The interplay of machine learning and metaheuristics

Hojjat Rakhshani

The Interplay of Machine Learning and Metaheuristics

by

Hojjat Rakhshani

DOCTOR OF PHILOSOPHY

in

Computer Science

UNIVERSITÉ DE HAUTE-ALSACE

Laetitia Jourdan, Full Professor, Université de Lille	Reviewer
Cyril Fonlupt, Full Professor, Universit du Littoral Cte d'Opale	Reviewer
Edward Keedwell, Full Professor, University of Exeter	Examiner
Pierre Collet, Full Professor, Université de Strasbourg	Examiner
Abderrafiaa Koukam, Full Professor, Université de Belfort Montbélliard	Examiner
Lhassane Idoumghar, Full Professor, Université de Haute-Alsace	Supervisor
Mathieu Brévilliers, Associate professor, Université de Haute-Alsace	Co-Supervisor
Julien Lepagnot, Associate professor, Université de Haute-Alsace	Co-Supervisor

15 May 2020

Abstract

Optimization algorithms have seen unprecedented growth thanks to their successful applications in fields including engineering and health sciences. Similarly, machine learning has been popularly used in perceptual tasks by both academic and industrial researchers. They are both designed to find solutions for some specific tasks and it is not straightforward to apply an existing method to a new domain and still have superior results. Hence, experts have to construct specialized methods for each given task. The extra degree of freedom from the design space could make this process very time-consuming and has motivated a demand for automated search methods that can be adopted easily without any expert knowledge. In this thesis, we claim the mentioned contribution by porting existing methods from machine learning to optimization domain and vice versa. The first part of this thesis suggests many lines of investigation with possibilities related to the development of more enhanced optimization algorithms using machine learning. The second part discusses a modeling scheme to optimize the performance of machine learning tools with metaheuristic algorithms.

Résumé

Les algorithmes d'optimisation ont connu une croissance sans prcdent grce leurs applications russies dans de nombreux domaines notamment en ingnierie et en sant. De mme, les algorithmes dapprentissage automatique ont t couramment utiliss sur des problmes de perception par les chercheurs et les industriels. Les deux catgories dalgorithmes sont conus pour trouver des meilleurs solutions certaines problmes spcifiques et leurs adaptations pour un nouveau problme ne garantit pas lobtention de bons rsultats. Par consquent, les experts doivent construire des mthodes spcialises pour chaque problme donn. Cela ajoute un degr de libert supplmentaire dans la phase de conception qui pourrait rendre le processus de dveloppement trs long. Cela a motiv une demande de mthodes de recherche qui peuvent tre adoptes facilement sans aucune connaissance experte. Dans cette thse, nous proposons des pistes dintgration des mthodes dapprentissage automatique dans les mtaheuristiques et vice-versa. La premire partie de cette thse dcrit quelques pistes d'investigation avec des possibilits lies au dveloppement d'algorithmes d'optimisation plus avancs utilisant le machine learning. La deuxime partie prsente un schma de modlisation pour optimiser les performances des outils d'apprentissage automatique tout en utilisant des algorithmes doptimisation bass sur des mtaheuristiques.

Preface

This thesis is submitted for the degree of Doctor of Philosophy in Computer Science at Université de Haute-Alsace. This research was conducted under the supervision of Prof. Lhassane Idoumghar. To the best of our knowledge, the presented work is original except where references are made to previous studies. Neither this, nor any other similar work is being submitted for any other degree, diploma or other qualification at any other university.

