a heatmap of ranking of algorithms based on ARR values for different methods considering all frequencies of change. This measure is to represent which method can recover faster after a change.

The heatmap for SR values (see Figure 5) 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 1 represents the percentages of the amount of time spent for calling NN unit compared to overall optimization time. In this part we also bring the results for $\tau = 0.5$ and $\tau = 1$ to have an idea compared to other frequencies. Regardless of the experiment and function, the results for $\tau = 0.5$ show around 19-27%, $\tau = 1$ around 9-14%, $\tau = 4$ around 2-3%, $\tau = 10$ around 1% and $\tau = 20$ around 0.5%. This shows when τ is higher, it is more cheap to use NN in terms of the computational cost. When τ is low the proportion of the time for doing optimization itself 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.

5 DETAILED EXAMINATION OF USING NEURAL NETWORKS

In this section, methods are compared based of each diversity variant. We want to examine what effect NN has over each of these diversity variants.

5.1 Crowding

From Figure1, we observe that CwN variants have the biggest difference to the optimum compared to other methods. This explains their fairly ineffective performance in MOF values looking to heatmap (see 2). Based of colors of heatmap is easily noticable that variants of this method acted dissatisfying. NN could enhance the results for this method but still is inferior compared to other methods. This method is particularly inferior in exp2 and exp3, even compared to noNN_No. With promoting diversity unnecessarily, it adversely affect the convergence of the algorithm to new optimum position which is not too distant from previous optimum.

CwN has been reported as one of the best methods in the recent survey study considering other diversity methods [14]. The reason that CwN is not competitive in our study is due to following reasons: Firstly, in the literature this method showed its best effectiveness for multimodal test problems in which we have several local optima. In that case, the idea of CwN is to diversify solutions by avoiding similar individuals in each sub region of the search space [25]. In addition, in [14], the test problem included disconnected feasible regions and small feasible areas that this method showed its thriving performance. Secondly, CwN was tested on the precious work [14]

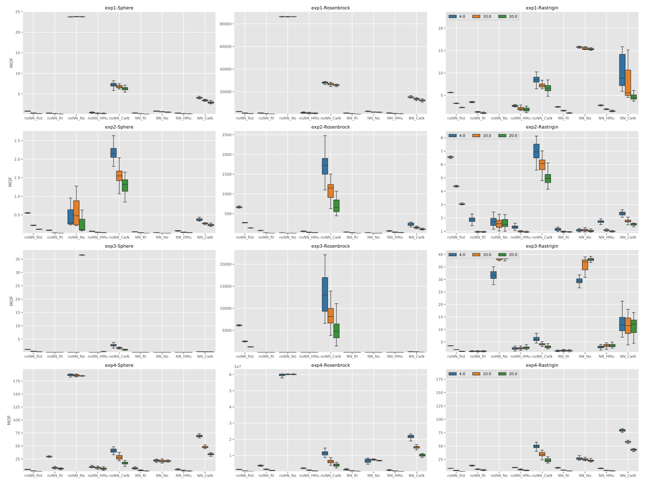
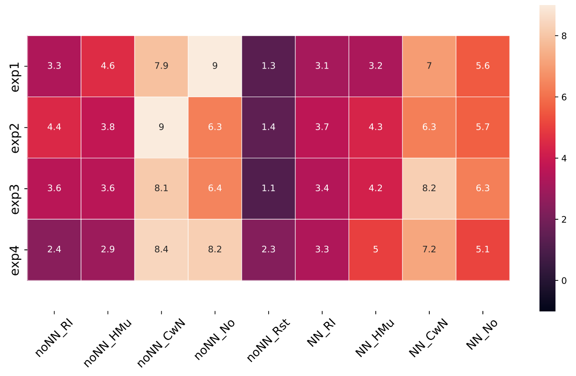


Figure 3: Distribution of MOF values for 30 runs, different frequencies (τ). Lower values represent better performances



	noNN_RI	noNN_HMu	noNN_CwN	noNN_No	noNN_Rst	NN_RI	NN_HMu	NN_CwN	NN_No
exp1	3.3	4.6	7.9	9	1.3	3.1	3.2	7	5.6
exp2	4.4	3.8	9	6.3	1.4	3.7	4.3	6.3	5.7
exp3	3.6	3.6	8.1	6.4	1.1	3.4	4.2	8.2	6.3
exp4	2.4	2.9	8.4	8.2	2.3	3.3	5	7.2	5.1

