OPT-GAN: A Broad-Spectrum Global Optimizer for Black-box Problems by Learning Distribution Appendix

A General Optimization Framework

The general optimization framework can be summarized as Algorithm 1. The optimizer maintains explicit/implicit models to obtain a conditional distribution $p(\boldsymbol{x}|\mathcal{D}_t)$, where $\mathcal{D}_t$ is the historical information base till t^{th} epoch, storing historical solutions, corresponding fitness, and other information acquired in optimization. In order to facilitate description, we simplify $p(\boldsymbol{x}|\mathcal{D}_t)$ as $p(\boldsymbol{x})$ here. Thus, the optimization process is looping among sampling from $p(\boldsymbol{x})$, evaluating samples, and updating historical information base $\mathcal{D}$. The major differences in various optimizers lie in designs of p.

Algorithm 1 General Optimization Framework

Initialize historical information base $\mathcal{D}_0$.
for epoch t= 0, 1, 2, ... **do**
  1) Sampling new independent solutions from the $p(\boldsymbol{x}|\mathcal{D}_t)$;
  2) Evaluating the new sampling solutions on f;
  3) Updating the historical information base $\mathcal{D}_{t+1}$;
  4) Updating the distribution $p(\boldsymbol{x}|\mathcal{D}_{t+1})$;
end for

There are multiple ways to implement this general framework. For model-building algorithms, Bayesian Optimization obtains $p(\boldsymbol{x})$ by modeling the problem landscape, which leverages $q(f|\mathcal{D})$ as its distribution model of surrogate, i.e. Gaussian process (GP). The parameters of GP are updated by $\mathcal{D}$ at each epoch. CMA-ES, an Estimation of Distribution Algorithm (EDA) method, directly models $p(\boldsymbol{x})$ as a multivariate Gaussian distribution, which parameters, i.e., mean value and covariance matrix, are incrementally updated by newly sampled solutions from $\mathcal{D}$. As a gradient estimation-based algorithm, Explicit Gradient Learning (EGL) manipulates $p(\boldsymbol{x})$ by learning ∇f to generate solutions. The ∇f is estimated by a neural network using historical best solutions and corresponding fitness.

Model-free algorithms do not explicitly maintain a model but perform the search process by manipulating solutions directly. For example, PSO decides the upcoming sampling region by an implicit $p(\boldsymbol{x})$, i.e., $\boldsymbol{x}$ is directly generated using previous velocities, positions, and historical best positions.

In terms of optimizer selection, multiple factors can affect the selection of optimization algorithms when solving a specific problem, such as the trade-off among *quality of solution*, *evaluation cost*, and *computation cost*. The *quality of solution* indicates how good the final solution is, i.e., fitness function value. The *evaluation cost* refers to the cost, e.g., time consumption or price, for evaluating candidate solutions, and the *computation cost* refers to the cost for generating new solutions, consisting of updating historical information base, updating models, and sampling new independent solutions.

Example: We give the Estimation of Distribution Algorithm (EDA) as a concrete example to show how it solves an optimization problem and how it is related to the general framework in Algorithm 1. EDAs progressively approach global optima by manipulating the estimated distribution of promising solutions, which has received extensive successful stories as a well-known black-box optimization

algorithm family[1, 2, 3, 4]. In EDA, $\mathcal{D}$ corresponds to the good population set, whose functionality is similar with $\boldsymbol{x}_{\text{opt}}$ in Algorithm 2. It constructs $p(\boldsymbol{x})$ based on $\mathcal{D}$ with a parametric distribution model, e.g., the Gaussian model. Note that its $p(\boldsymbol{x})$ only incorporates $h(\boldsymbol{x})$ without considering the explorative role of $u(\boldsymbol{x})$ (see Eq. 2 in paper), which easily leads to over exploitation, especially for continuous EDA methods in solving black-box problems. Although EDAs have been widely applied in many real-world problems, thorough theoretical analysis are still difficult to be conducted since these methods are mainly used to deal with black-box optimization problems [5, 6, 7].

B Weakness of GAN-based Solver

The GAN-based solver could search original space directly for arbitrary black-box problems, and is a pioneer of GAN-based optimization. However, it actually has many weaknesses.

- It is almost a *local optimizer*, as it only pays attention to the exploitation by gradually dividing the searching space, but loses the ability of jumping out of local optimum, to say nothing of balancing exploration-exploitation (E-E) trade-off.
- The use of fixed size and binary knowledge base cannot guide the generator to smoothly tune the distributions on the search domain.
- Learning the distribution accurately is difficult because of the mode collapse, which also results in premature convergence.
- Considering its nature of exploitatory searching and arbitrary initialization, the success of optimization is largely dependent on the intrinsic distribution of the initial generator.
- Only a couple of problems are used for verifying the efficacy, lacking of extensive experiments to demonstrate the features of GAN-based optimization.

C Methodology

Table 1 lists the notations used in the Methodology. Algorithm 2 is the full version of OPT-GAN's pseudocode. It describes how OPT-GAN is used to optimize a black-box problem. Fig 1 visualizes the influence of a on K.

Table 1: Description of notations.

Notation	Description	Notation	Description
n	The dimensionality of a benchmark.	G	The solution generator.
fitness	The value of f.	D	The bi-discriminators.
$U(\Omega)$	The uniform distribution on domain Ω.	D_I	The exploitation discriminator.
$\boldsymbol{x}_{\text{uni}}$	The solution set sampled from $U(\Omega)$.	D_R	The exploration discriminator.
$\boldsymbol{x}_{\text{samp}}$	The solution set sampled from $\boldsymbol{x}_{\text{opt}}$ by bootstrap method.	$\boldsymbol{\omega_G}$, $\boldsymbol{\omega_I}$, $\boldsymbol{\omega_R}$	The network parameter of G , D_I, D_R.
$\boldsymbol{x}_{\text{opt}}$	The historical best solution set.	$PreIter$	The number of generator pre-training iterations.
$\boldsymbol{x}_{\text{G}}$	The solution set generated by G.	$GANIter$	The number of training iterations of GAN.
M	The population size of $\boldsymbol{x}_{\text{G}}$ when updating.	$DIter$	The number of training iterations of D.
K	The size of $\boldsymbol{x}_{\text{opt}}$.	β	The gradient penalty factor.
a	The shrinking rate.	λ	The adjustment factor for exploration-exploitation trade-off.
FEs	The number of function evaluations.	S	The batch size when training GAN.
$MAXFes$	The maximum of FEs.	$\hat{\boldsymbol{x}}_{\boldsymbol{I}}$	It is sampled uniformly along straight lines between $\boldsymbol{x}_{\text{G}}$ and $\boldsymbol{x}_{\text{samp}}$.
$\boldsymbol{\eta}$	The random noise.	$\hat{\boldsymbol{x}}_{\boldsymbol{R}}$	It is sampled uniformly along straight lines between $\boldsymbol{x}_{\text{G}}$ and $\boldsymbol{x}_{\text{uni}}$.

Algorithm 2 OPT-GAN

Input: The size of optimal set is K, the adjustment factor λ, the shrinking rate a, the batch size is S, the population size for updating optimal set is M, the maximum number of fitness evaluations is $MAXFes$, the number of training iterations to GAN and $D = \{D_I, D_R\}$ are $GANIter$ and $DIter$ respectively, the number of iterations of generator pre-training $PreIter$.

Init:

Initialize $\boldsymbol{\omega_I}$, $\boldsymbol{\omega_R}$ for D_I, D_R, and $\boldsymbol{\omega_G}$ for G, respectively.

