An Evolutionary Optimization Algorithm for Gradually Saturating Objective Functions

Dolly Sapra
University Of Amsterdam
Amsterdam, Netherlands
d.sapra@uva.nl

Andy D. Pimentel
University Of Amsterdam
Amsterdam, Netherlands
a.d.pimentel@uva.nl

ABSTRACT

Evolutionary algorithms have been actively studied for dynamic optimization problems in the last two decades, however the research is mainly focused on problems with large, periodical or abrupt changes during the optimization. In contrast, this paper concentrates on gradually changing environments with an additional imposition of a saturating objective function. This work is motivated by an evolutionary neural architecture search methodology where a population of Convolutional Neural Networks (CNNs) is evaluated and iteratively modified using genetic operators during the training process. The objective of the search, namely the prediction accuracy of a CNN, is a continuous and slow moving target, increasing with each training epoch and eventually saturating when the training is nearly complete. Population diversity is an important consideration in dynamic environments wherein a large diversity restricts the algorithm from converging to a small area of the search space while the environment is still transforming. Our proposed algorithm adaptively influences the population diversity, depending on the rate of change of the objective function, using disruptive crossovers and non-elitist population replacements. We compare the results of our algorithm with a traditional evolutionary algorithm and demonstrate that the proposed modifications improve the algorithm performance in gradually saturating dynamic environments.

CCS CONCEPTS

• **Computing methodologies** → **Genetic algorithms**; • **Theory of computation** → **Evolutionary algorithms**;

KEYWORDS

Genetic Algorithms, Dynamic Optimization

ACM Reference Format:
Dolly Sapra and Andy D. Pimentel. 2020. An Evolutionary Optimization Algorithm for Gradually Saturating Objective Functions. In *Genetic and Evolutionary Computation Conference (GECCO '20), July 8–12, 2020, Cancún, Mexico.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3377930.3389834

GECCO '20, July 8–12, 2020, Cancún, Mexico

ACM ISBN 978-1-4503-7128-5/20/07…$15.00
https://doi.org/10.1145/3377930.3389834

1 INTRODUCTION

In traditional optimization problems, all environment variables and constraints are previously known and remain static throughout the optimization task. In real life optimization problems however, an environment may change due to several factors such as fault occurrence, slow degradation, planned updates and modifications over a long period of time [16, 30]. These are called dynamic optimization problems (DOPs) wherein objective functions, constraints or parameters change over time [32]. For static optimization problems the optimization model is predetermined and designed for a specific non-moving Objective. If an environment changes infrequently after long time frames, the dynamic optimization problem can be treated as a sequence of static optimization tasks. However, in continuously changing dynamic optimizations, the model might require continuous adaptations along with the changing parameters and/or moving optimum.

Evolutionary Algorithms (EAs) are considered to be a good candidate for dynamic optimizations, which are randomized heuristics based on principles of natural evolution, and easily adapt to changes in the environment. Evolutionary Dynamic Optimization (EDO) [20] in literature is focused on recurrent or abrupt changes in the environment. There are various methodologies to detect sudden changes in the landscape [12, 38], memory based approaches to handle recurring behavior (For e.g. [28, 35]), and prediction strategies to predict the moving optimum or the population suitable in the new environment [15, 17, 37]. In a gradually changing environment, these techniques are too complicated and somewhat of an overkill. Approaches based on maintaining diversity are more suitable in such scenarios. High diversity in the population restricts the convergence of the optimization algorithm to a small search space, consequently preventing it from getting stuck in a local optimum. This allows the algorithm to monitor diverse parts of the search space so the optimization can be efficient while the environment changes slightly with each iteration.

Our focus on dynamic environments with gradually changing objective function which have a tendency to saturate, hence turning the dynamic optimization into pseudo-static optimization after saturation. This work is motivated by an evolutionary neural architecture search methodology where a population of neural networks is trained in parallel on a dataset and their architecture is modified during the training using genetic operators [26]. The objective of the search is to find a neural network topology that is efficient with high prediction accuracy. However, the prediction accuracy is a continuous and slow moving target during the training process. The maximum achievable accuracy increases with each training epoch and eventually starts to saturate when the training is nearly complete. The importance for high diversity in the population in

saturated stages is not crucial, it might actually counteract the need for the algorithm to converge to good points now that the optimization problem is pseudo-static.

Gradually Saturating Optimization Problems (GSOPs) can be found in real life problems as well, especially where the environment start to settle down after phases of small disruptions. For example, in dynamic knapsack auction problems [4] with sequential bidding for fixed total space on a commodity such as an advertisement page, assuming that bids change in value over time from different bidders, the objective is to maximize the revenue. The maximum revenue saturates after some iterations with bids being guided from market values, other bidders and historic data on such bids [27].