Figure 4: Heatmap of methods' rank over absolute recovery rate (ARR) values for each experiment. Lower ranks represents better performances

on a benchmark with a problem dimension size of 2. As our problem's dimension is 30, and having a crossover rate (CR) of 0.3 leads the offspring only change in a couple of dimensions compared to the parent. So in this condition, the closest individual can be the parent itself. Therefore, this method acts similar to no-diversity mechanism but with an overhead of calculating distances at each iteration. To alleviate this in our CwN version, the competition will be between $N = 5$ closest individuals to the offspring.

Checking the results of SR-values for noNN_CwN show this algorithm is barely able to get to the optima for almost all the changes in exp2 for all functions and exp3 for rosenbrock function.

5.2 Random immigrants

From this figure, it is observable that in most functions and experiments, the best value achieved by NN_RI and noNN_RI are tracking the optimum more closely. From Figure 2, it is clearly visible that RI has better ranking both for noNN and NN. Comparing them, the results for NN show better performances than noNN in all functions

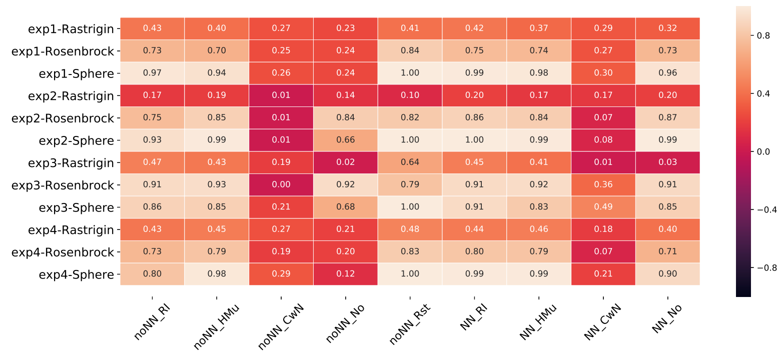


Figure 5: **Heatmap of mean values (30 runs) for success rate (SR); SR represents number of times algorithms reach to 10% of the vicinity of optima values per overall times**

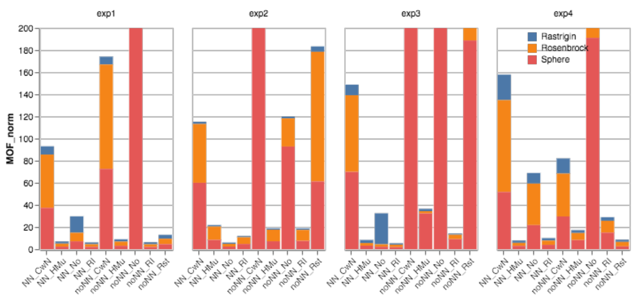


Figure 6: **MOF-norm values considering all frequencies for each method and experiment color-coded with functions**

and experiments except for rastrigin with a multimodal characteristic in exp3. 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. 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 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 interesting future work is to observe the effect of randomness over predicted solutions. In our experiment, we test with 3 inserted predicted solution and 3 randomly inserted solution. However, one can experiment with different combination of these forces.

As indicated before restart population on change can be considered as an extreme case for RI, in which we insert all the population with random solutions after a change.

noNN_Rst is ranked as the best for exp4 in which the position of the optimum have drastic changes. In the other experiments, we can clearly observe the behavior of RI is much better than this method. As mentioned before, Rst can be considered as a sever case of RI in which we discard all the previous attempts of algorithm and start new random solutions. Conversely, noNN_Rst achieve a low ranking for exp2 and exp3 in which the optimum position changes are not huge and with discarding the previous attempts of the algorithm we get worse results. This is the reason why noNN_No has an inferior performance for exp1 and exp4 on which the changes are bigger and it can not promote diversity to reach optimum. While for exp2 and exp3 this method 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. This measure is slightly biased over the first solution achieved. So if the first best solution is drastically changed for next generations, this measure reports better results. The reason this measure is better for noNN_Rst is that best solution has greater distance from the real optimum for the first generation.

noNN_Rst show medium results for SR measure. 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-mutations

HMu's results is quite similar to RI variant with slightly lower performances. The rankings in the heatmap 2, clarify this observation. However, the NN variant (NN_HMu), is less helpful for exp2 and exp3 in this case compared to RI variants for NN and noNN. This can indicate not always NN improve the results if used on top of diversity mechanisms.