$\boldsymbol{x}_{\text{opt}} = \{(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(K)}) | \boldsymbol{x}^{(s)} \sim U(\Omega),\ s = 1, 2, \cdots, K\}$; $MaxEpoch = \lceil (MAXFes - K)/M \rceil$

$t = K$; $K^{(0)} = K$

Generator Pre-training:

for $iterP = 1$ **to** $PreIter$ **do**
  for $iterG = 1$ **to** $GANIter$ **do**
    for $iterD = 1$ **to** $DIter$ **do**
      $\boldsymbol{x}_{\text{uni}} = \{(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(S)}) \mid \boldsymbol{x}^{(s)} \sim U(\Omega),\ s = 1, 2, \cdots, S\}$
      $\boldsymbol{x}_{\text{G}} = G_{\boldsymbol{\omega_G}}(\boldsymbol{\eta})$
      $\hat{\boldsymbol{x_R}} = \boldsymbol{\xi} \cdot \boldsymbol{x}_{\text{G}} + (1 - \boldsymbol{\xi}) \cdot \boldsymbol{x}_{\text{uni}},\ \boldsymbol{\xi} = \{(\boldsymbol{\xi}^{(1)}, \ldots, \boldsymbol{\xi}^{(S)}) \mid \boldsymbol{\xi}^{(s)} \sim U(\boldsymbol{0}, \boldsymbol{1}),\ s = 1, 2, \cdots, S\}$
      $\boldsymbol{\omega_R} = \arg\min_{\boldsymbol{\omega_R}} [\frac{1}{S} \sum_{s=1}^{S} (D_{R\omega_R}(\boldsymbol{x}_{\text{G}}^{(s)}) - D_{R\omega_R}(\boldsymbol{x}_{\text{uni}}^{(s)}) + \beta(\left\| \nabla_{\hat{\text{x}}_{\text{R}}} D_{R\omega_R}(\hat{\boldsymbol{x_R}}^{(s)}) \right\|_2 - 1)^2)]$
    end for
    $\boldsymbol{x}_{\text{G}} = G_{\boldsymbol{\omega_G}}(\boldsymbol{\eta})$
    $\boldsymbol{\omega_G} = \arg\min_{\boldsymbol{\omega_G}} [-\frac{1}{S} \sum_{s=1}^{S} D_R(\boldsymbol{x}_{\text{G}}^{(s)})]$
  end for
end for

Distribution Reshaping:

for $epoch = 1$ **to** $MaxEpoch$ **do**
  for $iterG = 1$ **to** $GANIter$ **do**
    Training Discriminators D_I **and** D_R**:**
    for $iterD = 1$ **to** $DIter$ **do**
      $\boldsymbol{x}_{\text{samp}} = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(S)}\}$, sampled from $\boldsymbol{x}_{\text{opt}}$ by bootstrapping method.
      $\boldsymbol{x}_{\text{uni}} = \{(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(S)}) | \boldsymbol{x}^{(s)} \sim U(\Omega),\ s = 1, 2, \cdots, S\}$
      $\boldsymbol{x}_{\text{G}} = G_{\boldsymbol{\omega_G}}(\boldsymbol{\eta})$
      $\hat{\boldsymbol{x_I}} = \boldsymbol{\xi} \cdot \boldsymbol{x}_{\text{G}} + (1 - \boldsymbol{\xi}) \cdot \boldsymbol{x}_{\text{samp}},\ \boldsymbol{\xi} = \{(\boldsymbol{\xi}^{(1)}, \ldots, \boldsymbol{\xi}^{(S)}) \mid \boldsymbol{\xi}^{(s)} \sim U(\boldsymbol{0}, \boldsymbol{1}),\ s = 1, 2, \cdots, S\}$
      $\hat{\boldsymbol{x_R}} = \boldsymbol{\xi} \cdot \boldsymbol{x}_{\text{G}} + (1 - \boldsymbol{\xi}) \cdot \boldsymbol{x}_{\text{uni}},\ \boldsymbol{\xi} = \{(\boldsymbol{\xi}^{(1)}, \ldots, \boldsymbol{\xi}^{(S)}) \mid \boldsymbol{\xi}^{(s)} \sim U(\boldsymbol{0}, \boldsymbol{1}),\ s = 1, 2, \cdots, S\}$
      $\boldsymbol{\omega_I} = \arg\min_{\boldsymbol{\omega_I}} [\frac{1}{S} \sum_{s=1}^{S} (D_{I\omega_I}(\boldsymbol{x}_{\text{G}}^{(s)}) - D_{I\omega_I}(\boldsymbol{x}_{\text{samp}}^{(s)}) + \beta(\left\| \nabla_{\hat{\text{x}}_{\text{I}}} D_{I\omega_I}(\hat{\boldsymbol{x_I}}^{(s)}) \right\|_2 - 1)^2)]$
      $\boldsymbol{\omega_R} = \arg\min_{\boldsymbol{\omega_R}} [\frac{1}{S} \sum_{s=1}^{S} (D_{R\omega_R}(\boldsymbol{x}_{\text{G}}^{(s)}) - D_{R\omega_R}(\boldsymbol{x}_{\text{uni}}^{(s)}) + \beta(\left\| \nabla_{\hat{\text{x}}_{\text{R}}} D_{R\omega_R}(\hat{\boldsymbol{x_R}}^{(s)}) \right\|_2 - 1)^2)]$
    end for
    Training Solution Generator G**:**
    $\boldsymbol{x}_{\text{G}} = G_{\boldsymbol{\omega_G}}(\boldsymbol{\eta})$
    $\boldsymbol{\omega_G} = \arg\min_{\boldsymbol{\omega_G}} [-\frac{1}{S} \sum_{s=1}^{S} (\frac{1}{1+\lambda} D_{I\omega_I}(\boldsymbol{x}_{\text{G}}^{(s)}) + \frac{\lambda}{1+\lambda} D_{R\omega_R}(\boldsymbol{x}_{\text{G}}^{(s)}))]$
  end for
  Updating:
  $\boldsymbol{x}_{\text{G}} = G_{\boldsymbol{\omega_G}}(\boldsymbol{\eta})$
  $\boldsymbol{B} = \boldsymbol{x}_{\text{opt}} \cup \boldsymbol{x}_{\text{G}}$.
  Sort $\boldsymbol{B}$ according to their $f(\boldsymbol{x}_{\text{B}})$ in ascending pattern.
  $t = t + M$
  Shrinking:
  $K^{(t)} = \lceil K^{(0)}(1 - a \cdot (t/MAXFes)) \rceil$
  $\boldsymbol{x}_{\text{opt}} = \boldsymbol{B}^{(1:K^{(t)})}$
end for
Return: $\arg\min f(\boldsymbol{x}_{\text{opt}})$

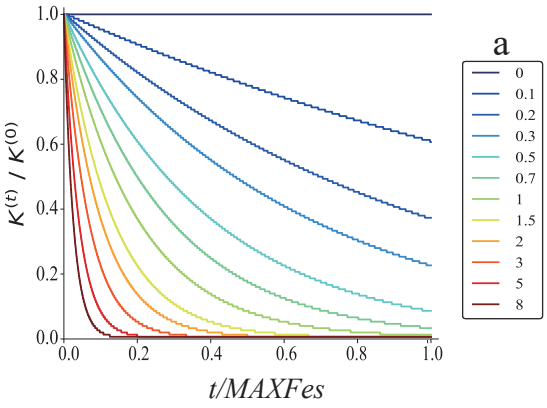

Figure 1: Shrinking process of K based on different a.

D Experiments

D.1 Experiment Setting

Benchmark Settings In the experiments, challenging black-box benchmarks are employed from the COCO platform [8], CEC'19 Benchmark Suite [9], Conformal Bent Cigar [10], and Simulationlib [11]. The information of these benchmarks is given in Table 2.

Architecture of OPT-GAN In experiment, the architectures of D_I, D_R, and G of OPT-GAN are multi-layer perceptrons (MLP) with single hidden layer. The input noise of G is drawn from a multivariate uniform distribution $U[\mathbf{-1}, \mathbf{1}]$ with dimensionality $2n$, where n is the benchmark's dimensionality. The number of neurons in the hidden layer is 50 for the generator and discriminators. LeakyRelu is served as the activation function.