Publications

International Journals

- H. Rakhshani, L. Idoumghar, S. Ghambari, J. Lepagnot, M. Brévilliers, Neural architecture search for global optimization, Journal of IEEE Transaction on Evolutionary Computation (**submitted**).
- H. Rakhshani, L. Idoumghar, J. Lepagnot, M. Brévilliers, Speed up differential evolution for computationally expensive protein structure prediction problems, Swarm and Evolutionary Computation, 2019, vol 50, pp. 1-18.
- W.W. Koczkodaj, J.P. Magnot, J. Mazurek, J.F. Peters, H. Rakhshani, M. Soltys, D. Strzak, J. Szybowsk, A. Tozzi, On normalization of inconsistency indicators in pairwise comparisons, International Journal of Approximate Reasoning, vol 86, pp. 73-79.
- H. Rakhshani, E. Dehghanian, A. Rahati, Hierarchy cuckoo search algorithm for parameter estimation in biological systems, Chemometrics and Intelligent Laboratory Systems, 2016, vol 159, pp. 97-107.
- H. Rakhshani, A. Rahati, Snap-drift cuckoo search: A novel cuckoo search optimization algorithm. Applied Soft Computing, vol 52, pp. 771-794.
- H. Rakhshani, A. Rahati, Intelligent multiple search strategy cuckoo algorithm for numerical and engineering optimization problems, Arabian Journal for Science and Engineering, vol 42.2, pp. 567-593.

International Conference with Program Committee

- H. Rakhshani, H. Ismail-Fawaz, L. Idoumghar, G. Forestier, J. Weber, J. Lepagnot, M. Brévilliers, P. Muller, Optimizing deep residual neural networks for time series classification, The 2020 International Joint Conference on Neural Networks (IJCNN), 2020, Glasgow (**Accepted**).
- H. Rakhshani, B. Latard, M. Brévilliers, J. Weber, J. Lepagnot, G. Forestier, M. Hassenforder, L. Idoumghar, Automated machine learning for information retrieval in scientific articles, 2020 IEEE Congress on Evolutionary Computation (CEC), 2020, Glasgow (**Accepted**).
- H. Rakhshani, L. Idoumghar, J. Lepagnot, M. Brévilliers, MAC: Many-objective Automatic Algorithm Configuration, In: Deb K. et al. (eds) Evolutionary Multi-Criterion Optimization, Lecture Notes in Computer Science, Springer, 2019, vol 11411.
- H. Rakhshani, L. Idoumghar, J. Lepagnot, M. Brévilliers, From feature selection to continues optimization, International Conference on Artificial Evolution (EA-2019), 2019, Mulhouse France. LNCS Volume, pp.1-8.

- H. Rakhshani, L. Idoumghar, J. Lepagnot, M. Brévilliers and E. Keedwell, Automatic hyperparameter selection in Autodock, 2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), Madrid, Spain, 2018, pp. 734-738.
- E. Keedwell, M. Brévilliers, L. Idoumghar, J. Lepagnot and H. Rakhshani, A Novel Population Initialization Method Based on Support Vector Machine, 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Miyazaki, Japan, 2018, pp. 751-756.
- H. Rakhshani, L. Idoumghar, J. Lepagnot, M. Brévilliers and A. Rahati, Accelerating Protein Structure Prediction Using Active Learning and Surrogate-Based Optimization, 2018 IEEE Congress on Evolutionary Computation (CEC), Rio de Janeiro, 2018, pp. 1-6.
- H. Rakhshani, L. Idoumghar, J. Lepagnot, M. Brvilliers, Application of the surrogate models for protein structure prediction, 7th International Conference on Metaheuristics and Nature Inspired Computing (META18), Oct 2018, Marrakech in Morocco.

International Workshops

- H. Rakhshani, L. Idoumghar, J. Lepagnot and M. Brvilliers, From feature extraction to continues optimization, International Workshop on Stochastic Local Search Algorithms, Sep 2019, Villeneuve d'Ascq, France.

Co-Authorship Statement

Over the past three years I have had the good fortune to work with excellent authors from academia and industry. I am grateful for their many contributions to the research described in this thesis. The first work relates to the major impact of surrogate models for high dimensional protein structure prediction problem. An early version of this work back to a published work during my MSc thesis, which was supervised by Amin Rahati. In another research with Edward Keedwell, we proposed to use a machine-learning technique to enhance the convergence rate of the differential evolution algorithm. Moreover, a part of the experiments in Part II is used for a joint work with Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, and Pierre-Alain Muller. We proposed a simple-yet-effective fine-tuning method based on repeated k-fold cross-validation in order to train deep residual networks using only a small amount of time series data. I am gratefully indebted to Hassan Ismail Fawaz, Germain Forestier and Jonathan Weber for being generous with their time and so prompt with their feedback.