Base on statistical test results, this method in its noNN version does not have significant difference with RI for exp4 (noNN_HMu and noNN_RI), sphere function, and noNN and NN for exp1 for rastrigin (noNN_HMu and NN_HMu).

6 CONCLUSIONS AND FUTURE DIRECTIONS

In this work, we investigated considering a NN to solve DCOPs together with common diversity mechanisms. To do fair investigations, we created a change after an actual running time of the algorithm instead of the number of fitness evaluations. This provides the same timing budget for the algorithm using NN and other

Table 1: **NN-time; time spent for training and using NN in proportion to overall optimization time (mean for 30 runs)**

experiment		exp1					exp2					exp3					exp4				
freq		0.5	1.0	4.0	10.0	20.0	0.5	1.0	4.0	10.0	20.0	0.5	1.0	4.0	10.0	20.0	0.5	1.0	4.0	10.0	20.0
Rastrigin	NN_CwN	0.194	0.097	0.024	0.010	0.005	0.197	0.099	0.025	0.010	0.005	0.200	0.099	0.025	0.010	0.005	0.189	0.094	0.023	0.010	0.005
	NN_HMu	0.192	0.098	0.024	0.010	0.007	0.194	0.097	0.025	0.010	0.005	0.201	0.136	0.025	0.010	0.007	0.192	0.099	0.024	0.010	0.005
	NN_No	0.197	0.099	0.024	0.010	0.005	0.196	0.098	0.025	0.010	0.005	0.201	0.101	0.024	0.010	0.005	0.196	0.095	0.024	0.010	0.005
	NN_RI	0.198	0.098	0.025	0.010	0.005	0.197	0.099	0.024	0.010	0.005	0.203	0.138	0.025	0.010	0.005	0.194	0.098	0.025	0.010	0.005
Rosenbrock	NN_CwN	0.196	0.098	0.024	0.010	0.005	0.192	0.097	0.024	0.010	0.005	0.200	0.100	0.025	0.010	0.005	0.193	0.095	0.023	0.009	0.005
	NN_HMu	0.193	0.097	0.024	0.010	0.005	0.195	0.097	0.025	0.010	0.005	0.197	0.098	0.025	0.010	0.005	0.192	0.096	0.024	0.010	0.005
	NN_No	0.197	0.096	0.024	0.010	0.005	0.194	0.099	0.024	0.010	0.005	0.202	0.101	0.025	0.010	0.005	0.193	0.095	0.024	0.010	0.005
	NN_RI	0.195	0.099	0.025	0.010	0.005	0.194	0.097	0.024	0.010	0.005	0.203	0.100	0.025	0.010	0.005	0.194	0.096	0.025	0.010	0.005
Sphere	NN_CwN	0.191	0.098	0.025	0.010	0.005	0.195	0.096	0.025	0.010	0.005	0.272	0.100	0.025	0.010	0.005	0.188	0.095	0.024	0.009	0.005
	NN_HMu	0.196	0.097	0.025	0.010	0.005	0.193	0.097	0.024	0.010	0.005	0.198	0.098	0.025	0.010	0.005	0.191	0.097	0.025	0.010	0.005
	NN_No	0.200	0.097	0.024	0.010	0.005	0.194	0.098	0.034	0.010	0.005	0.201	0.100	0.025	0.010	0.005	0.195	0.098	0.024	0.010	0.005
	NN_RI	0.268	0.100	0.025	0.010	0.005	0.200	0.099	0.024	0.010	0.005	0.200	0.101	0.025	0.010	0.005	0.188	0.097	0.025	0.013	0.005

