Convolutional neural network surrogate-assisted GOMEA

Arkadiy Dushatskiy
Centrum Wiskunde & Informatica
Amsterdam, the Netherlands
Arkadiy.Dushatskiy@cwi.nl

Adriënne M. Mendrik
Netherlands eScience Center
Amsterdam, the Netherlands
A.Mendrik@esciencecenter.nl

Tanja Alderliesten
Amsterdam UMC
University of Amsterdam
Amsterdam, the Netherlands
T.Alderliesten@amc.uva.nl

Peter A.N. Bosman
Centrum Wiskunde & Informatica
Amsterdam, the Netherlands
Delft University of Technology
Delft, the Netherlands
Peter.Bosman@cwi.nl

ABSTRACT

We introduce a novel surrogate-assisted Genetic Algorithm (GA) for expensive optimization of problems with discrete categorical variables. Specifically, we leverage the strengths of the Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA), a state-of-the-art GA, and, for the first time, propose to use a convolutional neural network (CNN) as a surrogate model. We propose to train the model on pairwise fitness differences to decrease the number of evaluated solutions that is required to achieve adequate surrogate model training. In providing a proof of principle, we consider relatively standard CNNs, and demonstrate that their capacity is already sufficient to accurately learn fitness landscapes of various well-known benchmark functions. The proposed CS-GOMEA is compared with GOMEA and the widely-used Bayesian-optimization-based expensive optimization frameworks SMAC and Hyperopt, in terms of the number of evaluations that is required to achieve the optimum. In our experiments on binary problems with dimensionalities up to 400 variables, CS-GOMEA always found the optimum, whereas SMAC and Hyperopt failed for problem sizes over 16 variables. Moreover, the number of evaluated solutions required by CS-GOMEA to find the optimum was found to scale much better than GOMEA.

CCS CONCEPTS

• **Mathematics of computing → Evolutionary algorithms**;

KEYWORDS

GOMEA, surrogate-assisted GA, convolutional neural network, surrogate model, discrete optimization, expensive optimization

ACM Reference Format:
Arkadiy Dushatskiy, Adriënne M. Mendrik, Tanja Alderliesten, and Peter A.N. Bosman. 2019. Convolutional neural network surrogate-assisted GOMEA. In *Genetic and Evolutionary Computation Conference (GECCO '19), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3321707.3321760

GECCO '19, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-6111-8/19/07…$15.00
https://doi.org/10.1145/3321707.3321760

1 INTRODUCTION

Optimization problems with time-consuming objective function evaluations (expensive optimization) arise in different domains. With the recent progress in deep learning, expensive optimization has become a topic of great interest as there are several connected expensive optimization tasks, such as deep neural network hyperparameter optimization. In this paper, we consider single-objective binary problems in a Black Box Optimization (BBO) setting with the assumption that function evaluations are expensive (i.e., training a deep neural network).

Many of the currently used expensive optimization algorithms are based on Bayesian Optimization (BO) and consider problems with real-valued variables [1]. The traditional BO algorithms, which use Gaussian Process models, have two major problems: a limit on the number of variables (in most works the functions in the experiments have only several variables) and difficulties with applications to problems with discrete categorical variables [20]. Two advanced algorithms commonly used for expensive optimization tasks, including applications to deep learning, are: SMAC [11] and Hyperopt [2]. SMAC has an option to use Random Forests (RF) instead of Gaussian Processes to better solve mixed integer and integer problems. Hyperopt is an implementation of the Tree-structured Parzen Estimator (TPE) [3], which is also a BO algorithm. SMAC and Hyperopt have been reported to be able to handle problems with several tens of variables [2, 10, 11] and perform better than Gaussian Process based algorithms for discrete problems [8].

Model-based optimization (MBO) algorithms for discrete problems are increasingly a topic of great interest [1]. These algorithms rely on an accurate objective function approximation with a surrogate model. In a similar vein as BO, most algorithms generate one new solution per iteration. The surrogate models reported to perform the best are Radial Basis Function Networks (RBFN) [18], Kriging [25] adapted to binary variables, and a model based on Walsh functions decomposition [23]. However, the reported working examples for binary functions are still limited to moderate number of variables (up to 20-30) [1].

Another class of algorithms that can potentially be used for solving expensive optimization problems is surrogate-assisted Genetic Algorithms (GAs) [7]. GAs are powerful search methods, but potentially require many function evaluations. Using surrogate models is a natural way to reduce the number of real function evaluations by replacing a part of them with surrogate ones, which are considered to consume much less time. A surrogate-assisted GA therefore has the potential to be highly powerful for expensive optimization, retaining the competent search characteristics of the GA, while reducing overall run time through surrogate function evaluations. Note that a surrogate-assisted GA approach differs from using a GA in BO or MBO algorithms for finding the next point to evaluate. Common types of surrogate models integrated in GAs are polynomials, Kriging, RBFN, and Support Vector Regression (SVR) [7]. To the best of our knowledge, there is currently no literature on surrogate-assisted GAs for solving problems with discrete categorical variables. However, many real-world problems have discrete (binary) components, such as architectural choices, in the form of connections, of deep neural networks.

In this paper, we consider a novel type of surrogate model, namely a Convolutional Neural Network (CNN) [14]. Though there are existing works in which artificial neural networks are used [17], to our knowledge no works use a CNN as the surrogate model in a GA. Here, we study the best way to realize the integration and the resulting scalability of terms of required evaluations to solve well-known benchmark problems to optimality. We consider as the baseline the Gene Pool Optimal Mixing Evolutionary Algorithm (GOMEA) [5] so as to assess the true potential added value of a surrogate model by testing it in conjunction with a state-of-the-art GA. A key feature of GOMEA is its ability to exploit problem structure in the form of linkage (i.e., interdependent problem variables). This makes GOMEA highly suited to solve complex optimization problems in a BBO setting with excellent scalability.