In this paper, we propose a new adaptive diversity control approach based evolutionary algorithm to solve dynamic optimization for Gradually Saturating Objective Functions (GSOF). We define two levels of diverseness within the population and modify the algorithm with disruptive crossovers and non-disruptive mutations while keeping the diverseness levels in mind. By introducing controlled diversity into the population through these genetic operators we are able to guide the optimization to achieve better results as proved by our experimental results. The main contributions of this paper are:

(1) Defining and discussing the dynamic Gradually Saturating Optimization Problems (GSOPs) and its challenges.
(2) Proposing an evolutionary algorithm modification that adaptively influences population diversity suitable for Gradually Saturating Objective Functions (GSOF).
(3) Validating the proposed algorithm modification through comparisons of the result of our algorithm with a baseline standard evolutionary algorithm.

The rest of the paper is organized as follows: In Section 2, we review related work in the field of dynamic optimization and diversity maintenance approaches. In Section 3, we discuss the specifics of our problem and formally define it. In Section 4, we outline the proposed methodology with details of the algorithm and how diversity is adaptively maintained in the population. Section 5 is dedicated to the experiments and evaluation of our approach. Finally, we conclude the paper in Section 6 with some directions for future work.

2 RELATED WORK

Dynamic optimization problems (DOPs) are characterized by a variety of mechanisms that can cause a change in the problem environment during the optimization process. Some of the attributes that outline a dynamic behavior are frequency, severity and predictability of the change. In evolutionary solutions for DOPs, these environmental changes are handled in various ways such as memory based approaches , multi-population based techniques, prediction based methods and diversity based approaches. Some hybrid approaches such as memetic algorithms [34], that combine different aspects of these approaches have also been proposed over the years. An appropriate approach is chosen depending on the type of dynamism present in the optimization problem's environment.

For periodical changes, memory based approaches are suitable where some candidates from the population are stored for later use. It reduces the computation complexity by making good candidates readily available in recurring situations. Memory can be implicitly encoded in genotype [35] or explicitly stored externally [28, 36]. For sudden and irregular changes, the main concern is to detect when the change occurs and to adapt the population to be suitable for the new optimum as quickly as possible. A change can be indicated by population statistics [24] or external sensors [25].

Multi-population approaches divide the population into multiple sub-populations, and each one tracks the optimum in different promising search areas [6, 7, 19, 21]. Sub-populations are generally independent of one another and each one might employ its own search technique or track different optimum in multi-objective optimization problems. Sub-populations usually remain disparate throughout the process, but some algorithms combine them after some iterations to combine the search space explored individually by each sub-population [22].

For gradually changing targets, which is the focus of this paper, the techniques for maintaining diversity are more relevant. Diverse individuals keeps the search space broad and prevents the algorithm from prematurely converging. This allows a wider exploration and lets algorithm move its focus in the search area with moving optimum. The benefits of diversity in evolutionary algorithm has been surveyed and analyzed in [29]. A classification of diversity maintaining, controlling and learning mechanisms is discussed extensively in [10].

Hyper-mutation [9] and random immigrants [13] are two well-known techniques and are widely used for introducing diversity in the evolutionary algorithms. Hyper-mutation increases mutation rate for a period of time when a change is detected and random immigrants introduces randomly generated individuals into the population with each generation. Variable local search [33] is similar to hyper-mutation, it increases mutation strength upon detecting a change, instead of changing the mutation rate. Fitness sharing [5] penalizes similar individuals to encourage diversity in the population. In dynamic problems where high diversity is critical, the problem is converted to a multi-objective optimization problem with diversity as an extra objective to be maintained throughout the optimization process [31].

Our approach is closer to hyper-mutation and variable local search approaches, where a disruptive genetic operator is used to introduce diversity in the population. However, that is where the similarity ends. There is little need to detect the changes in gradually moving functions, moreover high population diversity is not a requisite near the saturation points. Our work differs from most diversity maintenance techniques in the way diversity level is explicitly guided in an adaptive manner based on the rate of change of the moving optimum.

3 PROBLEM DEFINITION

This work is motivated from neural architecture search methodology where we train a population of neural networks and the objective is to find a good topology, such that the accuracy of the neural network is maximum (for the given dataset) upon completion of the training. In this section, we give an overview of neural networks and introduce their training function as a GSOF and finally define the objective of the optimization.

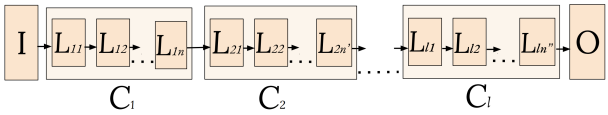


Figure 1: Linear chain structured topology where all layers are grouped into clusters of same type and have same constraints.