In another research, we proposed to use meta-heuristics for predicting the similarity between pairs of scientific articles. Accordingly, Bastien Latard, Jonathan Weber, Germain Forestier and Michel Hassenforder got involved in the project and, notably, contributed to gain insight into the problem. Mathieu Brévilliers deserves credit for much of the modification presented in this joint work; thank you Mathieu for valuable discussions of that research. In another context, we applied the meta-heuristics to find an optimized model from the TwoStream Inflated 3D architecture, pre-trained on the ImageNet and the Kinetics source datasets, to classify video sequences of crowd movements from the Crowd-11 target dataset. This joint work with Mounir Bendali-Braham, Germain Forestier, Jonathan Weber, and Pierre-Alain Muller is about to be submitted for publication.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would like to express my sincere gratitude to my supervisor Professor Lhassane Idoumghar for the continuous support of my PhD study and related research, for his patience, and motivation. I am also grateful to my co-supervisors Julien Lepagnot and Mathieu Brvilliers who worked and helped me on this thesis deserve specific commendations. Besides my supervisors, I would like to thank thesis committee for their insightful comments and encouragement. I also offer my heartfelt thanks to my former supervisor -Amin Rahati- for being excellent mentors and superb collaborator. I also want to thank all of the PhD students at the Université de Haute-Alsace. Thank you for making the IRIMAS an enjoyable and stimulating research institute: Soheila Ghambari, Hassan Ismail Fawaz, Mahmoud Golabi, Julien Kritter, Bastien Latard, Mokhtar Essaid, Mounir Bendali Braham, and Imene Zaidi. I especially learned a great deal from working with Julien; thank you for being so generous with your time when I was struggling with my GPU implementations. Last but not the least, I would like to thank my family: my parents and to my brother and sister for supporting me spiritually throughout writing this thesis and my life in general.

The author also would like to acknowledge the High Performance Computing center of the University of Strasbourg for supporting this thesis by providing scientific support and access to computing resources. Part of the computing resources were funded by the Equipex Equip@Meso project (Programme Investissements d'Avenir) and the CPER Alsacalcul/Big Data.

Thesis Outline

This thesis provides a comprehensive account of metaheuristics and machine learning interplay. We give an overview of the material and the proposed methodologies at the end of all chapters (except the last) for the time-constrained readers. The thesis is organized in two parts and in a hierarchical way. Part I, composed of Chapters 1 to 5, gives the motivations for integrating machine learning techniques into the metaheuristics and introduces the problem definitions, basic concepts and proposed methodologies that will be used throughout this thesis. Part II, which comprises Chapters 6 to 7, describes the application of metaheuristics for improving the accuracy of machine learning models. Part I is likely of more immediate interest to practitioners with a background in global optimization, while Part II is likely of more interest to researchers with a background in machine learning. Part III puts the interplay of metaheuristics and machine learning models into perspective.

Part I

Machine Learning for Metaheuristics

Chapter 1

Introduction

> All science, however, commences by being strange.
>
> Victor Hugo

The metaheuristic algorithms are one of the most well-known techniques for stochastic real-world optimization tasks. The traditional approaches face issues that limit their convergence rate especially in dealing with the large-scale problems. To address this issue, different hybrid algorithms were proposed. Through this chapter we show the ability of the machine learning, when embedded to metaheuristic, to solve the optimization problems efficiently.

The use of machine learning techniques for enhancing the performance of metaheuristics has been studied by many researches over the past few decades. In [79], authors proposed to study the interest of combining metaheuristics and data mining through a short survey that enumerates the different opportunities of such combinations based on literature examples. Accordingly, they distinguish the aim of the cooperation based on reducing the computational time, significantly reducing the search space, or improving the quality of the search by introducing knowledge in operators or in other parts of the metaheuristic. This can be achieved by integrating the machine learning techniques into population initialization, fitness evaluation and selection, population reproduction and variation, algorithm adaptation, and local search components. Moreover, various machine learning techniques can be used in metaheuristics to enhance the algorithm performance; including interpolation and regression, clustering analysis, principle component analysis, orthogonal experimental design, opposition-based learning, artificial neural networks, support vector machines, case-based reasoning, reinforcement learning, competitive learning, and Bayesian network. We refer the readers to several works [2, 11, 20, 30, 71, 73, 79, 96, 154, 160, 161, 181] which provide comprehensive studies on the application of machine learning techniques to enhance metaheuristics.