Table 2: The information of benchmarks

	Benchmarks Name	Search Domain Ω
Platform for Comparing Continuous Optimizers (COCO)	Sphere (F1), Ellipsoidal (F2), Rastrigin (F3), Buche-Rastrigin (F4), Linear Slope (F5), Attractive Sector (F6), Step Ellipsoidal (F7), Original Rosenbrock (F8), Rotated Rosenbrock (F9), non-separable Ellipsoida (F10), Discus (F11), Bent Cigar (F12), Sharp Ridge (F13), Different Powers (F14), non-separable Rastrigin (F15), Weierstrass (F16), Schaffers F7 (F17), Schaffers F7 ill-conditioned (F18), Griewank-Rosenbrock (F19), Schwefel (F20), Gallagher's Gaussian 101-me Peaks (F21), Gallagher's Gaussian 21-hi Peaks (F22), Katsuura (F23), Lunacek bi-Rastrigin (F24).	$[-5, 5]^n$
CEC'19 Benchmark Suite	Shifted and Rotated Schwefel, Shifted and Rotated Expanded Schaffer F6, Shifted and Rotated Weierstrass.	$[-100, 100]^n$
Simulationlib	Michalewicz, Sim: Sphere, Sim: Rastrigin, Sim: Rosenbrock.	$[0, 4]^n$ $[-5.12, 5.12]^n$
Conformal Bent Cigar	Conformal Bent Cigar.	$[-5, 5]^n$

Please note that the architecture of the generator and discriminators in the experiment is by no means

the best implementation of OPT-GAN. However, it is one of the most straightforward implementations to verify the framework's effectiveness, eliminating the interference of architectural tricks to evaluation results.

Hyperparameter Settings The main hyperparameter settings of these optimizers are described as follow. For a fair comparison, the hyperparameters of OPT-GAN and GAN-based Solver (GBS) are obtained by trial and error. We take the recommendations [12] for BFGS, Nelder-Mead method (NM), CMA-ES, and Explicit Gradient Learning (EGL). For PSO, the settings of hyperparameters follow the work [13]. For Weighted Retraining (WR), the hyperparameters are set according to the Ref.[14]. For a fair comparison, we also provide BoRisk-pop30, a variation of BoRisk with identical configurations to it [15], except for the candidate set size and the initial set size. The detailed hyperparameter settings of OPT-GAN and opponents are described in Table 3, respectively. Python is used to code these optimizers. Meanwhile, in order to maintain the fairness of all the experiments, we ran all algorithms on the CPUs, even though the NN-based methods can be accelerated on the high-performance platform.

The indicator, $fbest - f(\boldsymbol{x}^*) - prec$ [8], is used to record the fitness of optimizers, where $fbest$ is the historical best fitness, and $prec = 10^{-8}$ denotes the predefined precision level. The lower fitness on the benchmarks indicates the better performance of optimizers.Unless otherwise noted, each optimizer runs 15 repeated trials on each problem. When $n = 2$, the $MaxFes$ is 50000. On the other dimensions, the $MaxFes$ is 100000. A single run stops if: (1) fitness reaches the precision ($fbest - f(\boldsymbol{x}^*) - prec < 0$), (2) FEs arrives at the maximum number, or (3) the elapsed time exceeds 3 hours.

The proportion of trials (PT) of empirical cumulative distribution function(ECDF[1]) for an optimizer equals the number of trials whose fitness reaches the precision ($fbest - f^* > 10^1$, i.e. successful) divided by the total number of trials.

D.2 Ablation Analysis

To examine the necessity of different components of OPT-GAN, we conduct the ablation experiments for exploitation discriminator D_I, exploration discriminator D_R, shrinking strategy, generator pre-training, and bi-discriminators component. Fig. 2 visualizes the distribution learned by the generator G during optimization in the way of the heatmap. No-D_I OPT-GAN is an OPT-GAN without the exploitation discriminator D_I; No-D_R OPT-GAN is an OPT-GAN without the exploration discriminator D_R; No-Shrinking OPT-GAN represents an OPT-GAN without the shrinking strategy; No-Pre-train OPT-GAN denotes an OPT-GAN without the generator pre-training. Single-D OPT-GAN is an OPT-GAN that replaces the bi-discriminators with the single discriminator for a mixture of $h(\boldsymbol{x})$ and $u(\boldsymbol{x})$ to learn $p(\boldsymbol{x})$.

Ablation Analysis for D_I and D_R The experiments of No-D_I OPT-GAN and No-D_R OPT-GAN are used to verify the influence of D_I and D_R, respectively. D_I determines the exploitation, and D_R contributes to the exploration. When D_I is turned off, the solution generator G only learns from the $u(\boldsymbol{x})$. No-D_I OPT-GAN is similar to a random search optimizer given the lack of the ability to focus on the basin of attraction. Thus, the distribution learned by No-D_I OPT-GAN cannot converge to the optimum in Fig. 2 (a). On the contrary, when D_R is turned off, the solution generator G only learns from the knowledge base, and the mixture distribution $p(\boldsymbol{x})$ denoted by G is equal to $h(\boldsymbol{x})$. In Fig. 2 (b), although the pre-training strategy initializes G with a global viewpoint, No-D_R OPT-GAN rapidly converges to a local optimum because it loses the exploration ability and degenerates into a local optimizer.

Ablation Analysis for Shrinking Strategy No-shrinking OPT-GAN is designed to show the influence of the shrinking strategy which controls the shrinking rate of K (size of $\boldsymbol{x}_{\text{opt}}$). According

[1]The empirical (cumulative) distribution function $F : \mathbb{R} \to [0, 1]$ is defined for a given set of real-valued data S, such that $F(x)$ equals the fraction of elements in S which are smaller than x. The function F is monotonous and a lossless representation of the (unordered) set S.

Table 3: Hyperparameter settings of compared methods

OPT-GAN	The optimal set size $K = 150$, the population size $M = 30$, the shrinking rate $a = 1.5$, the adjustment factor $\lambda = 0.3$. The number of training iterations of GAN $GANIter = 150$, the number of training iterations of discriminators $DIter = 4$, the number of training iterations in pre-training $PreIter = 100$, the gradient penalty factor $\beta = 0.1$, the batch size $S = 30$. Adam is used as the training method. The learning rate of G is 0.0001, and the learning rate of $D = \{D_I, D_R\}$ is 0.005. The code of OPT-GAN is implemented based on the pytorch (version 1.10.2) [16] package, which released under the BSD license.
GBS	The number of drawn samples is 100, the percentile is 0.5. The structure of discriminator is MLP with single hidden layer, and the structure of generator is MLP with two hidden layers. Relu and sigmoid functions are adopted into the hidden layer and output layer, respectively. The number of neurons in the hidden layer is 100. The code of GBS is implemented based on the pytorch package.
WR	The population size $M = 30$. Other hyperparameters follow the settings from the paper[14]. The code of WR provides from authors.
CMA-ES, NM, BFGS, EGL	The population size $M = 30$. Other hyperparameters follow the settings from the paper [12]. The code of CMA-ES is implemented based on the pycma package. The code of NM, BFGS are implemented based on the Scipy package. The code of EGL provides by authors.
PSO	The population size $M = 30$. Other hyperparameters follow the settings from the paper[13]. The code is implemented based on the scikit-opt package.
BoRisk	BoRisk: Hyperparameters follow the setting from the paper[15]. BoRisk-pop30: The size of initial set $num\_samp = 150$, the size of candidates $candidate = 30$ at each iteration; other hyperparameters are same with BoRisk. Their codes provide from authors, which are implemented based on the Botorch package.

to the distributions learned by No-Shrinking OPT-GAN and OPT-GAN in Fig. 2 (c) and (f), respectively, both can learn a nice "arc" which represents the best region. However, OPT-GAN exploits this "arc" region faster than No-shrinking N at the later stage and converges to the optimum gradually, which due to OPT-GAN can quickly refine $h(\boldsymbol{x})$ (for exploitation) when K is small. The above phenomena reflect that the shrinking strategy reinforces OPT-GAN's exploitation ability over time and benefits its convergence.