3.1 Neural Network

A neural network $\widetilde{nn}$ is defined by its topology T, activation functions ϕ and coefficients $\omega \in \mathbb{R}$.

$$\widetilde{nn} = (T, \phi, \omega)$$

In this work, a neural network topology is considered as a linear chain of layers, where output from one layer serves as input to the next layer, grouped into clusters of same layer type and constraints. Figure 1 illustrates this concept.

$$T = \{I, C_1, C_2 ... C_l, O\},$$

$$C_k = \{L_{k1}, L_{k2} ... L_{kn}\} : \ \beta^{min} \leq n \leq \beta^{max}$$

$$L_{ki} = \{L | \, L \in [C^k_{type}, \, \eta_{ki}, \, \pi_{ki}]\} : \ \eta^k_{low} \leq \eta_{ki} \leq \eta^k_{up}$$

$$\{\beta^{min}, \, \beta^{max}, \, \eta^k_{low}, \eta^k_{up} \in \mathbb{N}^+\}$$

where I and O are input and output layers respectively; C_k is a cluster of n layers bounded by minimum and maximum values $(\beta^{min}, \beta^{max})$. Each layer is defined by the parents cluster's layer type (e.g. convolution, pooling, fully connected), along with the number of neurons and layer specific parameters, π_{ki}, such as kernel size and stride in a convolutional layer. The number of neurons in a layer are bounded by $(\eta^k_{low}, \eta^k_{up})$, the values of which are specified by the parent cluster and the parameters π_{ki} are independent of other layers in the same cluster. All layer parameters are designed to keep constant input-output feature map size in the cluster.

3.2 Training a Neural Network

To start training a neural network, all available data is split into train and test sets. The train data set is used to train the neural network while the test data is set aside for performance evaluation. With each training iteration, the coefficients of the neural network get updated. A neural network during training can be considered a function of time, $\widetilde{nn}(t)$ and is trained on a subset of train data using $f_{train}()$ upto τ_{max} time. $\widetilde{nn}(0)$ is initialized with random coefficients.

$$\widetilde{nn}(t) = (T, \phi, \omega(t))$$

$$\forall \, 1 \leq t \leq \tau_{max}, \ \widetilde{nn}(t) = f_{train}(\widetilde{nn}(t-1))$$

Every Neural network $\widetilde{nn} \in \widetilde{NN}$, where $\widetilde{NN}$ is a set of all possible neural networks and every topology $T \in \widetilde{T}$, where $\widetilde{T}$ is the set of all possible topologies, with predefined constraints. We use validation accuracy on test data, $Acc(\widetilde{nn})$, as the main performance metric of a neural network, which is dependent on topology as well as its training time or the number of iterations of the evolutionary algorithm.

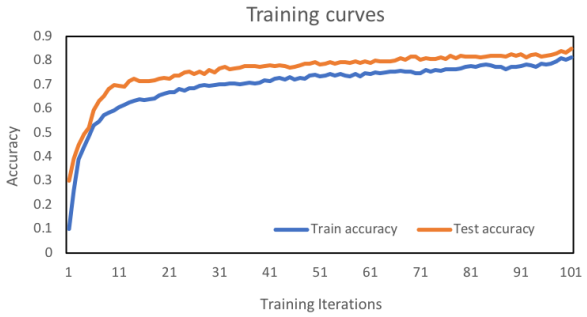


Figure 2: Example of training curves. Train and test accuracy are evaluated on the train and test data set respectively.

Figure 2 illustrates an example of a neural network performance during the training process, also called as training curves. The training curve represents the iterative performance of a neural network during the training and closely resembles an increasing saturating function such as functions from the power law or the sigmoidal family [11]. Accuracy during training is defined as:

$$\forall \, 1 \leq t \leq \tau_{max}, \ Acc(\widetilde{nn}(t)) = Acc(\widetilde{nn}(t-1)) + \frac{\partial (Acc(\widetilde{nn}))}{\partial t}$$

In GSOF, $\frac{\partial (Acc(\widetilde{nn}))}{\partial t}$ is small and in the saturation phase it is almost zero.

3.3 Optimization Objective

A population of several neural networks is trained in parallel on a given dataset. The optimization objective is to find the neural network topology with maximum performance evaluation. Given, $\widetilde{T}$ as the set of all possible topologies and $\widetilde{NN}$ as the set of all possible neural networks. Mathematically, the objective is to find neural network $\widetilde{nn}'$ with topology $(T' \in \widetilde{T})$, such that

$$max_{\widetilde{nn}' \in \widetilde{NN}} \ Acc(\widetilde{nn}'(T', \phi, \omega(t)),$$

$$\text{and feasible}(T') = true$$

4 METHODOLOGY

This section describes the evolutionary algorithm for GSOP and other design choices pertaining to the it.

4.1 Topology Representation and Diversity

To reduce computational complexity, we have a fixed sized genotype representation of the topology by fixing the number of clusters for every topology. For different problems with each a different dataset, the number of clusters and constrains in a cluster may vary. Even though the number of clusters is fixed, each cluster can have a variable number of layers, resulting in a different total number of layers in every randomly created topology. In a population based evolutionary methodology, there are many diversity maintenance techniques as discussed in Section 2. Diversity in this paper refers to the distance between individual neural network topologies. We do not measure the distance between individuals explicitly, instead

we define two coarse-grained levels of diverseness. That is, two individuals are *dissimilar* if the total number of layers or layer types are different from each other. Two individuals are *similar* to each other when the total number of combined layers as well as type of each layer is same for both. Individual layer parameters (π_l) may be different for every layer. The layer parameters play a big role in making a neural network more efficient than others even when having exactly same layer types. Two individuals with *similar* diversity level does not imply they have similar prediction accuracy. We explore these layer parameters during the algorithm through the mutation operator, but mutation does not influence the diversity as the number of layers remain unchanged.