2 CNN SURROGATE MODEL

We have chosen to use a CNN as the approximator due to its known capacity and extrapolation ability, as opposed to decision trees, for which output values are bounded by the values in the training dataset [9]. Compared to Fully-Connected neural networks, a CNN can learn the same information using much less parameters because filters are applied to different input data locations, essentially giving the CNN the power to learn higher-order combinations efficiently. Thus, the overfitting problem is less likely to occur.

2.1 Pairwise regression

We consider the task of fitness function approximation as a supervised machine learning regression problem. In general, the approximation procedure entails:

$$\min_{f} \; Loss\left([y_1, \ldots, y_N], [f(x_1), \ldots, f(x_N)]\right),$$

where f is an approximation function, $f : \{0,1\}^l \mapsto \mathbb{R}$; $x \in \{0,1\}^l$ are solutions, $y \in \mathbb{R}$ are fitness function values; *Loss* is a loss function, e.g., Mean Squared Error (MSE): $MSE(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$; N is the training dataset size. In machine learning terms, the solutions represent input features of a machine learning model and the fitness function values represent target values.

Despite the great potential of CNNs, a downside is that they need large numbers of data points for training. To alleviate this issue and make the most out of previously evaluated solutions, we propose to approximate the *difference* of fitness function values of two solutions rather than directly approximate the fitness of a solution. As input the CNN thus gets two solutions. As output it produces the predicted difference of their fitness values. The training dataset is formed by all possible *ordered* pairs of solutions. Thus, the size of the training dataset becomes n^2, with n the number of evaluated solutions, increasing the number of training samples by an order of magnitude. This effect is demonstrated in Figure 1 for the Onemax function described in Section 4.1 with $l = 100$. The pairwise regression needs less data points (i.e., evaluations of solutions) to produce accurate predictions on the validation set (solutions in both training and validation datasets are randomly generated in this example). The training procedure is described in Section 2.2.

After training it is possible to predict the fitness value of an arbitrary solution s by using the solutions from the training dataset $t_1, \ldots, t_N$, where N is the training dataset size, as pivotal solutions. Specifically, we form the pairs $(t_1, s), \ldots, (t_N, s), (s, t_1), \ldots, (s, t_N)$. For each pair the fitness difference is predicted and thereby the fitness value estimation is calculated. The resultant fitness value estimation of s is averaged over all pairs. Moreover, it is possible to use only part of the training solutions as pivotal ones to accelerate the prediction process. The full prediction procedure is described in Algorithm 1.

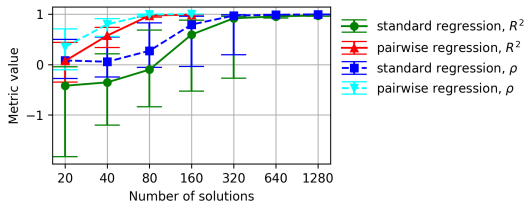

Figure 1: R^2 coefficient and Spearman correlation coefficient ρ for increasing numbers of evaluated solutions used as training samples for a CNN in case of direct regression and pairwise regression using the same internal CNN architecture. The approximated function is Onemax. For each number of solutions data generation and training are done 30 times. The dots represent the median values, the bars indicate the $2nd$ and $29th$ order statistics.

1: **function** PREDICTFITNESSVALUE(s)
2: *pivotalSolutions* ← subset of *trainData* of size *maxPivotalSize*
3: *predictions* ← ∅
4: **for** *refS* in *pivotalSolutions* **do**
5: *predictions* ← *predictions* ∪ {*refFitness* + modelPredict(*s, refS*)}
6: *predictions* ← *predictions* ∪ {*refFitness* - modelPredict(*refS, s*)}
7: *finalPrediction* ← mean(*predictions*)
8: **return** *finalPrediction*

Algorithm 1: Fitness prediction after pairwise regression.

2.2 Training procedure

Since we consider binary problems, we apply a standard simple data transformation before training: subtract 0.5 from input x to center it around 0 and divide target values y by the maximum absolute value seen in the training dataset to clip them between -1 and 1, as the initial weights of the CNN are generated randomly also from the $[-1; 1]$ interval.

We train the CNN with a hold-out validation set. For the division into training and validation sets, systematic sampling is used: the target values are sorted and every qth sample is taken into the validation set. In the experiments q is set to 5 (i.e., 80% of data is used for training, 20% for validation). After obtaining the training and validation sets, we form pairs of samples. Specifically, imitating the prediction procedure described in Section 2.1, the CNN is validated on pairs containing one solution from the training dataset, one from the validation one.

One training epoch consists of processing all pairs of samples from the training set. The loss that is minimized during training is the MSE. As a training algorithm we use standard backpropagation with the gradient-descent based optimizer Adam [12], with the following parameters: $\alpha = 0.0005, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$ (the standard parameters from [12], except the learning rate, which is slightly lowered to make the learning process more robust). An early stopping criterion is used for training: the training is stopped if the error on the validation pairs of solutions is not improving for several consecutive epochs. In our implementation this number of epochs is dependent on the training dataset size: it varies from 2 for large datasets of size $N \geq 10^5$ to 6 for smaller datasets of size $N < 10^4$ (for datasets of size $10^4 \leq N < 10^5$ it is set to 4).

2.3 CNN architecture

We would like the surrogate model to capture information about the impact of blocks of dependent, both closely and distantly located, variables on the fitness values. For this purpose dilated convolutional filters [24] can be used, which are capable of capturing dependencies between distant variables. They are also capable of capturing the dependencies between tightly located variables if the dilation factor, which determines the distance between variables to which the filter is applied, is set to 1. Each filter has the following hyperparameters: the filter size k that determines the number of variables captured by the filter at once, the stride st that determines the distance between successive moves of a filter along the input, the dilation factor d and weights w that are learned during the training procedure. Given a string of length l_{in}, it produces a string of length $l_{out} = \frac{l_{in}-d(k-1)-1}{st} + 1$. Usually, several filters with the same hyperparameters, but with independent weights are applied at once, each producing its own output. These filters with separately learned weights can capture different dependencies, or aspects thereof.