While the above-mentioned works can help the optimization algorithms search more effectively, they also increase the computational burden. In recent years, the increasing computation power and the availability of Big Data have redefined the value of such approaches by utilizing advanced computation power with GPU and massive-data processing techniques. Better understanding and improving cooperation between recent machine learning models and metaheuristics therefore will play a significant role in enhancing optimization algorithms efficiently. In this direction, we propose and study the use of advanced deep learning models for optimal guidance profile of an optimization mission. These models are selected due to their efficient performance, high number of citations, their similarity to

evolutionary components, interesting interaction mechanisms between components, number of parameters, and stagnation prevention strategies. Interestingly, we also investigate the direct application of the deep learning models for the optimization task. This could be a very promising research direction which includes balancing the performance improvement and computational burden caused by the direct application of machine learning. Furthermore, we also study the integration of the existing techniques in machine learning (e.g. transfer learning [177]) into the metaheuristics so as to significantly reduce the computational burden. More importantly, we should note that the proposed algorithms are not applied only on simple numerical optimization benchmarks, but also on complex real-world applications. We do not claim to be superior to the other researchers, but hope to present some interesting and relatively novel approaches.

Chapter 2

Application I: Protein Structure Prediction

> If you wish to make an apple pie from scratch, you must first invent the universe.
>
> Carl Sagan, Cosmos

Although powerful metaheuristics have been proven effective to tackle the non-linear optimization problems, researchers are faced with the challenge of computationally expensive simulations for the protein structure prediction (PSP). This chapter introduces a new modification of differential evolution (DE) which makes use of the computationally cheap surrogate models and gene expression programming (GEP) in order to address the aforementioned issue. The incorporated GEP is used to generate a diversified set of configurations, while radial basis function (RBF) surrogate model helps DE to find the best set of configurations. In addition to this, covariance matrixadaptation evolution strategy (CMAES) is also adopted to explore the search space more efficiently. The experiments show that the proposed SGDE performs better than the state-of-the-art algorithms on the PSP problems in both terms of the convergence rate and accuracy. In the case of run time complexity, SGDE significantly outperforms the other competitive algorithms for the adopted all-atom model.

Metaheuristics give rise to a large number of studies for proposing more effective and rapid algorithms. However, there is still room for further improvement duetothegrowing-complexityofthe optimization problems. In [179], Zhang et al. presented a taxonomy of the machine learning (ML) enhanced metaheuristics in order to study the interest of such integration. The most important insight which has resulted from this research is that insertion of ML techniques usually leads to: 1) speeding up the search process and 2) improving the quality of the solutions. Accordingly, we would like to take a more detailed look on the importance of the ML techniques for PSP problem. The implementation steps are based on surrogate modeling, genetic programming [130] and the CMAES [115] algorithm which are integrated into the DE [158]. The first component known as the approximation model, meta-model or response surface model is a computationally cheap mathematical model which has been successfully employed to replace expensive fitness function f in time-demanding real-world applications. The Gaussian process/Kriging, polynomial regression (PR), RBF, radial basis neural network (RBNN) and support vector regression (SVR) are well-known meta-models which have been reported in literature [76]. They have exhibited

superior performance in solving computationally expensive dynamic, multi-objective and single-objective optimization problems. The advantages of the cheap surrogate models become clear where they could be used to explore a large area of the configuration space using far less computation time. This is due to the fact that conventional optimization algorithms are highly sensitive to their search strategies and associated control parameters and it is necessary to perform a trial-and-error search to find the best combination for the problem at hand. Consequently, fitness functions need to be frequently evaluated which results in costly computational expenses. Motivated by this aspect, we employed a surrogate model based on RBF for online configuration of DE usingless fitness evaluations. As another novel feature in DE, the second consideration replaces mutation strategies of DE by the generated ones using a GP technique. The introduced GEP enables us to have a diversified pool of mutation search operators for the individuals in the population. The GEP produces linear and non-linear extrapolation of the individuals which results in different combinations of solutions. This differs from previous studies that use a predefined set of configurations to generate competitive trial vectors. Finally, the standard DE is equipped with the CMAES algorithm in order to learn and takes into account dependencies of the optimization parameters [143]. On large scale problems with high dependencies, CMAES becomes a valuable alternative and could reduce the number of required function evaluations.