Ablation Analysis for Generator Pre-training Generator pre-training initializes the G and endows G with a global field-of-view for generating uniformly scattered solutions initially in the searching domain. No-Pre-train OPT-GAN is adopted to demonstrate the significance of generator pre-training. In Fig. 2 (d), the search regions covered by No-Pre-train OPT-GAN only occupy a part of the search domain at initialization. Although No-Pre-train OPT-GAN explores more regions over time owing to the support from D_R, it rapidly converges to the local optimum compared with OPT-GAN. These phenomena illustrate that OPT-GAN initialized with a global field-of-view is beneficial to avoid premature convergence and find optimum.

Ablation Analysis for Single-D OPT-GAN Single-D OPT-GAN is designed to illustrate the

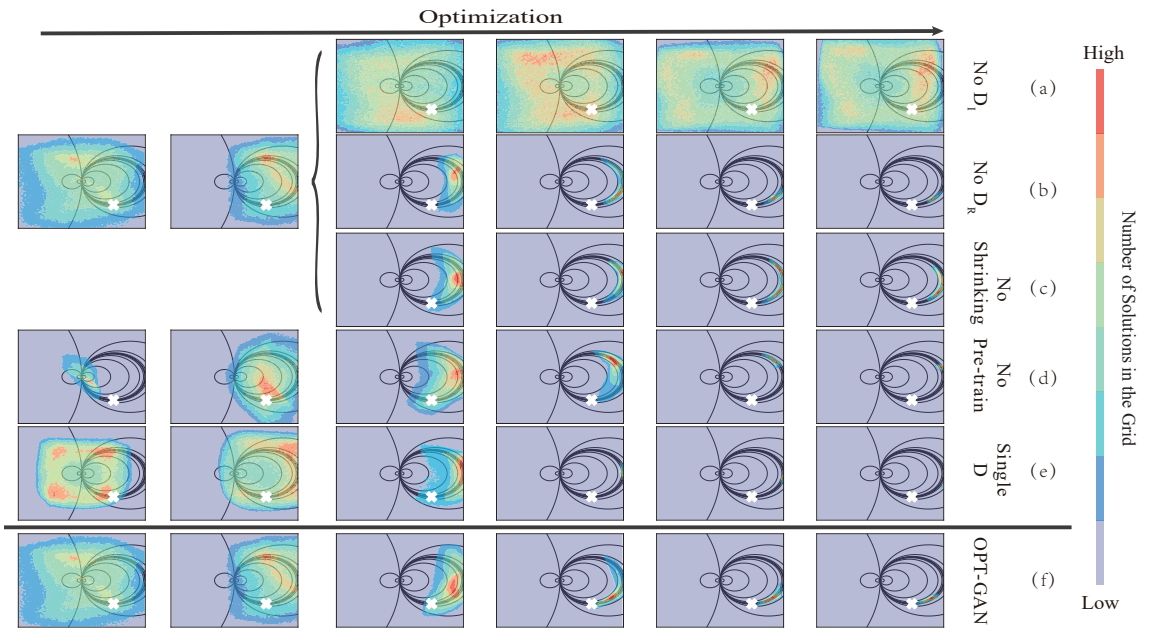

Figure 2: Ablation experiment for exploration discriminator D_R, exploitation discriminator D_I, generator pre-training, shrinking strategy, and bi-discriminators on the 2D Conformal Bent Cigar benchmark. Each subplot is a mixture of the landscape and heatmap. Subplots in the same column denote that they are generated with the same number of FEs. The heatmap displays the distribution denoted by G, and is approximated by Monte Carlo sampling with one million samples. The black $\times$ denotes the position of the optimum. For No-D_R OPT-GAN, No-D_I OPT-GAN, and No-Shrinking OPT-GAN, the corresponding component is removed from OPT-GAN to observe the influence on the optimization, when FEs exceeds the threshold ($FEs > 500$). The hyperparameters are set as follows: $a = 0.525$, $\lambda = 0.3$, $m = 30$, $K = 150$, $PreIter = 130$, and $MAXFes = 3500$. Other hyperparameter settings are the same with those in Table 3.

significance of the bi-discriminators. Fig. 2 (e) shows the distribution learned by Single-D OPT-GAN. Both OPT-GAN and Single-D OPT-GAN generators estimate the mixture distribution of $h(\boldsymbol{x})$ and $u(\boldsymbol{x})$. As one component of this mixture distribution is the uniform distribution, the generator G is more interested in the region where $h(\boldsymbol{x})$ gives a higher probability to sample than other regions. For Single-D OPT-GAN, the single discriminator does not explicitly guide G to balance the exploitation ($h(\boldsymbol{x})$) and exploration ($u(\boldsymbol{x})$) in the learning process simultaneously, G may prefer to estimate $h(\boldsymbol{x})$ more accurately in order to decrease adversarial loss. Thus, Single-D OPT-GAN may focus on exploitation rather than exploration over time. It can be observed that Single-D OPT-GAN rapidly converges to a local optimum. For OPT-GAN, G is guided by D_I and D_R, respectively, and is enforced to confuse D_I and D_R simultaneously. Thus, G needs to consider the balance of exploration ($u(\boldsymbol{x})$) and exploitation ($h(\boldsymbol{x})$) to decrease its adversarial loss, because the generator in the GAN with multi-discriminators tends to prefer the balance for all discriminators [17]. Although the convergence speed of OPT-GAN is slower than Single-D OPT-GAN, OPT-GAN converges to the global optimum gradually. These phenomena reveal the performance of OPT-GAN with bi-discriminators is better than the single discriminator.

D.3 Convergence Analysis

As discussed in Section A, EDA algorithms are one of the commonly used global optimizers, proven by numerous practical applications and experiments [3, 18, 19]. Unfortunately, the theoretical

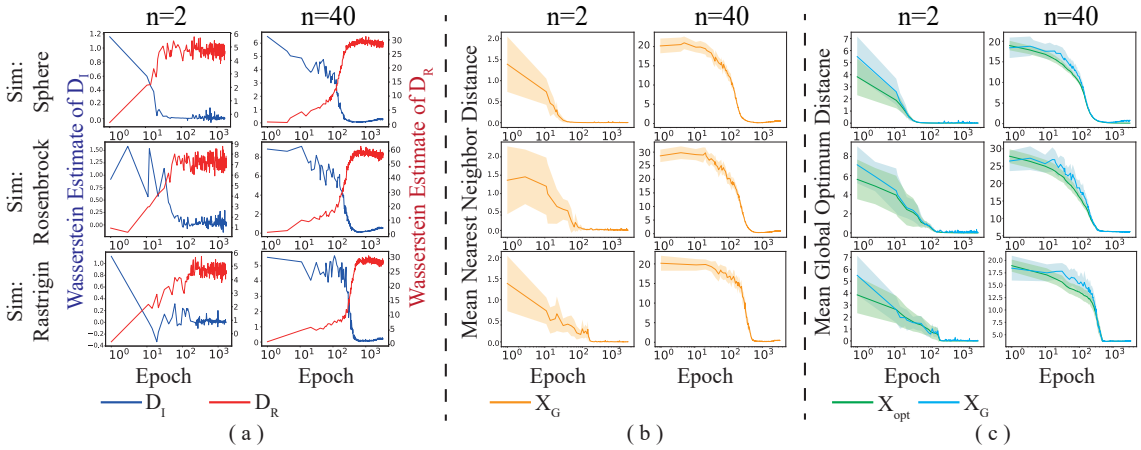

Figure 3: Verifying the convergence of OPT-GAN through the three perspectives: (a) wasserstein estimate of D_I and D_R; (b) the mean nearest neighbor distance curve between generated solutions $\boldsymbol{x_G}$; (c) the mean global optimum distance curves of the global optimum $\boldsymbol{x}^*$ to $\boldsymbol{x_G}$ and $\boldsymbol{x_{opt}}$ respectively. The shaded area of curves reflects the standard deviation.

performance analysis of EDA, as metaheuristic algorithms, is still limited and only restricted to particular problem types. Usually, the experimental study is used to analyze the properties of EDA algorithms, such as convergence analysis[5, 6, 7].