Another important feature of the proposed neural network architecture is the input representation in a form of two stacked solutions. This allows the convolutional filters to naturally capture the information about differences in corresponding genes of two solutions and its connection with differences in fitness values. The working principle of dilated filters in our context is demonstrated in Figure 2.

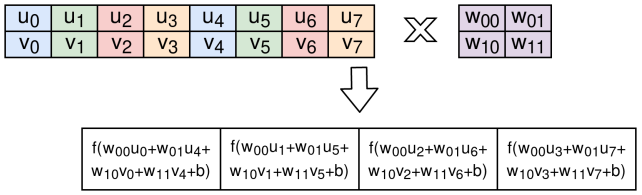

Figure 2: Example of applying a dilated filter to two stacked solutions u and v with 8 variables. Filter size $k = 2$, dilation factor $d = 4$, stride $st = 1$, w_{ij} are filters weights, b is a bias term, f is an activation function. Different colors represent the variables captured by the filter at each position while moving along the input. As a result, the filter produces a new string of length 4.

In this paper, we consider the convolutional neural network architecture with 2 convolutional layers, an optional fully-connected (dense) hidden layer, and an output layer, which is also fully connected. The architecture for the particular problem is optimized with the grid-search procedure [13]; more details are provided in Section 2.3.1. In the second convolutional layer, the filters are regular convolutional filters without dilation, because the first convolutional layer is supposed to already capture the dependencies between distant variables. We use C_1 and C_2 filters in the convolutional layers respectively. Moreover, the size of a fully-connected layer is C_3. The structure of the neural network is shown in Figure 3. The activation functions of all layers, except the output one are Rectified Linear Units (ReLUs): $f(x) = max(0, x)$; the output layer has a linear activation function: $f(x) = x$. After the second convolutional layer dropout [21] is applied with a common ratio of 0.2 to reduce model overfitting.

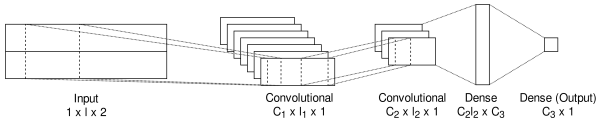

Figure 3: The CNN architecture used, with 1 hidden dense layer. l is the number of variables, l_1, l_2 are the second dimensions of tensors after processing by two convolutional layers. The input is two stacked solutions.

2.3.1 Hyperparameter Search. We perform an often-used grid search to find the best architecture and hyperparameters for a particular problem. In preliminary experiments, it was found that C_1, C_2, C_3 can be set to 100, 30, 30 respectively to provide a reasonable model complexity. These hyperparameters are not optimized during the grid search procedure.

The hyperparameters and their corresponding domains (search is performed over integer values) are presented in Table 1: k_1, st_1, d_1 are the filter size, stride size and dilation factor size of filters in the first convolutional layer respectively; k_2 is the filter size in the second convolutional layer, N_{fc} is the number of fully-connected layers (the hidden ones). The search is performed in two stages: in the first stage the hyperparameters of the first convolutional layer are optimized, the second layer is fixed with $k_2 = 1, st_2 = 1, d_2 = 1$, and N_{fc} is fixed at 1. In the second stage, the best found hyperparameters for the first layer are fixed and the search is performed

for k_2 and N_{fc} hyperparameters. The search is divided into two stages to accelerate it. We consider this division to be acceptable as the hyperparameters of the first layer have greater impact than the other ones on capturing the problem structure and hence on the CNN accuracy. The search is terminated in case the model quality (Spearman correlation on the validation set as described in Section 2.4) on a set of hyperparameters is ≥ 0.97.

Parameter	k_1	st_1	d_1	k_2	N_{fc}
Interval	$[1; min(10, \frac{l}{2})]$	$[1; \frac{l}{4}]$	$[1; \frac{f_1}{2}]$	$[1; 2f_1]$	[0,1]

Table 1: Hyperparameters and intervals of search.

2.4 Surrogate model quality

Common metrics for assessing the performance of machine learning algorithms for solving regression problems (e.g. MSE, R^2) can be applied to assess the quality of the surrogate model. However, when using a surrogate model in a GA we are more interested in consistency between solution ranks as ranked by the real fitness values and the surrogate values than in small differences between real and predicted fitness values. Thus, we define the surrogate model quality (in all further mentions) as the Spearman correlation coefficient: $r_s(x, y) = \frac{cov(rg_x, rg_y)}{\sigma_{rg_x} \sigma_{rg_y}}$, where x and y are respectively real and surrogate fitness values of solutions in a validation set, rg_x and rg_y are their ranks among the considered set of solutions.

3 CONVOLUTIONAL SURROGATE GOMEA

3.1 GOMEA

The key components of GOMEA are linkage learning, the Gene-pool Optimal Mixing (GOM) operator, the Forced Improvements (FI) procedure, and the Interleaved Multistart Scheme (IMS). We briefly outline each of these components here, and refer the interested reader to the literature for more details [5, 15, 22].

3.1.1 Linkage Learning. The goal of linkage learning algorithms is to reveal the nature of the linkage between variables, which can often be defined in terms of subsets of dependent variables. Multiple linkage learning algorithms and corresponding linkage structures have been introduced for GOMEA. In this paper we use the Linkage Tree (LT) [5] as the linkage structure because this structure has the ability to capture the structure of hierarchical problems and outperforms in most cases other linkage structures, as stated, e.g., in [22]. The structure obtained after linkage learning contains subsets of linked variables and is called the Family of Subsets (FOS).

3.1.2 Gene-pool Optimal Mixing. After learning the linkage structure, offspring solutions are generated. Offspring generation with the GOM starts with copying a parent solution. Then, the linkage structure is iterated over in a random order. For each subset, a donor solution is randomly selected from the population and genes from loci in the current linkage subset are copied from donor to offspring. The change is accepted only if the fitness function value does not deteriorate.