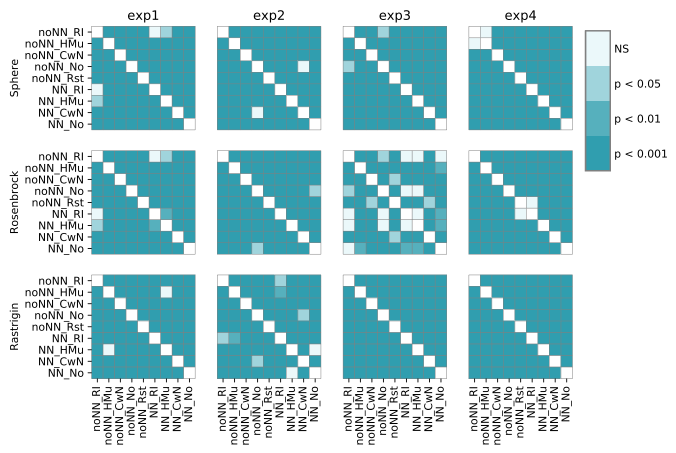


Figure 7: **Kruskal-Wallis statistical test on MOF values for τ=10, NS represents not-significant**

dynamic handling mechanisms. Our results showed considering the timing spent for training and using NN, there is significant enhancement in the results of the algorithm using NN compared to noNN counterparts. Moreover, for the algorithm with lack of any diversity mechanism, improvement is to a greater extent. The results presented in this work, belong to one specific structure of NN. We encourage as future work more NN designs be applied and compared in the context of DCOPs. Our designed experiments, can be used as a reference to investigate the future designed NN architectures in DCOPs. We observed RI in combination with NN consistently showed the best results, regardless of the experiments and functions. Thus this algorithm is worth to be investigated in more detail. One suggestion is to observe the effect of added randomness by RI, that is controlled by replacement rate, on one hand and the predicted solutions inserted by NN to the population on the other hand.

REFERENCES

[1] María-Yaneli Ameca-Alducin, Maryam Hasani-Shoreh, Wilson Blaikie, Frank Neumann, and Efrén Mezura-Montes. 2018. A comparison of constraint handling techniques for dynamic constrained optimization problems. In *2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 1–8.

[2] Maria-Yaneli Ameca-Alducin, Efrén Mezura-Montes, and Nicandro Cruz-Ramirez. 2014. Differential evolution with combined variants for dynamic constrained optimization. In *Evolutionary computation (CEC), 2014 IEEE congress on*. IEEE, 975–982. https://doi.org/10.1109/CEC.2014.6900629

[3] Jürgen Branke, Thomas Kaußler, Christian Smidt, and Hartmut Schmeck. 2000. A multi-population approach to dynamic optimization problems. In *Evolutionary design and manufacture*. Springer, 299–307.

[4] Jürgen Branke and Hartmut Schmeck. 2003. Designing evolutionary algorithms for dynamic optimization problems. In *Advances in evolutionary computing*. Springer, 239–262.

[5] C. Bu, W. Luo, and L. Yue. 2016. Continuous Dynamic Constrained Optimization with Ensemble of Locating and Tracking Feasible Regions Strategies. *IEEE Transactions on Evolutionary Computation* PP, 99 (2016), 1–1. https://doi.org/10.1109/TEVC.2016.2567644

[6] L. T. Bui, H. A. Abbass, and J. Branke. 2005. Multiobjective optimization for dynamic environments. In *2005 IEEE Congress on Evolutionary Computation*, Vol. 3. 2349–2356 Vol. 3. https://doi.org/10.1109/CEC.2005.1554987

[7] Helen G. Cobb. 1990. An Investigation into the Use of Hypermutation as an Adaptive Operator in Genetic Algorithms Having Continuous, Time-Dependent Nonstationary Environments.

[8] Kalyanmoy Deb. 2000. An efficient constraint handling method for genetic algorithms. *Computer methods in applied mechanics and engineering* 186, 2 (2000), 311–338.