In order to verify the convergence of OPT-GAN, a series of experiments are carried out via several benchmarks from three perspectives, including (a) whether the distribution represented by G can converge to a stable state over time, (b) whether the generated solutions by G can converge together as iteration goes on, and (c) whether the generated solutions converge toward the region with global optimum.

Fig. 3 (a) provides the Wasserstein estimate curves related with G and $D = \{D_I, D_R\}$. The Wasserstein estimate is proposed in Wasserstein GAN [20], and is used to estimate the Earth-Mover Distance between the distribution denoted by generator and the distribution behind the "real" samples. When Wasserstein estimates move toward a fixed value, it means that the distribution represented by the generator converges to a stable state.

The Wasserstein estimate between G and D_I ($\mathcal{W}_{\mathrm{D_I}}$) is defined as

$$\mathcal{W}_{\mathrm{D_I}} = \frac{1}{S}\sum_{s=1}^{S}(D_I(G(\boldsymbol{\eta}^{(s)})) - D_I(\boldsymbol{x}^{(s)}_{\mathrm{sampling}})), \tag{1}$$

and the Wasserstein estimate between G and D_R ($\mathcal{W}_{\mathrm{D_R}}$) is defined as

$$\mathcal{W}_{\mathrm{D_R}} = \frac{1}{S}\sum_{s=1}^{S}(D_R(G(\boldsymbol{\eta}^{(s)})) - D_R(\boldsymbol{x}^{(s)}_{\mathrm{uniform}})). \tag{2}$$

In Fig. 3 (a), the blue curve that represents $\mathcal{W}_{\mathrm{D_I}}$ fluctuates down and converges to zero gradually; the red curve that represents $\mathcal{W}_{\mathrm{D_R}}$ fluctuates up and converges to a stable value. The pre-training strategy aids G to learn a multivariate uniform distribution $u(\boldsymbol{x})$ before optimization. Thus, the initial value of $\mathcal{W}_{\mathrm{D_R}}$ is small, whereas $\mathcal{W}_{\mathrm{D_I}}$ is large. As iteration goes on, $\boldsymbol{x}_{\mathrm{opt}}$ is constantly updated by the generated solutions from G, and its size gradually shrinks until the $K = 1$. As a result, the distribution $h(\boldsymbol{x})$ denoted by the optimal set becomes stable gradually, and $\mathcal{W}_{\mathrm{D_I}}$ converges to zero. $p(\boldsymbol{x})$ is also able to move toward a stable distribution, because $p(\boldsymbol{x})$ is the mixture distribution of $h(\boldsymbol{x})$ and a fixed multivariate uniform distribution $u(\boldsymbol{x})$.

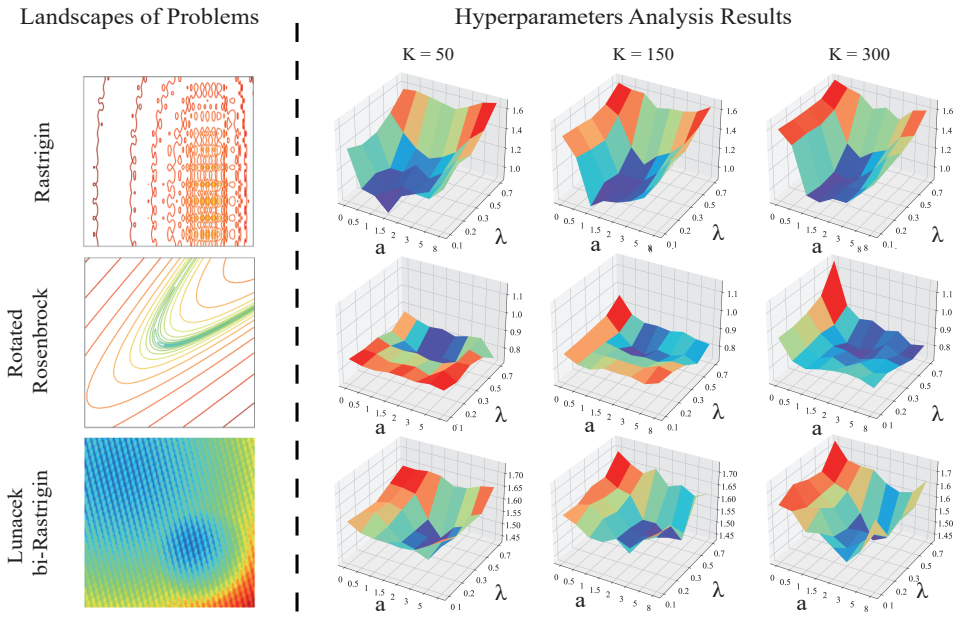

Figure 4: Hyperparameter analysis of the optimal set size K, shrinking rate a, and adjustment factor λ. The ordinate represents the fitness, which is computed by $\log_{10} f$. The experimental results are derived from the average of 15 repeated trials with $MAXFes = 10000$. Other hyperparameter settings are the same with those in Table 3.

Fig. 3 (b) displays the mean nearest neighbor distance (MNND) between generated solutions at each iteration. The MNND denotes the mean of the Euclidean distances between each generated solution and its nearest neighbor in $\boldsymbol{x}_{\mathrm{G}}$. It can be observed that the orange curves gradually converge to zero from the large value as the iteration goes on, which illustrates that the generated solutions spread in different regions at the early stage and gradually converge together at the later stage. Fig. 3 (c) shows the mean global optimum distances based on $\boldsymbol{x}_{\mathrm{G}}$ and $\boldsymbol{x}_{\boldsymbol{opt}}$, respectively. The mean global optimum distance (MGOD) is the mean of the Euclidean distances between the global optimum $\boldsymbol{x}^*$ and each solution. The blue curve (or green curve) represents MGOD between $\boldsymbol{x}^*$ and the generated solutions in $\boldsymbol{x}_{\mathrm{G}}$ (or the historical best solutions in $\boldsymbol{x}_{\boldsymbol{opt}}$). The blue and green curves start with a large value; then, they fluctuate down and stabilize to a small value. This phenomenon reveals that OPT-GAN is able to converge to the global optimum. Therefore, the distribution $p(\boldsymbol{x})$ represented by G can converge to a stable state over time, and move toward the delta distribution whose spike is located at $\boldsymbol{x}^*$, according to Fig. 3 (a), (b), and (c).

D.4 Hyperparameter Analysis

How can the hyperparameters of OPT-GAN be tuned in practice? Although OPT-GAN involves multiple hyperparameters, some (e.g., parameters related to the GAN) are analyzed [21, 22]. Thus, only the main hyperparameters related to the proposed OPT-GAN, which are the adjustment factor λ, shrinking rate a, and optimal set size K, are analyzed and reported in Fig. 4. Several benchmarks with different preferences to the exploration and exploitation ability are used. The dimensionality is set to 10. The Rastrigin function includes many local optimal regions around the optimum, and is preferred by the optimizer with strong exploitation ability. The Rotated Rosenbrock function is an ill-conditioned function with a bending valley, which is preferred by the optimizer with strong exploration ability. Lunacek bi-Rastrigin function possesses two funnel-shaped regions, and the global optimum is located in the smaller funnel, which poses the challenge for the optimizer's ability to balance E-E trade-off.

The adjustment factor λ guides the trade-off between exploration and exploitation at each iteration by controlling the effects of $h(\boldsymbol{x})$ (for exploitation) and $u(\boldsymbol{x})$ (for exploration) on $p(\boldsymbol{x})$ (for estimation the distribution of optimum). When λ is small, $p(\boldsymbol{x})$ is dominated by $h(\boldsymbol{x})$, which enables OPT-GAN to display a better exploitation ability. With the increase of λ, the influence of $u(\boldsymbol{x})$ on $p(\boldsymbol{x})$ increases gradually, enhancing the exploration ability of OPT-GAN. Extremely, if λ is $+\infty$, then it degrades into a random search optimizer. In Fig. 4, Rastringin function prefers low λ because it stresses more on exploitation ability than on exploration ability. For the Rotated Rosenbrock function, OPT-GAN achieves better performance when λ is large as this benchmark has two peaks. This characteristic forces an optimizer to explore the searching domain. The same phenomenon can be observed on Lunacek bi-Rsatrigin because of the existence of two major basins of attraction. Thus, a carefully tuned λ can balance the exploration and exploitation of OPT-GAN. Balancing E-E trade-off would boost the performance on diversified landscapes. Overall, λ is suggested with the small value for the unimodal problems, and is recommended with the large value for multi-modal optimization problems.