3.1.3 Forced Improvements. If the GOM operator has failed to improve the fitness of an offspring (after iterating over all the subsets in the linkage structure), or if the local elitist (of this population) solution has not improved for $log_{10}(populationSize)$ generations then the FI procedure is applied. The FI procedure performs mixing of the offspring with the global (taken over all populations) elitist solution. All subsets of the linkage structure are considered in a random order. For each linkage subset the genes from corresponding loci from the elitist solution are copied to the offspring; the change is accepted only in case of a real improvement of the fitness value. Once an improvement occurs, the FI procedure is immediately terminated. If the offspring has not improved even after this part of FI, it is set to the elitist solution.

3.1.4 Interleaved Multistart Scheme. The IMS aims to offer a reasonable alternative to needing to set the population size parameter by hand. Several populations of increasing sizes are run in an interleaved fashion. The smallest population in the scheme is of fixed size. Here, we use a population of size 8. The next populations are double the size of the previous population. The populations are interleaved, such that for each c^{IMS} generation iterations of a population of size $populationSize$, one iteration of the population with population size $2populationSize$ is performed. In our implementation, c^{IMS} is set to 4. A population is terminated if a population with a larger population size has a better average objective value.

3.2 Adding the surrogate model to GOMEA

The CNN described in Section 2 is used as a surrogate model in GOMEA. To do so, function evaluations are replaced by fitness value estimations made by the surrogate model under certain conditions. Moreover, each population in the IMS has its own surrogate model.

3.2.1 Preliminary Actions before IMS Loop. The general outline of CS-GOMEA is provided in Algorithm 2. Before starting to use the surrogate model, initial training data is collected. We call this process a *warm-up period*. After initial $warmupSize$ random solutions generation, a CNN hyperparameter search is performed. Then, more random solutions are generated until the model quality surpasses the threshold T. Re-training of the model is performed every time after $generationStep$ solutions are generated. In our experiments, we set $generationStep$ to 10, as a compromise between the number of model trainings and the number of evaluated solutions at each step, $warmupSize$ is adjusted with respect to the problem size and difficulty as described in Section 4.2.1.

If the surrogate model has failed to achieve a quality that is above the threshold T after a maximal allowed number of solution evaluations, then, probably, the predictive ability of this model will not be sufficient to find the optimum with GOMEA if surrogate fitness is used. Thus, a special algorithm mode called a mixed populations mode is then activated: a subset of the solutions in each population will evolve separately. In these subsets, only real evaluations are used, and, moreover, in generating offspring only solutions from these subsets are used as parents to avoid any bias in solutions induced by the surrogate model. The size of this subset of the population is determined by the model quality. The lower the quality is, the larger this subset of the population must be. Hence, we used $sizeMixed = \lceil populationSize * (T - modelQuality) \rceil$, but $sizeMixed$ is not allowed to be larger than half the population size. In preliminary experiments we have observed that it is reasonable to set T to 0.9.

```
1: function Main(warmupSize)
2:     generateRandomSolutions(warmupSize)
3:     performHyperparamatersSearch(warmupSize)
4:     while modelQuality < T and numEvaluated < 2warmupSize do
5:         generateRandomSolutions(generationStep)
6:         model ← trainModel(evaluatedArchive)
7:         modelQuality ← checkModelQuality(model)
8:     if modelQuality < T then
9:         mixedPopulationsMode ← true
10:    runIMS()
```

Algorithm 2: General outline of CS-GOMEA.

3.2.2 Real and Surrogate Fitness Values. As stated in other studies [4], mixing real and surrogate fitness values in a population might be an issue. This is because comparisons between real fitness values and surrogate ones are compromised as the surrogate model does not predict the fitness values with perfect accuracy. Thus, we store only surrogate fitness values, except when in mixed populations mode. In that case, the real fitness values are stored for the subset of the population that uses only real fitness values. As each population has its own surrogate model, the fitness values in a population are the estimates made by the model belonging to this population. Also, we maintain surrogate fitness values for each local elitist solution in each population. A global elitist solution for all populations is furthermore maintained, for which only the real fitness is stored.

3.2.3 CS-GOMEA Population Loop. The main loop of the optimization process in CS-GOMEA is described in Algorithm 3 [1].

```
1: function runPopulation(populationIndex)
2:     populationSize ← basePopulationSize*2^populationIndex
3:     P ← initializePopulation(populationSize)
4:     while population not terminated do
5:         learnFOS(P)
6:         generateOffspring(P)
7:         trainModel(randomArchive ∪ ⋃_{k=0}^{populationIndex} optArchive_k)
8:         updateSurrogateValues(P)
9:         for s ∈ P do
10:            doRealEvaluation(s)
```

Algorithm 3: Main CS-GOMEA loop.

One iteration over the population includes creating the offspring solutions, retraining the model, updating the surrogate values of the population, and performing real evaluations for offspring solutions to check whether there is an improvement of the elitist [2]. During the offspring generation described in Algorithm 4 there are several cases when a real evaluation can be executed. Here, we describe them in detail, along with underlying ideas.

1. *If the surrogate value of the solution is the new elitist surrogate value.* The reason for doing a real evaluation in this case is the assumed correlation between the model predictions and the real fitness values. Because of this, the new surrogate elitist has a good chance to be a real elitist solution too.
2. *If the real elitist solution has not been updated after populationSize offspring generations.* In this case we execute real evaluations more aggressively to find out whether any solution is the real new elitist. If the surrogate value of the solution is close to the elitist surrogate value, then a real evaluation is performed. The fitness distance of a solution with surrogate value v to the local elitist surrogate with value $v_{elitist}$ is determined by $\delta = \frac{v_{elitist} - lowerBound}{v - lowerBound}$, where $lowerBound$ is the minimal value found in the initial random set of solutions (after warm-up period). The correction by $lowerBound$ is required to make δ independent from the range of fitness values. The real evaluation is performed if $\delta < \delta_{threshold}$. During the preliminary experiments we have found that a good value for $\delta_{threshold}$ is 1.02.
3. *If the solution has not been changed both after the first stage of generating the offspring and the FI.* In this case, a real evaluation is performed, as it is expected that this solution, the surrogate value of which has not been improved during GOM and FI, has a high real fitness value as well.