4.2 Genetic Operators

By defining two levels of diverseness, we can differentiate the behavior of mutation and crossover operators w.r.t. the diversity level it introduces in the population. We implement a disruptive genetic crossover operator to introduce more diversity by creating children with a different number of total layers, whereas mutations operate on a layer's parameters only, therefore not contributing to a change in diversity levels of the population. We describe both operators below.

4.2.1 Mutation. Our mutate operator randomly selects a layer from the neural network topology and changes only one the layer parameters (π_l) by a small value. Change in the number of neurons of selected layer is constrained by ϱ_m%. The number of layers in the offspring remain the same creating a *similar* individual in terms of diversity. The mutate operations are designed to be function preserving taken from [8, 18], which means that the disturbance on the training process and on the current performance of the child topology is minimal. As the training continues, coefficients values of the child topology change and these little changes may contribute to a better performing topology towards the convergence of the optimization algorithm. Algorithm 1 describes the mutate operator on a topology, T_{parent}, and returns the mutated topology T_{child}.

Algorithm 1: Mutate

Inputs: T_{parent}, ϱ_m

$L_m \leftarrow randomLayer(T_{parent})$

$L_m \leftarrow ChangeParameterOf(L_m, \varrho_m)$

$T_{child} \leftarrow Merge(T_{parent}, L_m)$

return T_{child}

4.2.2 Crossover. Out crossover operates on two individuals, randomly selects cluster position and swaps the whole cluster between both the topologies. Figure 3 exemplify a swap operator with topology having four clusters. The reason for this being a disruptive operator comes from the fact that even though the layers being swapped are roughly at the same position in the layer chain, the number of layers present in each cluster are different. One cluster of two convolutional layers might get swapped with another cluster containing five convolutional layers, thus creating diverse *dissimilar* offsprings. The clusters at that same positions are designed to keep the same input-output feature map sizes, so the crossover does not result in a corrupt neural network. There is some disturbance caused to the training process by the crossover operator, but as training continues, the loss incurred is observed to have recovered in after a few iterations. Algorithm 2 shows the steps for the crossover operator, which accepts two parent topologies and returns children topologies with the crossover applied.

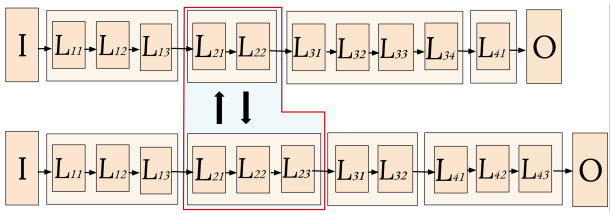


Figure 3: Example of crossover operation in two neural networks with 4 clusters each.

Algorithm 2: Crossover

Inputs: $T_{parent1}, T_{parent2}$

$k \leftarrow random(T_{parent1}.Num_{cluster})$

$T_{child1}, T_{child2} \leftarrow SwapClusterAt(k, T_{parent1}, T_{parent2})$

return T_{child1}, T_{child2}

4.3 Adaptive diversity

The desired level of population diversity for the optimization varies depending on the rate of change in the GSOF at any given time. High diversity is advantageous when the rate of change of the objective function is high and vice versa. In the proposed methodology, we adaptively influence the diversity via crossover probability modification throughout the iterative process. Crossover probability is an individual's selection probability to undergo crossover operation. When the shape of the objective function is known apriori, it is possible to setup an offline adaptive control function with expected rate of change to guide it. In absence of this prior knowledge, the average rate of change of the objective function over a short interval gives a good indication of diversity needed at any given time point. We call this an online adaptive diversity control function.

4.3.1 Offline Adaptive. For an offline adaptive crossover probability function, we select an exponential decay function which loosely represents the inverted accuracy function during training. Where α is the decay factor, crossover probability, P_r as a function of time is defined as:

$$P_r(t) = P_r(0) * \alpha^t, : 0 < \alpha < 1 \tag{1}$$

4.3.2 Online Adaptive. For the online adaptive crossover probability function, the change in objective function is monitored and crossover probability is modified based on its current rate of change. This generic function can be applied to any GSOF based optimization. With γ as the scale factor, crossover probability is modified by the following:

$$P_r(t) = P_r(0) * \gamma * \frac{\partial(Acc(\widetilde{nn}))}{\partial t} \tag{2}$$

4.4 Selection and Replacement

In dynamic optimization, selection and replacement policies play an important role in managing diversity as well as retaining the good individuals. We adopt a remove-worst strategy to select the next generation of the population with a very low replacement rate, Ω, of about 2-5% of the total population. The worst performing individuals are removed from the next generation and from the remaining population individuals are selected for reproduction based on mutation probability(P_m) and crossover probability(P_r). The population size is kept constant, so some individuals may reproduce more than once. Since training in itself is a stochastic process, there is a chance that an individual achieves much higher accuracy as compared to the rest of population by accident. To prevent this individual from crowding and dominating the search and resulting in a loss in diversity, we follow a non-elitist random selection policy. Every individual has an equal chance of being selected to create an offspring which replaces the worst performing individual.