2.1 Motivation

The PSP represents the optimization problem of how to determine the 3D structure of proteins from their primary sequence. Generally speaking, it is characterized by arranging a sequence of basic elements a-helix, b-strand and coil. Determining 3D structures of a protein can affect its functions and is vitally important for rational drug design. The early works on PSP subject, i.e., X-ray crystallography and Nuclear Magnetic Resonance (NMR) were based on experimental techniques. The aforementioned methods are very expensive and time-consuming. In the case of X-ray diffraction, not all proteins can be successfully crystallized. Besides, most of the membrane proteins are difficult to crystallize and they will not dissolve in normal solvents [36]. Moreover, the question arises as to whether structure in single crystals adequately characterizes the protein conformation in a complex and dynamic environment of living cells [159]. NMR is indeed a very powerful tool in determining the 3D structures of membrane proteins, but the interpretation of NMR spectra is very complex, and the assignment of interproton distances is not always feasible. These practical limitations stimulate continual progress in the development of various structural bioinformatics tools for the PSP. The problem becomes easier if similar proteins (also referred to as templates) are found in the Protein data bank (PDB). In this case, a database of known structures could be used to find high-resolution models by aligning target sequences to the solved similar structures. Otherwise, we need to build a model from scratch without the explicit use of templates (i.e.ab initioPSP). The 3D structure obtained by homology modeling is very sensitive to the sequence alignment of the query protein with the structure-known protein (template) [148]. In addition, not all proteins with unknown structures have ideal templates. Furthermore, recent CASP10 experiment

reveals the fact that the prediction process for the hard targets with unknown templates has not been improved [95]. Up to September 2014, there had been only 100,000 proteins in PDB compared to 80 million unknown protein sequences in UniprotKB/TrEMBL [85]. This encourages the need for developing ab initio PSP methods.

A successful ab initio model depends on two components: a powerful search algorithm and an accurate potential energy function. Thelater consideration is helpful to distinguish between the native and non-native structures of the proteins. Despite significant advances in computational capabilities, the simulation cost of such energy functions is still too expensive (approximately 150 CPU days for a small protein < 100 residues) [43]. So, researchers have adopted simplified models in order to develop and test their new search algorithms. Thisincludesinstances of very simplified models [21], as well as for more complex all-atom models where the energy functions is computed based on experiments inphysics, chemistry and calculations inquantum mechanics [16, 110]. However, it should be noted that even the simplest models are still too complex from the computational point of view.

2.2 Problem Definition

Protein structure prediction (PSP) plays an important role in the field of computational molecular biology and is one of the most challenging problems; using both simplified 3D AB off-lattice model and all-atom AMBERforce field models. The search for an appropriate computational method for PSP and the protein-ligand docking has led to the development of models which make use of interaction between the atoms in order to assessthe quality of a given solution. This thesis compares performance of the SGDE and other methods from the literature using two models. The first one is 3D AB off-lattice model which takes less computational time and is useful to analyze and benchmark the behavior of the SGDE during the development process. The second model is a more complicated all-atom model under a classical force field energy function.