3.2.4 Evaluated Solutions Archive. As the real evaluations of solutions are considered to be expensive, we maintain an archive of already evaluated solutions to avoid repeated evaluations. Each population maintains its own archive of solutions acquired by the steps of optimization algorithm ($optArchive_i$). The solutions acquired by the random generation are placed in the special archive $randomArchive$, which is shared among all populations.

Both types of archive are re-used during the re-training of the surrogate model. Using random solutions is important as they are distributed throughout the search space while the solutions obtained during the optimization process tend to be located in a certain part of the space. Specifically, in the kth population in the IMS for model training and validation the solutions from $randomArchive \cup optArchive_0 \cup \cdots \cup optArchive_k$ are used.

4 EXPERIMENTS

4.1 Optimization problems

We consider a set of well-known benchmark functions of binary variables. All functions need to be maximized.

The first function is Onemax, in which the variables are independent:

$$f_{Onemax}(x) = \sum_{i=0}^{l-1} x_i$$

The other functions on which we test the algorithms, have blocks of dependent variables. Two of such functions are the well-known deceptive traps [6] of order $k = 4$, with tight (dependent variables are located close to each other) and loose encoding (dependent variables are distant):

$$f_{TrapK-Tight}(x) = \sum_{i=0}^{l/k-1} f^{sub}_{TrapK}\left(\sum_{j=0}^{k-1} x_{ik+j}\right)$$

$$f_{TrapK-Loose}(x) = \sum_{i=0}^{l/k-1} f^{sub}_{TrapK}\left(\sum_{j=0}^{k-1} x_{i+(l/k)j}\right)$$

[1] In the IMS, no instance of GOMEA is run exactly as outlined in Algorithm 3. Rather, only c^{IMS} generations are done every time an instance is called upon (without re-initialization).

[2] Using the LT, one application of the GOM operation potentially requires $2l - 2$ evaluations, as this is the number of internal nodes in a LT, see [22]. Hence, evaluating all solutions fully at the end of a generational cycle requires a minor fraction of $\approx \frac{N}{N(2l-2)} = \frac{1}{2l-2}$ real evaluations (in non-mixed populations mode), thus potentially increasing the efficiency of the CNN integration with problem size.

```
1: function generateOffspring(P)
2:     if mixedPopulationsMode then
3:         realPopulationSize ← ⌈populationSize*(0.9-modelQuality)⌉
4:         if realPopulationSize > populationSize/2 then
5:             realPopulationSize ← populationSize/2
6:     for s in P do
7:         if mixedPopulationsMode and sIndex < realPopulationSize then
8:             offspring ← generateOffspringWithoutSurrogate(s)
9:         else
10:            offspring ← generateOffspringUsingSurrogate(s)
11:
12: function generateOffspringWithoutSurrogate(parent)
13:    offspring ← parent
14:    for subset ∈ FOS do
15:        donor ← random solution from P[0 . . . realPopulationSize-1]
16:        candidate ← generateSolution(offspring, subset, donor)
17:        doRealEvaluation(candidate)
18:        if change is accepted then
19:            offspring ← candidate
20:    if offspring not changed then
21:        performForcedImprovements()
22:    return offspring
23:
24: function generateOffspringUsingSurrogate(parent)
25:    offspring ← parent
26:    realEvaluationPerformed ← false
27:    for subset ∈ FOS do
28:        donor ← random solution from population
29:        candidate ← generateSolution(offspring, subset, donor)
30:        v ← predictFitnessValue(candidate)
31:        if realEvaluationPerformed = false then
32:            if v > v_elitist then
33:                v_elitist ← v                ▷ updating local surrogate elitist
34:                doRealEvaluation(candidate)
35:                realEvaluationPerformed ← true
36:            else
37:                if elitist not improved for populationSize offspring then
38:                    δ = (v_elitist − lowerBound) / (v − lowerBound)
39:                    if δ < δ_threshold then
40:                        doRealEvaluation(candidate)
41:                        realEvaluationPerformed ← true
42:        if change is accepted then
43:            offspring ← candidate
44:    if offspring not changed then
45:        performForcedImprovements(offspring)
46:    if offspring not changed then
47:        doRealEvaluation(offspring)
48:    return offspring
49:
50: function doRealEvaluation(s)
51:    r ← evaluate(s)
52:    if r > r_elitist then
53:        r_elitist ← r                        ▷ updating global real elitist
54:    add (s,r) to evaluated solutions archive
```

Algorithm 4: Offspring generation.

$$f^{sub}_{TrapK}(u) = \begin{cases} 1 & \text{if u=k} \\ \frac{k-1-u}{k} & \text{otherwise} \end{cases}$$

The other function with blocks of dependent variables is the Hierarchical If-And-Only-If (HIFF) function. It is considered to be more difficult as blocks of different sizes are presented (blocks hierarchy):

$$f_{Hiff}(x) = \sum_{k \in \{1,2,4 \ldots \frac{l}{2}, l\}} \sum_{i=0}^{l/k-1} f^{sub}_{Hiff}(x_{ik \ldots (i+1)k-1})$$

$$f^{sub}_{Hiff}(u) = \begin{cases} 1 & \text{if } \sum_{j=0}^{k} u_j = k \text{ or } \sum_{j=0}^{k} u_j = 0 \\ 0 & \text{otherwise} \end{cases}$$

The final function that we consider are the NK-landscapes with maximum overlap and blocks of $k = 5$ [19]. This function contains blocks of dependent variables, but with fitness values of subfunctions depending on the block position:

$$f_{NK-S1}(x) = \sum_{i=0}^{l-k} f^{sub}_{NK-S1}(x_{(i,i+1,\ldots,i+k-1)})$$

where the values of $f^{sub}_{NK-S1}(x_{(i,i+1,\ldots,i+k-1)})$ are tabular values, generated randomly in interval $[0; 1]$.