4.5 Algorithm

Here we consolidate all the concepts in one place and outline the modified evolutionary algorithm for GSOPs. Algorithm 3 outlines the complete approach.

Algorithm 3: Evolutionary Optimization

Evolutionary Inputs: N_g, N_p, P_r, P_m, Ω
Training Inputs : τ_{params}, δ_k

$\wp_o \leftarrow InitializePopulation(N_p)$
for $i \leftarrow 0 .... N_g$ **do**
  $\wp_i \leftarrow Train(\wp_{i-1}, \tau_{params}, \delta_k)$
  $Acc_i \leftarrow EvaluatePopulation(\wp_i)$
  $\wp_{best} \leftarrow BestSelection(\Omega, \wp_i, Acc_i)$
  $\wp_r \leftarrow randomFrom(\Omega, \wp_i)$
  update $\wp_i \leftarrow \wp_{best} + \wp_r$
  $\wp_{mu} \leftarrow MutatePopulation(\wp_i, P_m)$
  $P'_r \leftarrow updateCrossoverProbability(P_r, i, Acc_i)$
  $\wp_{rc} \leftarrow CrossoverPopulation(\wp_i, P'_r)$
  $\wp_{remaining} \leftarrow UnchangedPopulation()$
  update $\wp_i \leftarrow \wp_{mu} + \wp_{rc} + \wp_{remaining}$
end
return $BestCandidatesOf(\wp_{N_g})$

InitializePopulation() generates a population of neural networks of size N_p using a factory pattern class for the topology genotype based on clusters and initializes them by training them for one epoch. Afterwards, this iterative algorithm runs for N_g generations. *Train*() trains all individuals with randomly selected data, from the train dataset, of size δ_k using τ_{params} training parameters, such as learning rate and batch size. δ_k defines the interval at which genetic operators are applied for topology modification during the training process. *EvaluatePopulation*() evaluates the population using the test set and *BestSelection*() selects the $(1 - \Omega)$% best individuals using the accuracy on the validation set achieved so far. To keep population size constant, Ω% randomly selected individuals are added back to the pool. *MutatePopulation*() and *CrossoverPopulation*() are evolutionary operators, they select individuals from the population with selection probability of P_m and P_r respectively and use Algorithms 1 and 2 to operate on selected individuals. The main modification of this algorithm comes from the function *updateCrossoverProbability*() , which is called in every iteration to modify the Crossover probability rate depending on the chosen approach, i.e. according to equation (1) or (2). Finally, the algorithm returns the best candidates of neural networks from the final population.

5 EXPERIMENTAL STUDY

In this section, we evaluate our algorithm using PAMAP2 [23] dataset for human activity recognition and outline the setup of our experiments. We have used the Java based Jenetics library [1] for evolutionary computation and the Python based Caffe2 [3] library for training and testing. We used the ONNX [2] format to represent and transfer the neural networks across different modules. Our experiments use one GPU (GeForce RTX 2080) for training the neural networks, however the algorithm is scalable and is able to use multiple GPUs in parallel during each iteration.

5.1 Setup

5.1.1 DataSet. The PAMAP2 dataset provides recordings from three body worn Inertial Measurement Units (IMU) and a heart rate monitor. Together, the input data is in the form of time-series from 40 channels performing 12 activities. We do not consider optional activities in this experiment. Following [14], recordings from participants 5 and 6 are used as testing set. IMUs' recordings are downsampled to 30 Hz and a sliding window approach with a window size of 3s (100 samples) and a step size of 660ms (22 samples) is used to segment the sequences. To augment the data, a sliding window is moved by different step sizes while keeping the window size the same at 3s.

5.1.2 Topology. The topology structure for PAMAP2 has five clusters, with the total number of layers varying between 7 and 20. Table 1 indicates each cluster's details and various constraints. Every layer is followed by a ReLu activation layer and number of neurons are modified in steps of 8 during the mutation operation resulting in total design points to the tune of 10^7.