The initial size of the optimal set K and its shrinking rate a are interrelated. K defines the initial size of the optimal set, and a controls how fast the size of the optimal set reduces. The resultant force of K and a can control the E-E trade-off along the time line. A small a indicates a slow shrinking speed. A large K with a small a supports the optimal set with a large size, and its size decreases slowly, which enables the exploration ability of OPT-GAN to reduce slowly. In Fig. 4, with the increase of K, the performance improves on the Rotated Rosenbrock, and declines on the Rastrigin. These phenomena reveal that the exploration ability raises as K increases. However, when a is over small or K is over large, refining $h(\boldsymbol{x})$ could be slow and trapped into a "tug-of-war" as the iteration proceeds, which declines the convergence speed of OPT-GAN. In addition, OPT-GAN performs better on the Rastrigin with the growth of a. A large a leads to a rapid decline in the size of the optimal set and boosts the preference to exploitation. Therefore, K and a should be tuned carefully to balance the exploration and exploitation. Overall, K combined with a small a is suggested for the multi-modal problems in order to boost the exploration ability of OPT-GAN, and K combined with a large a is recommended for the problems that are preferred by the optimizer with strong exploitation ability.

Table 4: Results of t-tests with the significance level of 0.05 between OPT-GAN and trandional optimizers.

	BoRisk-pop1		BoRisk-pop30		CMA-ES		BFGS		Nelder-Mead		PSO		BORE	
	n=2	n=10	n=2	n=10	n=2	n=10	n=2	n=10	n=2	n=10	n=2	n=10	n=2	n=10
Better	7	7	6	8	5	4	7	7	7	7	3	8	4	8
Worse	0	0	0	0	0	3	0	1	0	1	2	0	1	0
Same	1	1	2	0	3	1	1	0	1	0	3	0	3	0
Merit	7	7	6	8	5	1	7	6	7	6	1	8	3	8

D.5 Comparison with Traditional Optimizers

To validate the performance of OPT-GAN compared with traditional optimizers in terms of statistical significance, a t-test is conducted. The results are reported in Table 4. "Better", "Same", and "Worse" represent the number of benchmarks that the performance of OPT-GAN is significantly better, almost the same as, and significantly worse than the opponents, respectively. "Merit" is "Better" minus "Worse". It can be observed that OPT-GAN is better than BoRisk, BoRisk-pop30, BFGS, CMA-ES, and Nelder-Mead in 2 and 10-dimensional problems. Moreover, "Merit" rises with the increase of the dimensionality compared with PSO, BoRisk and BoRisk-pop30. However, compared with BFGS, Nelder-Mead, and CMA-ES, "Merit" decreases as the dimensionality increases from 2 to 10. This is perhaps because the complexity of certain high-dimensional problems results in OPT-GAN having to deliberate on the landscape, slowing down the optimization.

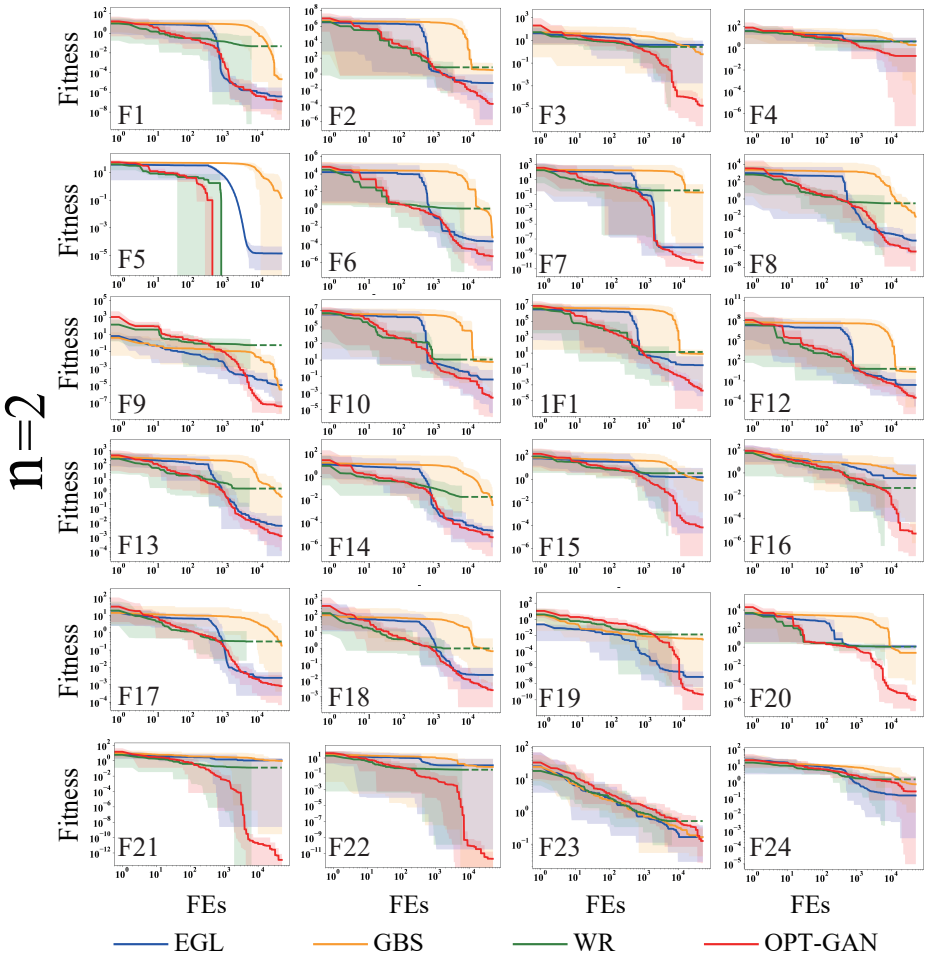

Figure 5: Average convergence curves of OPT-GAN and NN-based optimizers on different benfchmarks in 2 dimensions. Each curve denotes the average fitness of an optimizer on 15 repeated trials. The shadow area of each curve is limited by the worst and best solutions in each evaluation. The dotted line indicates that the optimizer stops because its elapsed time exceeds 3 hour.

D.6 Comparison with NN-based Optimizers

We conduct a t-test to analyze the statistical significance of OPT-GAN over other neural network competitors. The results are reported in Table 5. OPT-GAN is significantly better than other models on most COCO benchmarks. The “Merit" of OPT-GAN over GBS verifies that the adopted strategies for balancing E-E trade-off improves GAN for global optimization. EGL displays better performance as the dimensionality increases because it is designed to tackle the high dimensional problems [12]; even so, the “Better" and “Merit" of OPT-GAN exceed EGL. Furthermore, the “Merit" of OPT-GAN boosts as the dimensionality increases compared with WR. A possible explanation is that the high time consumption of WR affects its ability to find the optimum under the limited resources.

We also provide all the convergence results of those NN-based optimizers on all the 24 problems in 2D, 10D problems (Fig. 5 and Fig. 6). On F3, F15-F18, F21, and F22, OPT-GAN has an undeniable advantage over other optimizers in 2 and 10 dimensions, while in F1, F14, and F23, it is equal to or even slightly worse than EGL. By observing the contour maps of different functions, we deduced that EGL, as a local gradient algorithm, is good at solving unimodal ill-conditioned problems. While OPT-GAN, as a global optimizer, has better average performance and is good at solving multi-modal or non-convex problems.