2.2.1 3D AB Off-lattice

This section explains how we can define three-dimensional structure of a protein by a set of bond/ torsional angles, and unit-length bonds between two amino-acids. The AB off-lattice model is among the most popular models for the PSP problem [74], according to which monomers are linked up by unit-length chemical bonds to form a structural chain in the three dimensional space. Conformation of such a chain with n amino acids will be determined by horizontal bond angles $\theta = [\theta_1, \theta_2, \cdots, \theta_{n-2}]$ and torsional angles $\beta = [\beta_1, \beta_2, \cdots, \beta_{n-3}]$. Here, θ is used to consider the angles between adjacent amino acids in XOY plane, while β takes into account the angles for corresponding amino acids and XOY plane. Thus, the position of amino acids is determined as follow [74]:

$$(x_i, y_i, z_i) = \begin{cases} (0,0,0) & \text{if } i = 1 \\ (0,1,0) & \text{if } i = 2 \\ (cos(\theta_1), sin(\theta_1) + 1, 0) & \text{if } i = 3 \\ \begin{pmatrix} x_{i-1} + cos(\theta_{i-2}) \times cos(\beta_{i-3}), \\ y_{i-1} + sin(\theta_{i-2}) \times cos(\beta_{i-3}), \\ z_{i-1} + sin(\beta_{i-3}) \end{pmatrix} & \text{if } 4 \leqslant i \leqslant n \end{cases} \tag{2.1}$$

After defining a unique structure with the above mentioned properties, the free energy of a protein can be computed. To do so, the AB off-lattice model considers the intra-molecular bending potential energy of the backbone and non-bonded interactions ; as given in Eq. 2.2.

$$E = \sum_{i=2}^{n-1} v_1(\theta_i) + \sum_{i=1}^{n-2} \sum_{j=i+2}^{n} v_2(r_{i,j}, \xi_i, \xi_j) \tag{2.2}$$

Here, v_1 is defined as follow:

$$v_1(\theta_i) = \frac{1}{4}(1 - cos(\theta_i)) \tag{2.3}$$

and the non-bonded interactions is computed using Eq. 2.4 and Eq. 2.5, which represent the Euler distance between two residues i and j.

$$v_2(r_{i,j}, \xi_i, \xi_j) = 4 \times [r_{i,j}^{-12} - \zeta(\xi_i, \xi_j) \times r_{i,j}^{-6}] \tag{2.4}$$

$$\zeta(\xi_i, \xi_j) = \frac{1}{8}(1 + \xi_i + \xi_j + 5 \times \xi_i \times \xi_j) \tag{2.5}$$

Thereby, the main task is to find parameter settings $[\theta_1, \theta_2, \cdots, \theta_{n-2}, \beta_1, \beta_2, \cdots, \beta_{n-3}]$ with a minimum free energy state.

2.2.2 AMBER Force Field

Force fields are a class of computational methods which measure the interaction energy of the atoms with each other to determine whether a model is native or not. They do so by means of the purely physics-based principles, quantum-based calculations or empirical-based fitting approaches. As the energy value reduces, native conformation is more likely to be detected. The force fields get coordinates of the atoms in 3D dimensional space and output its energy value. The geometry of chemical bonds is fixed during the PSP process and the coordinates of the proteins can be modified by changing the dihedral and torsion angles. Thus, a solution representation approach commonly used by PSP methods is based on backbone and side-chain angles. There are three backbone dihedral angles ϕ, ψ and ω. As shown in Figure 2.1, ϕ is the angle of right-handed rotation around NCA bond, ψ defined around CA-C bond and ω rotates about C-N bond. Depending on their amino acid sequence, proteins also have different number of side-chain angles namely χ_1, χ_2, χ_3 and χ_4. Having these in mind, solution representation for a conformation has been presented in Figure 2.2.

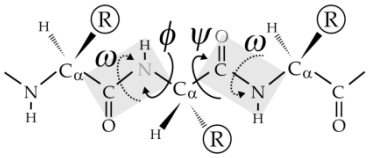


Figure 2.1: A representation of the dihedral angles ϕ(C-N-CA-C), ψ(N-CA-C-N) and ω(CA-C-N-CA)

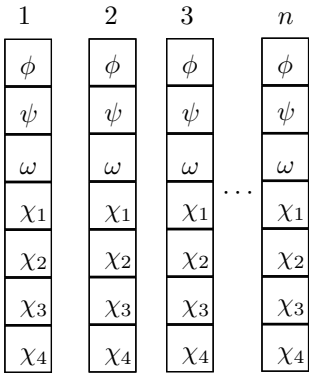


Figure 2.2: The proposed encoding representation schema for a protein with n residues using the backbone and side-chain dihedral angles