Table 1: Topology Search Space

CLuster Type	Layers		Neurons		Kernel	
	β^{min}	β^{max}	η_{low}	η_{up}	K_{min}	K_{max}
C_1:Convolution	2	7	64	128	3x1	7x1
C_2:MaxPool	1	1	1	1	2x1	2x1
C_3:Convolution	2	7	96	256	3x1	7x1
C_4:GlobalMaxPool	1	1	1	1	2x1	2x1
C_5:Fully Connected	1	4	128	512	-	-

5.1.3 Algorithm Parameters. The parameters of the algorithm for all variants were the same and are summarized in Table 2. The parameters were determined during preliminary experiments. Training parameters (τ_{params}) are learning rate and batch size as listed in

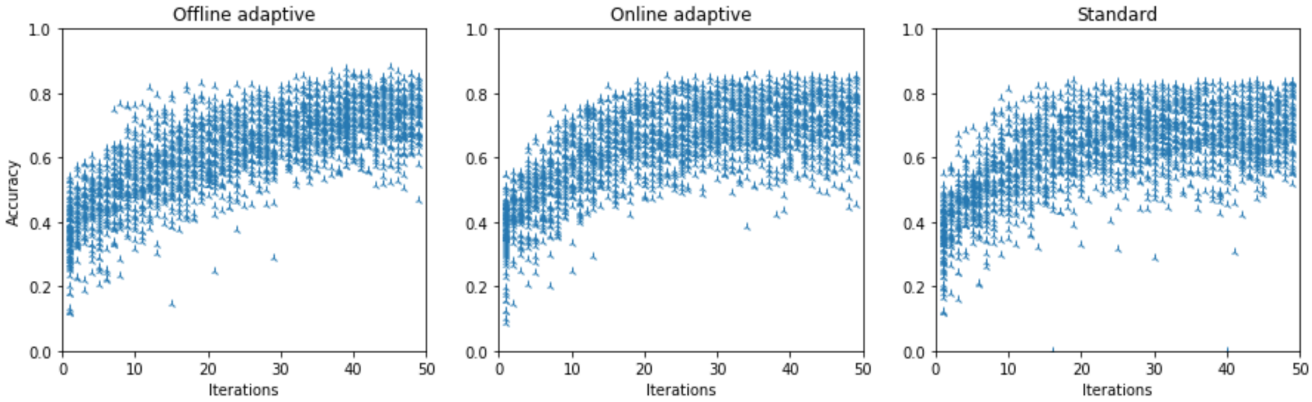


Figure 4: Accuracy values for all of the neural networks evolving during an experiment each for offline adaptive, online adaptive and standard evolutionary algorithms.

the Table 2, which were used with the Adam optimizer for training the neural networks.

Table 2: Algorithm parameter values in experiments

Parameter		Value
Mutation change rate	ϱ_m	0.12
Mutation selection probability	P_m	0.3
Initial Crossover selection probability	$P_r(0)$	0.4
Adaptive offline decay factor	α	0.95
Adaptive online scale factor	γ	60
Population size	N_p	50
No of iterations	N_g	50
Population replacement rate	Ω	0.03
Training interval size	δ_k	20,000
Training Parameters	τ_{params}	
Learning rate		$1e^{-4}$
Batch size		50

5.2 Results

Figure 4 illustrates how the population of neural networks is evolving while training during one run each of offline adaptive, online adaptive and standard evolutionary algorithms. Each algorithm takes approximately 10 hours to complete with majority of time spent in training a neural network. The population size is fixed during the genetic iterations, so the total number of training operations are same for all the algorithms. We consider standard evolutionary algorithm to have static mutation and crossover probabilities. Experimental results of mean of average and best accuracy after completion of optimization algorithm are presented in Table 3. All presented numbers are average of 10 independent runs. Figure 6 shows the performance of offline adaptive, online adaptive and the standard evolutionary algorithm with best and average accuracy found during each iteration.

Table 3: Experimental results of the mean average accuracy and best accuracy in the final iteration

Algorithm	Average	Best
Offline adaptive	0.739(±0.012)	0.859(±0.010)
Online adaptive	0.718(±0.009)	0.842(±0.016)
Standard	0.688(±0.015)	0.809(±0.012)

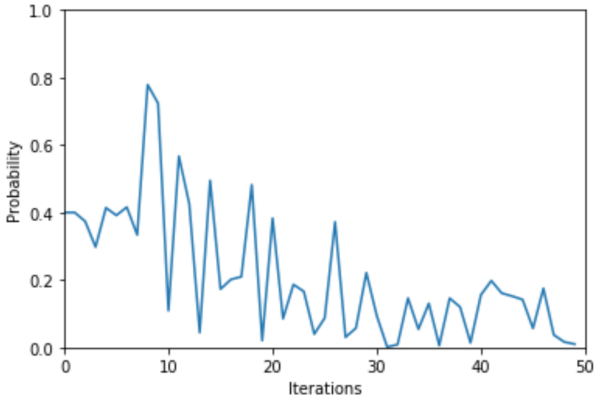
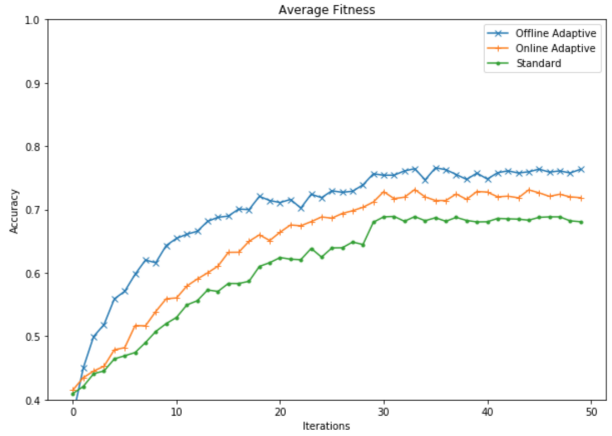
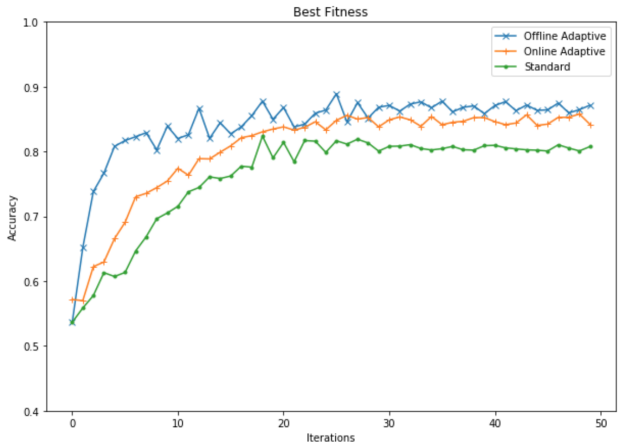