4.2 Experimental setup

The proposed CS-GOMEA is compared with GOMEA, SMAC and Hyperopt. In order to make the comparison fair, in all algorithms we count evaluations of unique solutions only. We consider the problems described in Section 4.1 with the following problem sizes: for Trap4-Tight, Trap4-Loose, and HIFF problems $l \in \{8, 16, 32, 64, 128, 256\}$, for Onemax $l \in \{25, 50, 100, 200, 400\}$, for NK-landscapes $l \in \{25, 50\}$. We run SMAC and Hyperopt for Trap4 and HIFF problems for $l \in \{8, 16\}$, for Onemax $l = 25$, for NK-landscapes $l = 25$. We do not find it reasonable to run SMAC and Hyperopt for larger problem sizes as they fail to find an optimum in some runs even for the considered problem sizes. For each problem instance we perform 30 runs (for CS-GOMEA each run is a full optimization procedure including the neural network hyperparameter search).

The stopping criterion for GOMEA and CS-GOMEA was either finding an optimal solution or exceeding a time limit. The time limit for most problems was set to 4 hours. For NK-S1 the time limit was set to 6 hours. For Trap4 and HIFF problem instances with $l = 256$, the time limit was set to 8 hours. The time limit also determines the maximum allowed time for the CNN hyperparameter search which is not allowed to consume more than half of the overall computational budget. For SMAC and Hyperopt the stopping criteria was executing 10000 function evaluations, except for NK-S1, for which it was 20000 evaluations.

We study the scalability of the algorithms, i.e., the relation between the number of required evaluations to find the optimum and the problem size. The results are presented in Figure 4. Only the runs with the optimum achieved are reported. Tables with experimental results can be found in Supplementary material.

Besides scalability, we analyze convergence speed, i.e. whether the elitist solution is being gradually improved or the algorithm rapidly finds a solution close to the optimum (what is expected when an accurate surrogate model is used). Convergence behaviour

l	8	16	32	64	128	256
Trap4-Tight	100	100	100	150	200	300
Trap4-Loose	100	100	100	150	200	300
HIFF	100	100	100	150	200	300
l	**25**	**50**	**100**	**200**	**400**	
Onemax	100	100	100	150	200	
NK-S1	200	200				

Table 2: Values of *warmupSize* hyperparameter.

analysis might be important for practical usage in real-world expensive optimization tasks as it might be sufficient to quickly find a good quality solution. Convergence plots are presented in Figure 5.

4.2.1 warmupSize Hyperparameter. We adjust the value of the *warmupSize* hyperparameter with respect to the problem size (Table 2), starting with $warmupSize = 100$ for moderate dimensionalities. For larger dimensionalities doubling the problem dimensionality is accompanied by approximately doubling the training dataset size. For practical usage it means that we can start with a moderate *warmupSize* (e.g., 100) and gradually increase it until the fitness of the found elitist solution stops improving.

4.2.2 Implementation and Experimental Environment Details. The main algorithm code is written in C with Python function calls. Neural network operations are implemented in Python using the Pytorch framework. The experiments are run on an Intel Xeon E5-2630 CPU core and an Nvidia Titan X (Pascal architecture) GPU. Code is available at https://github.com/ArkadiyD/CS-GOMEA.

4.3 Results

For Onemax, Trap4-Tight, Trap-Loose, HIFF and NK-S1 problems CS-GOMEA was able to find an optimal solution in all executed runs for all considered problem sizes. For the first four problems the number of evaluations required by CS-GOMEA scales substantially better than for GOMEA. According to our calculations, the median minimal function evaluation time, that makes CS-GOMEA usage reasonable (i.e., decreasing overall run time compared to GOMEA) is ≈ 5 seconds for all problems with $l \leq 16$ and ≈ 1 second for problems with $l > 16$. Such evaluation time is well below the function evaluation time in various real-world optimization problems, especially in the deep learning field.

The comparison with SMAC and Hyperopt shows that they are competitive with CS-GOMEA only for low-dimensional problems. For dimensionality 8, SMAC found the optimum in all runs for Trap4 and HIFF problems. For dimensionality 16, the optimum was not found in all runs. Hyperopt failed to find the optimum in some cases, even for dimensionality 8. For dimensionality 25 for Onemax and NK-S1, both of the algorithms have failed to find the optimal solution in some runs. Moreover, as shown in Figure 5, their convergence speed is lower than for CS-GOMEA.

The NK-landscapes function turned out to be the most difficult for the surrogate model to approximate. The achieved model quality seems to be insufficient to solve the problem using predominantly the surrogate model. To be able to achieve the optimum, the mixed populations mode is required. In this mode, the number of evaluations grows significantly, but still CS-GOMEA finds the optimum with less evaluations than GOMEA.

4.4 Statistical Tests

To verify the significance of the results, we perform statistical tests. As the normality of the distribution of the required number of evaluations to achieve an optimum cannot be assumed, we perform the Mann-Whitney U test [16]. For each problem and each dimensionality we perform a statistical test, testing that the number of evaluations to achieve the optimum by GOMEA is larger than by CS-GOMEA. The considered level of statistical significance is $\alpha = 0.05$. Bonferroni correction is applied. According to the conducted experiments, CS-GOMEA requires statistically significant less number of evaluations than GOMEA for Onemax, Trap4-Tight, Trap4-Loose for all considered values of l. For HIFF with $l \geq 16$.