This study uses the well-known Assisted Model Building with Energy Refinement (AMBER) force field. More precisely, the amber force fields ff99SB is used to provide the protein parameters. The functional form of the AMBER force field is as follows:

$$E_{total} = \sum_{bounds} K_r \times (r - r_{eq})^2 + \sum_{angles} K_\theta \times (\theta - \theta_{eq})^2 + \sum_{dihedrals} \frac{V_n}{2}[1 + cos(n\phi - \gamma)] + \sum_{i<j}[\frac{A_{ij}}{R_{ij}^{12}} - \frac{B_{ij}}{R_{ij}^{6}} + \frac{q_i \times q_j}{\varepsilon \times R_{ij}}] \tag{2.6}$$

In Eq. 2.6, r_{eq} and θ_{eq} denote equilibration variables, K_r, K_q, V_n are force constants, A, B, and q are the non-bonded potentials, n is multiplicity and ϕ is the phase angle. Indeed, the first term computes the energy between covalently over bonded atoms. The second term denotes the energy of electron orbitals involved in covalent bonding over the bond angels. Also, the third term takes into the account the energy for twisting a bond over torsion angles. Finally, the fourth term considers the non-bonded energy between all atom pairs i and j, which can be decomposed into van der Waals and electrostatic energies. A detailed study of the other parameters can be found in [110].

2.3 Related Works

It has been frequently shown that choosing appropriate parameter settings and mutation strategy for differential evolution (DE) is not a trivial task [41, 173]. This is due to

the complex interactions between the decision variables and characteristics of the problem at hand. An inappropriate mutation strategy may lead to the so-called premature convergence or stagnation problems which resultsinatime consuming and costly optimization process.Therefore, DE variants based on parameter adaption and ensembles of mutation strategies attract increasing attention. For example, SaDE [128] uses its previous successful experiences for gradually self-adapting both trial vector generation strategies and their associated parameter settings. Interestingly, jDE [24] adds F and CR at the individual level and accordingly more promising solutions will propagate also better parameter settings. In CoDE [170], trial vector generation is based on a pool of three control parameter settings and mutation strategies. At each generation, this configuration pool is used to create three trial vectors and the best solutions goes for the selection step. In a similar way, EPSDE [117] employs an ensemble of mutations and parameter settings to improve the results of the standard DE. In another study, JADE [180] introduces a new mutation strategy based on an archive and a parameter adaption strategy. More precisely, the JADE keeps recent successful crossover and amplification values for each individual in order to generate new parameter settings. Similarly, SHADE [162] adopts a historical memory of good settings to adjust F and CR values for each individual. Remarkably, L-SHADE [163] further improves the SHADE by incorporating a linear population size reduction strategy.

The aforementioned algorithms share a common characteristic: they attempt to find the best DE's configurations for the problem at hand. Consequently, recent works tried to further improve the results by applying cheap surrogate models to adapt the best parameters using less evaluations. The SA-DE-DPS [51] incorporated the surrogates to dynamically select the best set of the amplification factor, crossover rate and population size. In another work [75], authors put forward LES-CDE which adopts an ensemble of several adjacent local surrogates to have more promising trial vectors in the crowding DE (CDE). Mallipeddi et al. [116] introduced ESMDE to enhance the performance of the EPSDE by means of a Kriging model. To further continue the research in this direction, we developed a novel approach by utilizing the surrogate models. As illustrated in Figure 2.3, the main point to distinguish the proposed approach from the previously surrogate assisted works lies in the aim of the integration.