Figure 5: Crossover probability during iterations of online adaptive evolutionary algorithm.

It is clear from the results that both of the adaptive diversity control evolutionary algorithms outperform the standard evolutionary algorithm. Among the adaptive varieties, the offline version performs slightly better than the online version. It is to be expected as the rate of change of accuracy function can have small *blips* because of stochastic nature of training process, which leads to a lower or a higher crossover probabilities for short intervals. Figure 5 shows that the graph of crossover probability with respect to the algorithm iterations is not as smooth as the exponential decay



(a) Average accuracy of population



(b) Best accuracy

Figure 6: Training curves. Average accuracy refers to the average performance of whole population at the given optimization iteration. Best accuracy refers to best found performance of an individual model from the population.

function of offline adaptive algorithm. However, the trend is similar and we see decreasing values of crossover probability over the iterations.

We see better results in the Offline adaptive version, where the crossover probability curve is smooth and diversity is tightly controlled over iterations. Results illustrate that whenever the GSOF is known, it is preferable to design the diversity control function based on this knowledge. However, evolutionary algorithm with online adaptive diversity function still performs better than the standard evolutionary algorithm with fixed crossover probability suggesting that in saturation phase, having less diversity is better to explore local search space. Best neural networks found through these algorithms can be further processed, modified or trained as needed outside this algorithm. The best one found through offline adaptive diversity control approach was modified to add Batch Normalization layers after every convolutional layer and trained further with a dropout ration of 0.2, to achieve 94.3% accuracy.

6 CONCLUSION

This paper proposed an adaptive diversity control based methodology for evolutionary algorithms suitable for dynamic optimization of GSOFs. By defining coarse grained diversity levels and designing genetic operators specific to each level, we can influence the population diversity meaningfully. We validated our approach by performing topology search during training of neural networks with accuracy function as the objective of the optimization. We showed that for GSOFs, adaptive diversity control based on the rate of change of objective function provides better candidates with higher accuracies than standard evolutionary algorithm.

As future work we would like to work on benchmark generator for GSOFs. Another task would be to measure diversity of the population explicitly and modify the adaptive control function through direct feedback.

ACKNOWLEDGMENTS

This project has received funding from the European Union's Horizon 2020 Research and Innovation programme under grant agreement No. 780788.

REFERENCES

[1] 2019. Jenetics library. (2019). https://jenetics.io/
[2] 2019. ONNX: Open Neural Network Exchange Formet. (2019). https://onnx.ai/
[3] 2019. PyTorch: An open source deep learning platform. (2019). https://pytorch.org/
[4] Gagan Aggarwal and Jason D Hartline. 2006. Knapsack auctions. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*. Society for Industrial and Applied Mathematics, 1083–1092.
[5] HC Andersen. 1991. An investigation into genetic algorithms, and the relationship between speciation and the tracking of optima in dynamic functions. *Brisbane, Australia: Honors, Queensland Univ* (1991).
[6] Tim Blackwell and Jürgen Branke. 2006. Multiswarms, exclusion, and anti-convergence in dynamic environments. *IEEE transactions on evolutionary computation* 10, 4 (2006), 459–472.
[7] Chenyang Bu, Wenjian Luo, and Lihua Yue. 2016. Continuous dynamic constrained optimization with ensemble of locating and tracking feasible regions strategies. *IEEE Transactions on Evolutionary Computation* 21, 1 (2016), 14–33.
[8] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. 2015. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641* (2015).
[9] Helen G Cobb. 1990. *An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments.* Technical Report. Naval Research Lab Washington DC.
[10] Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. 2013. Exploration and exploitation in evolutionary algorithms: A survey. *ACM computing surveys (CSUR)* 45, 3 (2013), 1–33.
[11] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. 2015. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.
[12] David Fagan and Michael O'Neill. 2017. Exploring Target Change Related Fitness Reduction in the Moving Point Dynamic Environment. In *International Conference on Theory and Practice of Natural Computing*. Springer, 63–74.
[13] John J Grefenstette et al. 1992. Genetic algorithms for changing environments. In *PPSN*, Vol. 2. 137–144.
[14] Nils Y Hammerla, Shane Halloran, and Thomas Plötz. 2016. Deep, convolutional, and recurrent models for human activity recognition using wearables. *arXiv preprint arXiv:1604.08880* (2016).
[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*. 1201–1208.
[16] Yaochu Jin and Jürgen Branke. 2005. Evolutionary optimization in uncertain environments-a survey. *IEEE Transactions on evolutionary computation* 9, 3 (2005), 303–317.
[17] Wee Tat Koo, Chi Keong Goh, and Kay Chen Tan. 2010. A predictive gradient strategy for multiobjective evolutionary algorithms in a fast changing environment. *Memetic Computing* 2, 2 (2010), 87–110.
[18] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2016. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710* (2016).