5 DISCUSSION

The proposed type of surrogate model demonstrated high approximation accuracy for several functions with blocks of dependent variables, including the HIFF problem. NK-landscapes are however difficult to be approximated by the proposed CNN due to the CNN main principle of assuming the same subfunctions occuring in different solution locations (the convolutional filters weights are re-used in the convolutions) while in NK-landscapes such subfunctions are random and thus different everywhere. In the current paper we do not in any way exploit the linkage information captured in the LT in surrogate modeling. Rather, we focused on providing a baseline proof-of-principle that a CNN-based surrogate assisted GOMEA has merit. However, we believe linkage integration is an important research question for future work, i.e., taking into account the subsets of dependent variables in neural network architecture design instead of moving the filters along the input for all variables. This may well be highly beneficial for model quality on such difficult non-regular problems as NK-S1.

6 CONCLUSIONS

We have introduced a novel surrogate-assisted GA for solving expensive discrete (boolean) optimization problems. The key novel features of our algorithm are keeping the strengths of the GOMEA algorithm while using a convolutional neural network (CNN) as a surrogate model with a pairwise regression approach for model training to be able to train the CNN with small numbers of samples.

Experiments on a well-known set of benchmark functions show that the proposed integration requires much less function evaluations than GOMEA to reach the optimum and scales much better than GOMEA. The small number of solutions required to find the optimum is promising for the ultimate goal of using the algorithm for expensive (real-world) optimization problems, which is part of our near-future research.

The comparison with the existing Bayesian-optimization based, expensive discrete algorithms SMAC and Hyperopt showed that using the proposed surrogate-assisted GA is more promising for the considered type of optimization problems as, firstly, CS-GOMEA is more scalable and can solve problems with much larger problem sizes, and, secondly, even for small problem sizes CS-GOMEA requires less numbers of evaluations to achieve the optimum.

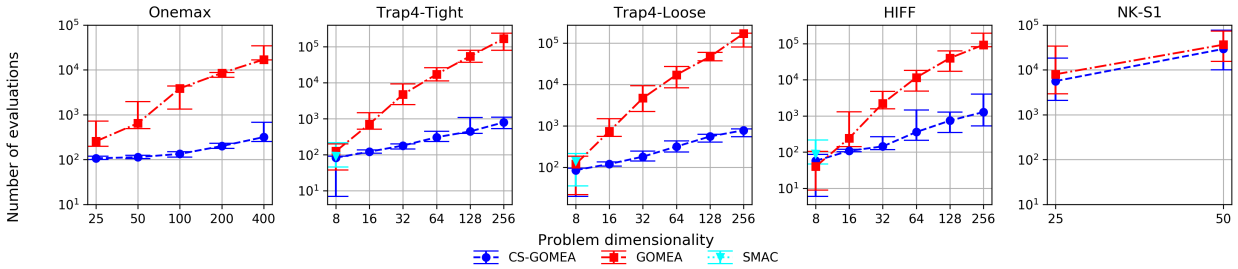

Figure 4: Scalability of considered algorithms in terms of required number of evaluations to reach the optimum, summarized over 30 runs. The algorithms are presented only for the problem instances in which an optimum was achieved in all 30 runs. There are no such problem instances for Hyperopt. The bars show the the intervals of required number of evaluations excluding the lowest and highest values (what corresponds to ≈ 93% interval). The points indicate the median values.