[9] Joaquín Derrac, Salvador García, Daniel Molina, and Francisco Herrera. 2011. 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 (2011), 3–18. https://doi.org/10.1016/j.swevo.2011.02.002

[10] C. K. Goh and K. C. Tan. 2009. A Competitive-Cooperative Coevolutionary Paradigm for Dynamic Multiobjective Optimization. *IEEE Transactions on Evolutionary Computation* 13, 1 (Feb 2009), 103–127. https://doi.org/10.1109/TEVC.2008.920671

[11] John J Grefenstette et al. 1992. Genetic algorithms for changing environments. In *PPSN*, Vol. 2. 137–144.

[12] Maryam Hasani-Shoreh, María-Yaneli Ameca-Alducin, Wilson Blaikie, and Marc Schoenauer. 2019. On the behaviour of differential evolution for problems with dynamic linear constraints. In *2019 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 3045–3052.

[13] Maryam Hasani-Shoreh, Renato Hermoza Aragonés, and Frank Neumann. 2020. Neural Networks in Evolutionary Dynamic Constrained Optimization: Computational Cost and Benefits. (2020). arXiv:cs.NE/2001.11588

[14] Maryam Hasani-Shoreh and Frank Neumann. 2019. On the Use of Diversity Mechanisms in Dynamic Constrained Continuous Optimization. In *International Conference on Neural Information Processing*. Springer, 644–657.

[15] Iason Hatzakis and David Wallace. 2006. Dynamic multi-objective optimization with evolutionary algorithms: a forward-looking approach. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. ACM, 1201–1208.

[16] Min Jiang, Zhongqiang Huang, Liming Qiu, Wenzhen Huang, and Gary G Yen. 2017. Transfer learning-based dynamic multiobjective optimization algorithms. *IEEE Transactions on Evolutionary Computation* 22, 4 (2017), 501–514.

[17] Li Liu, Emily M Zechman, E Downey Brill, Jr, G Mahinthakumar, S Ranjithan, and James Uber. 2008. Adaptive contamination source identification in water distribution systems using an evolutionary algorithm-based dynamic optimization procedure. In *Water Distribution Systems Analysis Symposium 2006*. 1–9.

[18] Xiao-Fang Liu, Zhi-Hui Zhan, Tian-Long Gu, Sam Kwong, Zhenyu Lu, Henry Been-Lirn Duh, and Jun Zhang. 2019. Neural Network-Based Information Transfer for Dynamic Optimization. *IEEE transactions on neural networks and learning systems* (2019).

[19] Almuth Meier and Oliver Kramer. 2018. Prediction with recurrent neural networks in evolutionary dynamic optimization. In *International Conference on the Applications of Evolutionary Computation*. Springer, 848–863.

[20] Almuth Meier and Oliver Kramer. 2019. Predictive Uncertainty Estimation with Temporal Convolutional Networks for Dynamic Evolutionary Optimization. In

International Conference on Artificial Neural Networks. Springer, 409–421.

[21] T.T. Nguyen, S. Yang, and J. Branke. 2012. Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation* 6, 0 (2012), 1 – 24. https://doi.org/10.1016/j.swevo.2012.05.001

[22] Trung Thanh Nguyen and Xin Yao. 2012. Continuous dynamic constrained optimization—The challenges. *IEEE Transactions on Evolutionary Computation* 16, 6 (2012), 769–786.

[23] Hendrik Richter. 2013. *Evolutionary Computation for Dynamic Optimization Problems.* Springer Berlin Heidelberg, Berlin, Heidelberg, Chapter Dynamic Fitness Landscape Analysis, 269–297.

[24] Claudio Rossi, Mohamed Abderrahim, and Julio César Díaz. 2008. Tracking moving optima using Kalman-based predictions. *Evolutionary computation* 16, 1 (2008), 1–30.

[25] Bruno Sareni and Laurent Krahenbuhl. 1998. Fitness sharing and niching methods revisited. *IEEE transactions on Evolutionary Computation* 2, 3 (1998), 97–106.

[26] Anabela Simões and Ernesto Costa. 2008. Evolutionary algorithms for dynamic environments: prediction using linear regression and markov chains. In *International Conference on Parallel Problem Solving from Nature*. Springer, 306–315.

[27] Anabela Simões and Ernesto Costa. 2009. Improving prediction in evolutionary algorithms for dynamic environments. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. ACM, 875–882.