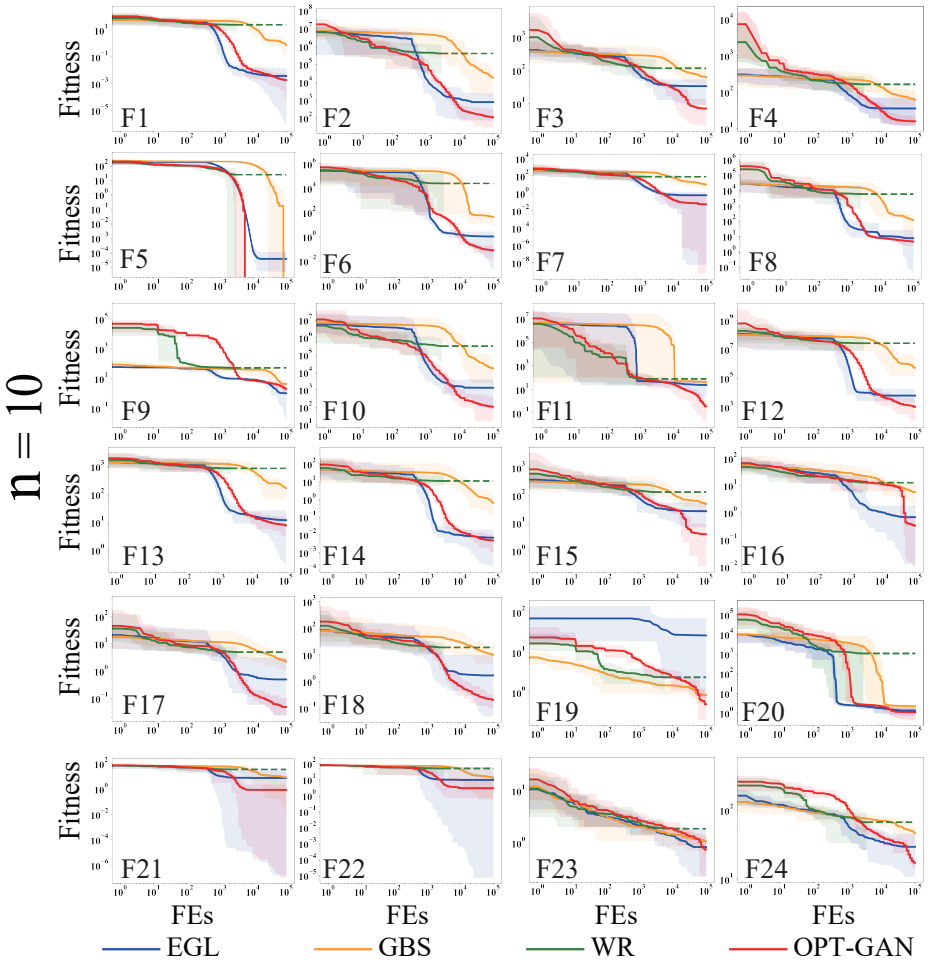

Figure 6: Average convergence curves of OPT-GAN and NN-based optimizers on different benchmarks in 10 dimensions. Each curve denotes the average fitness of an optimizer on 15 repeated trials. The shaded area of the averaged convergence curve reflects the “Range". The dotted line indicates that the optimizer stops because its elapsed time exceeds 3 hour.

Fig. 7 displays the comparative results of the ECDF in 2 and 10 dimensions. The results demonstrate the superiority of OPT-GAN over other neural network-based optimizers, especially with the increase in dimensionality. Although the PT of OPT-GAN lags behind other competitors at the early stages, it boosts rapidly as the optimization proceeds. This is because the strategies to balance E-E trade-off take time initially to form a global perspective, and OPT-GAN pinpoints the global optimum gradually along the basin of attraction.

D.7 Experimental Configuration for Optimizating NEEP Problem

In this part, we validate the performance of OPT-GAN by a real-world problem of optimizing the neural network-based symbolic regression method (NEEP)[23]. As a symbolic regression model, a recurrent neural network (RNN) in NEEP is used to generate the expression trees that describe the relationship between the input variables and the output variable. Therefore, the optimization of NEEP is to optimize the connection weights of RNN, which is a multi-modal optimization problem with the high dimensionality. The mean square error (MSE) can be considered as the fitness function. In the optimization, the uncertainty of the expression tree, such as structure and length, causes

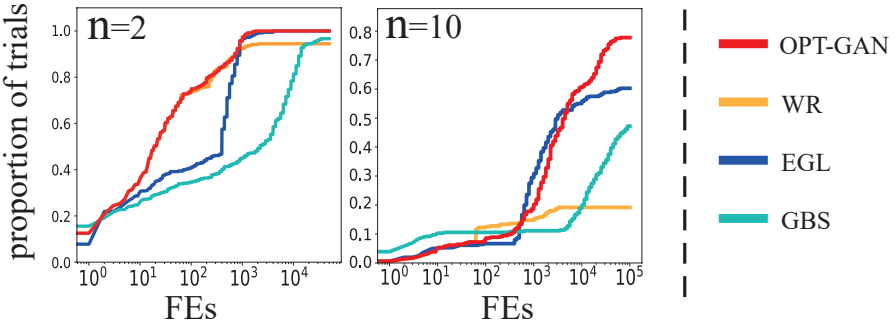

Figure 7: The ECDF results of OPT-GAN compared with NN-based optimizers based on 15 repeated trials on 24 benchmarks in 2 and 10 dimensions.

Table 5: Results of t-tests with the significance level of 0.05 between OPT-GAN and neural network-based optimizers.

	EGL			GBS			WR		
	n=2	n=10	n=40	n=2	n=10	n=40	n=2	n=10	n=40
Better	16	15	16	8	23	23	19	23	24
Worse	0	1	3	0	0	1	0	0	0
Same	8	8	5	16	1	0	5	1	0
Merit	16	14	13	8	23	22	19	23	24

the uncertain evolved function. It is difficult to directly compute gradients with BP for training RNN in NEEP [23]. As an E-E trade-off global optimizer, OPT-GAN can optimize NEEP without considering the gradient.

We evaluated the performance of OPT-GAN compared with neural network-based optimizers (GBS, WR, EGL), and the traditional optimizers (BoRisk, CMA-ES, PSO) by optimizing NEEP on 3 synthetic symbolic regression problems (nico5, nico6, nico11) [24] and 2 UCI data sets (Concrete and Energy)[25]. **The dimension of NEEP optimization problem is 400 for nico5, nico6, and nico11, and 640 for concrete and energy.** For visualizing the 2-dimensional (2D) projected landscapes [26] of NEEP, we use two random vectors named d1 and d2, with range of [-2, 2]. Precisely, They can map the area of the high-dimensional landscape around the weight optimum into the 2D bounded space of [-2, 2].

The generated expression tree is based on the function symbol set $\{+, -, \times, /, sin, cos, \ln|x|\}$. In order to tackle that divisor is zero, the / is realized by $x_1/(x_2+\varphi)$ where $\varphi = 0.01$; and the ln is protected by $\ln|x+\varphi|$. The hyperparameters of the optimizers follow the table 3. Each optimizer repeats 15 times on each dataset, the maximum number of iterations is 100,000. A single run is terminated if: (1) fitness reaches the precision ($fbest - f^* - prec < 0$), (2) FEs arrives at the maximum number, or (3) the elapsed time exceeds 10 hours.

D.8 Time Consumption of Learning-based Algorithms

In Table 6, we present the time consumption of learning-based algorithms. OPT-GAN has comparable time consumption with other NN-based optimizers. BoRisk performs a posteriori evaluation based on all historical candidates on the objective function and models a Gaussian process, which makes its time consumption increase exponentially with FEs. Therefore, according to the experimental results, the time consumption of BoRisk at $FEs = 10000$ is much larger than 24 hours. In contrast, GBS uses the real objective function for evaluation during the neural network training process, so FEs are consumed quickly and with low CPU time consumption. However, this optimization approach makes GBS underfitting with limited FEs and poor accuracy of modeling in optimization (see Fig.

Table 6: Time consumption of OPT-GAN and learning-based optimizers.