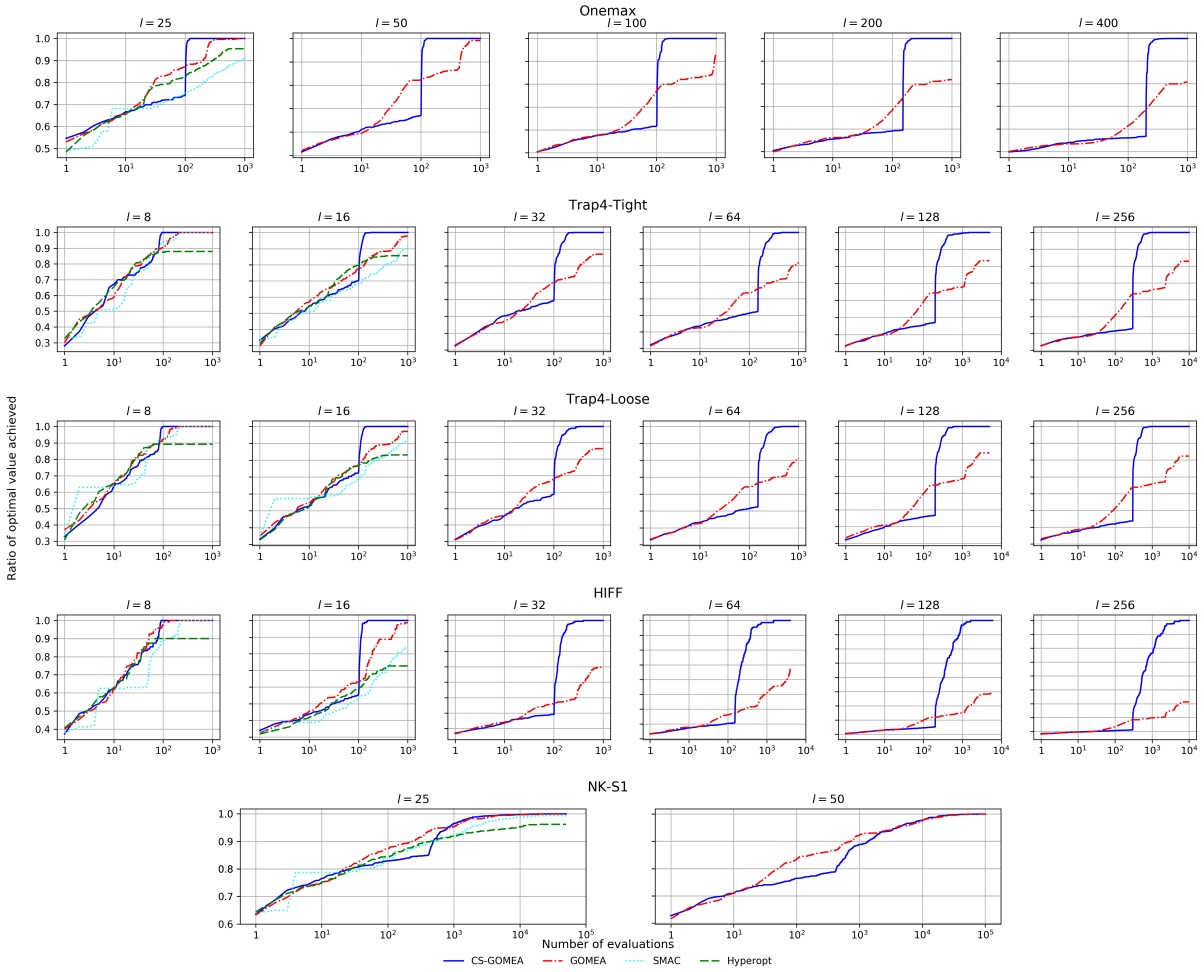

Figure 5: Convergence of all considered algorithms for all considered problems in terms of ratio to optimal value. For each problem instance and for every number of evaluations the fitness value of an elitist solution is averaged over 30 runs. The ranges for the horizontal axis are chosen dependent on problem size and CS-GOMEA performance.

ACKNOWLEDGMENTS

This work is part of the research project FEDMix funded by the Netherlands Organization for Scientific Research (NWO) (project number 628.011.012). We gratefully acknowledge the support of NVIDIA Corporation with providing Titan X GPU cards used for this research.

REFERENCES

[1] Thomas Bartz-Beielstein and Martin Zaefferer. 2017. Model-based methods for continuous and discrete global optimization. *Applied Soft Computing* 55 (2017), 154–167.

[2] James Bergstra, Brent Komer, Chris Eliasmith, Dan Yamins, and David D. Cox. 2015. Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery* 8, 1 (2015), 014008.

[3] James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. In *Advances in proceedings of Neural Information Processing Systems (NIPS 2011)*. 2546–2554.

[4] Kalyan Shankar Bhattacharjee, Hemant Kumar Singh, Tapabrata Ray, and Juergen Branke. 2016. Multiple surrogate assisted multiobjective optimization using improved pre-selection. In *Evolutionary Computation (CEC), 2016 IEEE Congress on*. IEEE, 4328–4335.

[5] Peter A.N. Bosman and Dirk Thierens. 2012. Linkage neighbors, optimal mixing and forced improvements in genetic algorithms. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation (GECCO 2012)*. ACM, 585–592.

[6] Kalyanmoy Deb and David E. Goldberg. 1994. Sufficient conditions for deceptive and easy binary functions. *Annals of Mathematics and Artificial Intelligence* 10, 4 (1994), 385–408.

[7] Alan Díaz-Manríquez, Gregorio Toscano, Jose Hugo Barron-Zambrano, and Edgar Tello-Leal. 2016. A review of surrogate assisted multiobjective evolutionary algorithms. *Computational Intelligence and Neuroscience* 2016 (2016).

[8] Katharina Eggensperger, Matthias Feurer, Frank Hutter, James Bergstra, Jasper Snoek, Holger Hoos, and Kevin Leyton-Brown. 2013. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *NIPS workshop on Bayesian Optimization in Theory and Practice*, Vol. 10. 3.

[9] Tomislav Hengl, Madlene Nussbaum, Marvin N. Wright, Gerard B.M. Heuvelink, and Benedikt Gräler. 2018. Random forest as a generic framework for predictive modeling of spatial and spatio-temporal variables. *PeerJ* 6 (2018), e5518.

[10] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. 2013. An evaluation of sequential model-based optimization for expensive blackbox functions. In *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO 2013)*. ACM, 1209–1216.

[11] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*. Springer, 507–523.

[12] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[13] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. 2007. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th International Conference on Machine Learning (ICML 2007)*. ACM, 473–480.

[14] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. 1999. Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*. Springer, 319–345.

[15] Ngoc Hoang Luong, Han La Poutré, and Peter A.N. Bosman. 2018. Multi-objective Gene-pool Optimal Mixing Evolutionary Algorithm with the Interleaved Multi-start Scheme. *Swarm and Evolutionary Computation* 40 (2018), 238–254.

[16] Henry B. Mann and Donald R. Whitney. 1947. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics* (1947), 50–60.

[17] Andrea Massaro and Ernesto Benini. 2015. A surrogate-assisted evolutionary algorithm based on the genetic diversity objective. *Applied Soft Computing* 36 (2015), 87–100.

[18] Alberto Moraglio and Ahmed Kattan. 2011. Geometric generalisation of surrogate model based optimisation to combinatorial spaces. In *European Conference on Evolutionary Computation in Combinatorial Optimization*. Springer, 142–154.

[19] Martin Pelikan, Kumara Sastry, David E. Goldberg, Martin V. Butz, and Mark Hauschild. 2009. Performance of evolutionary algorithms on NK landscapes with nearest neighbor interactions and tunable overlap. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO 2009)*. ACM, 851–858.

[20] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando De Freitas. 2016. Taking the human out of the loop: A review of bayesian optimization. *Proc. IEEE* 104, 1 (2016), 148–175.

[21] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

[22] Dirk Thierens and Peter A.N. Bosman. 2011. Optimal mixing evolutionary algorithms. In *Proceedings of the 13th annual conference on Genetic and Evolutionary Computation (GECCO 2011)*. ACM, 617–624.

[23] Sébastien Verel, Bilel Derbel, Arnaud Liefooghe, Hernan Aguirre, and Kiyoshi Tanaka. 2018. A surrogate model based on Walsh decomposition for pseudo-boolean functions. In *International Conference on Parallel Problem Solving from Nature*. Springer, 181–193.

[24] Fisher Yu and Vladlen Koltun. 2015. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122* (2015).

[25] Martin Zaefferer, Jörg Stork, Martina Friese, Andreas Fischbach, Boris Naujoks, and Thomas Bartz-Beielstein. 2014. Efficient global optimization for combinatorial problems. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. ACM, 871–878.