[19] Wenjian Luo, Juan Sun, Chenyang Bu, and Ruikang Yi. 2018. Identifying species for particle swarm optimization under dynamic environments. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 1921–1928.

[20] Trung Thanh Nguyen, Shengxiang Yang, and Juergen Branke. 2012. Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation* 6 (2012), 1–24.

[21] Daniel Parrott and Xiaodong Li. 2006. Locating and tracking multiple dynamic optima by a particle swarm model using speciation. *IEEE Transactions on Evolutionary Computation* 10, 4 (2006), 440–458.

[22] Hani Pourvaziri and B Naderi. 2014. A hybrid multi-population genetic algorithm for the dynamic facility layout problem. *Applied Soft Computing* 24 (2014), 457–469.

[23] Attila Reiss and Didier Stricker. 2012. Introducing a new benchmarked dataset for activity monitoring. In *2012 16th International Symposium on Wearable Computers*. IEEE, 108–109.

[24] Hendrik Richter and Franz Dietel. 2010. Change detection in dynamic fitness landscapes with time-dependent constraints. In *2010 Second World Congress on Nature and Biologically Inspired Computing (NaBIC)*. IEEE, 580–585.

[25] Shaaban Sahmoud and Haluk Rahmi Topcuoglu. 2016. Sensor-based change detection schemes for dynamic multi-objective optimization problems. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 1–8.

[26] Dolly Sapra and Andy D Pimentel. 2020. Constrained evolutionary piecemeal training to design convolutional neural networks. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer.

[27] Shamik Sengupta and Mainak Chatterjee. 2008. Designing auction mechanisms for dynamic spectrum access. *Mobile Networks and Applications* 13, 5 (2008), 498–515.

[28] Anabela Simões and Ernesto Costa. 2011. Memory-based CHC algorithms for the dynamic traveling salesman problem. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. 1037–1044.

[29] Dirk Sudholt. 2020. The benefits of population diversity in evolutionary algorithms: a survey of rigorous runtime analyses. In *Theory of Evolutionary Computation*. Springer, 359–404.

[30] Christopher LE Swartz and Yoshiaki Kawajiri. 2019. Design for dynamic operation-A review and new perspectives for an increasingly dynamic plant operating environment. *Computers & Chemical Engineering* (2019).

[31] Andrea Toffolo and Ernesto Benini. 2003. Genetic diversity as an objective in multi-objective evolutionary algorithms. *Evolutionary computation* 11, 2 (2003), 151–167.

[32] Krzysztof Trojanowski and Zbigniew Michalewicz. 2000. Evolutionary optimization in non-stationary environments. *Journal of Computer Science & Technology* 1 (2000).

[33] Frank Vavak, KA Jukes, Terrence C Fogarty, et al. 1998. Performance of a genetic algorithm with variable local search range relative to frequency of the environmental changes. *Genetic Programming* (1998), 22–25.

[34] Hongfeng Wang, Dingwei Wang, and Shengxiang Yang. 2009. A memetic algorithm with adaptive hill climbing strategy for dynamic optimization problems. *Soft Computing* 13, 8-9 (2009), 763–780.

[35] Shengxiang Yang. 2003. Non-stationary problem optimization using the primal-dual genetic algorithm. In *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.*, Vol. 3. IEEE, 2246–2253.

[36] Shengxiang Yang and Xin Yao. 2008. Population-based incremental learning with associative memory for dynamic environments. *IEEE Transactions on Evolutionary Computation* 12, 5 (2008), 542–561.

[37] Aimin Zhou, Yaochu Jin, and Qingfu Zhang. 2013. A population prediction strategy for evolutionary dynamic multiobjective optimization. *IEEE transactions on cybernetics* 44, 1 (2013), 40–53.

[38] Juan Zou, Qingya Li, Shengxiang Yang, Jinhua Zheng, Zhou Peng, and Tingrui Pei. 2019. A dynamic multiobjective evolutionary algorithm based on a dynamic evolutionary environment model. *Swarm and evolutionary computation* 44 (2019), 247–259.