	OPT-GAN	WR	EGL	GBS	BoRisk
Rastrigin	0.44h	25.3h	0.28h	4.36E-4h	$\gg$24h
Attractive Sector	0.52h	28.1h	0.38h	4.53E-4h	$\gg$24h
Different Powers	0.51h	22.5h	0.27h	4.11E-4h	$\gg$24h
Schaffers F7	0.52h	20.5h	0.27h	4.03E-4h	$\gg$24h
Gallagher's Gaussian 101-me	0.44h	25.4h	0.28h	3.33E-4h	$\gg$24h
Average	0.49h	24.36h	0.30h	4.08E-4h	$\gg$24h

The *FEs* is set to 10000, the dimensionality of benchmarks is 10. Other hyperparameter settings are the same with those in Table 3. h means hours.

5- Fig. 7).

Despite the fact that OPT-GAN, as well as most NN-based optimizers, is more time-consuming than classical optimizers, with the development of hardware, such as the increase in computing power, decrease in hardware price, and especially development of application-specific processors (see AI Accelerators [27, 28]), the time-consumption gap between deep learning-based optimizers and classical ones will become negligible. This type of method will then be a promising direction by providing better solutions within a similar total optimization time.

D.9 Computing Resource

The numerical experiments were collected from several machines, including Intel(R) Core(TM) i7-10700K 3.80GHz CPU with Linux operating system, Intel(R) Xeon(R) CPU E5-2620v3 2.40GHz CPU X2 with Linux operating system, Intel(R) Xeon(R) CPU E5-2620 2.00GHz CPU X2 with Linux operating system, AMD Rome 7H12 2.60GHz CPU with Linux operating system, Intel(R) Xeon(R) Silver 4214R 2.40GHz with Linux operating system, Intel(R) Core(TM) i7-8750H 2.20 GHz CPU NVIDIA GeForce GTX 1060Ti GPU with Windows 10 operating system.

E Visualization of $p(\boldsymbol{x})$

The visualization of the distribution $p(\boldsymbol{x})$ contains a true landscape of the problem and a heatmap of $p(\boldsymbol{x})$. The landscape of the problem is visualized as a contour map. The heatmap of $p(\boldsymbol{x})$ is approximated by Monte Carlo sampling with one million samples. Sampling methods from $p(\boldsymbol{x})$ are different in different optimizers due to the various designs of p.

According to Section A, CMA-ES explicitly obtains the $p(\boldsymbol{x})$ from a Gaussian model. Thus, the one million solutions are sampled from the updated Gaussian model in a stage. OPT-GAN obtains the $p(\boldsymbol{x})$ by the intrinsic distribution of G, which is used to sample the one million solutions with different input random noise. BoRisk, as a surrogate optimization algorithm, indirectly learns $p(\boldsymbol{x})$ by GP, which models the problem landscape $q(f)$. BoRisk generally samples a candidate solution from $p(\boldsymbol{x})$ by maximizing the acquisition function over the GP. The one million solutions are obtained by randomly repeating the process of optimizing the acquisition function one million times.

References

[1] Jeremy De Bonet, Charles Isbell, and Paul Viola. Mimic: Finding optima by estimating probability densities. In *Advances in Neural Information Processing Systems*, volume 9. MIT Press, 1996.

[2] Weishan Dong, Tianshi Chen, Peter Tiňo, and Xin Yao. Scaling up estimation of distribution algorithms for continuous optimization. *IEEE Transactions on Evolutionary Computation*, 17 (6):797–822, 2013.

[3] Mark Hauschild and Martin Pelikan. An introduction and survey of estimation of distribution algorithms. *Swarm and Evolutionary Computation*, 1(3):111–128, 2011.

[4] Nikolaus Hansen, Sibylle D. Müller, and Petros Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary Computation*, 11(1):1–18, 2003.

[5] Ramses Sala, Niccolò Baldanzini, and Marco Pierini. Global optimization test problems based on random field composition. *Optimization Letters*, 11(4):699–713, Apr 2017.

[6] M. Gallagher and Bo Yuan. A general-purpose tunable landscape generator. *IEEE Transactions on Evolutionary Computation*, 10(5):590–603, 2006.

[7] Jonathan L. Shapiro. Diversity loss in general estimation of distribution algorithms. In *Parallel Problem Solving from Nature - PPSN IX*, volume 193, pages 92–101, 2006.

[8] Nikolaus Hansen, Anne Auger, Raymond Ros, Olaf Mersmann, Tea Tušar, and Dimo Brockhoff. COCO: a platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36(1):114–144, 2021.

[9] K. V. Price, N. H. Awad, M. Z. Ali, and P. N. Suganthan. Problem definitions and evaluation criteria for the 100-digit challenge special session and competition on single objective numerical optimization. Technical report, Nanyang Technological University, Nov 2018.

[10] Chunxiuzi Liu, Fengyang Sun, Qingrui Ni, Lin Wang, and Bo Yang. A novel graphic bending transformation on benchmark. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1538–1543, 2020.

[11] S. Surjanovic and D. Bingham. Virtual library of simulation experiments: Test functions and datasets. Retrieved December 16, 2021, from `http://www.sfu.ca/~ssurjano`.

[12] Elad Sarafian, Mor Sinay, Yoram Louzoun, Noa Agmon, and Sarit Kraus. Explicit gradient learning for black-box optimization. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 8480–8490, Jul 2020.

[13] Ho-Kin Tang and Sim Kuan Goh. A novel non-population-based meta-heuristic optimizer inspired by the philosophy of yi jing. *arXiv preprint arXiv:2104.08564*, 2021.

[14] Austin Tripp, Erik Daxberger, and José Miguel Hernández-Lobato. Sample-efficient optimization in the latent space of deep generative models via weighted retraining. In *Advances in Neural Information Processing Systems*, volume 33, pages 11259–11272, 2020.

[15] Sait Cakmak, Raul Astudillo Marban, Peter Frazier, and Enlu Zhou. Bayesian optimization of risk measures. In *Advances in Neural Information Processing Systems*, volume 33, pages 20130–20141, 2020.

[16] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, volume 32, pages 8024–8035. 2019.

[17] Isabela Albuquerque, Joao Monteiro, Thang Doan, Breandan Considine, Tiago Falk, and Ioannis Mitliagkas. Multi-objective training of generative adversarial networks with multiple discriminators. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 202–211, Jun 2019.

[18] Ilya Loshchilov. Cma-es with restarts for solving cec 2013 benchmark problems. In *2013 IEEE Congress on Evolutionary Computation*, pages 369–376, 2013.

[19] Oswin Krause, Dídac Rodríguez Arbonès, and Christian Igel. Cma-es with optimal covariance update and storage complexity. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

[20] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 214–223, Aug 2017.

[21] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. 2018. arXiv: 1802.05957.

[22] Diederik P. Kingma and Jimmy Ba. Adam: a method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2017.

[23] Aftab Anjum, Fengyang Sun, Lin Wang, and Jeff Orchard. A novel neural network-based symbolic regression method: Neuro-encoded expression programming. In *Artificial Neural Networks and Machine Learning – ICANN 2019: Deep Learning*, volume 11728, pages 373–386, 2019.

[24] Miguel Nicolau, Alexandros Agapitos, Michael O'Neill, and Anthony Brabazon. Guidelines for defining benchmark problems in genetic programming. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 1152–1159, 2015.

[25] Arthur Asuncion and David Newman. UCI machine learning repository, 2007.

[26] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31, 2018.

[27] Norman P. Jouppi, Doe Hyun Yoon, Matthew Ashcraft, Mark Gottscho, Thomas B. Jablin, George Kurian, James Laudon, Sheng Li, Peter Ma, Xiaoyu Ma, Thomas Norrie, Nishant Patil, Sushma Prasad, Cliff Young, Zongwei Zhou, and David Patterson. Ten lessons from three generations shaped google's tpuv4i : Industrial product. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–14, 2021.

[28] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *SIGARCH Comput. Archit. News*, 42(1):269–284, Feb 2014.