2.4 Preliminaries

2.4.1 Radial Basis Function

In the optimization context, surrogate models (or meta-models) are a class of mathematical techniques that simulate the behavior of a computationally expensive fitness function. Given solution vector $\mathbf{x}$ and exact fitness function f, a surrogate model is represented as in 2.7, where ϵ is the approximation error.

$$\hat{f}(x) = f(x) + \epsilon \tag{2.7}$$

For an m-dimensional problem, suppose we have n observations S and $\boldsymbol{f}_s$ for a known

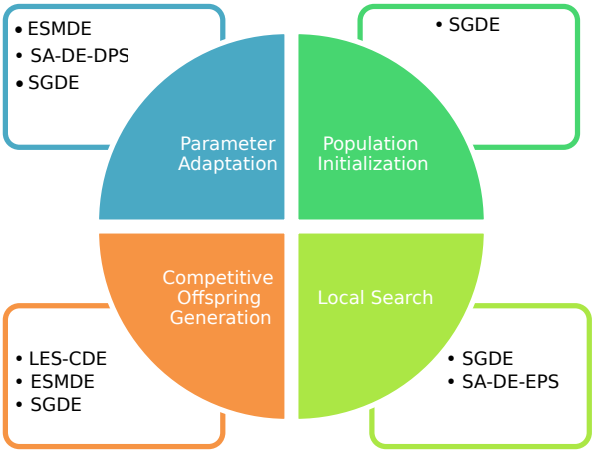


Figure 2.3: Comparing the aim of integrating surrogates into the DE algorithm based on the SA-DE-DPS, ESMDE, LES-CDE and SGDE.

function $f : \mathbb{R}^m \to \mathbb{R}$ as follows:

$$S = [\boldsymbol{x}^1, \cdots, \boldsymbol{x}^n]^\top \in \mathbb{R}^{n \times m}, \boldsymbol{x}^i = [x_1^i, \cdots, x_m^i] \in \mathbb{R}^m \tag{2.8}$$

$$\boldsymbol{f}_s = [f(\boldsymbol{x}^1), \cdots, f(\boldsymbol{x}^n)]^\top \in \mathbb{R}^n \tag{2.9}$$

In particular, a surrogate model tries to predict the fitness function f for any unseen input vector $\hat{x}$ according to the data sets $(\mathrm{S}, \boldsymbol{f}_s)$. Among different surrogate models, we utilized the RBF which is a good model for high dimensional problems [30, 31]. The RBF model is an interpolation method for scattered multivariate data which takes into account all the sample points. To do so, it adopts linear combinations of a radial function $h(\boldsymbol{x})$ to approximate a response function $\hat{f}(\hat{\boldsymbol{x}})$ as:

$$\hat{f}(\hat{\boldsymbol{x}}) = \sum_{i=1}^{n} w_i \times h(|\hat{\boldsymbol{x}} - \boldsymbol{x}^i|) + \boldsymbol{b}^\top \times \boldsymbol{x} + a \tag{2.10}$$

where w_i represents the unknown weight coefficient, $\hat{\boldsymbol{x}} \in \mathbb{R}^m$ is an unseen point, h denotes a radial basis kernel and ϵ are independent errors with variance σ^2. A function $h : \mathbb{R}^m \to \mathbb{R}$ is called a radial function if it satisfies the property $h(\boldsymbol{x}) = h(\|\boldsymbol{x}\|)$. Typical radial basis functions are linear splines, thin-plate splines, cubic splines, cubic splines, cubic splines, inverse multiquadric. Given a suitable kernel h; the unknown parameters a,$\boldsymbol{b}$, and $\boldsymbol{w}$ could be obtained by solving the following system of linear equations:

$$\begin{pmatrix} \Phi & P \\ P^T & 0 \end{pmatrix} \times \begin{pmatrix} \boldsymbol{w} \\ \boldsymbol{c} \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}_s \\ 0 \end{pmatrix} \tag{2.11}$$

In 2.11, Φ is a $n \times k$ matrix with $\Phi_{i,j} = h(\left\|\boldsymbol{x}^i - \boldsymbol{x}^j\right\|_2)$, $\boldsymbol{c} = (\boldsymbol{b}^\top, a)^\top$ and

$$P^{\top} = \begin{pmatrix} x^1 & \cdots & x^n \\ 1 & \cdots & 1 \end{pmatrix} \tag{2.12}$$

2.4.2 Differential Evolution

In evolutionary computation, DE finds a solution by iterative improvement of a candidate solution with regard to a given measure of quality. It is one of the most powerful optimization tools that operate on the basis of the same developmental process in evolutionary algorithms. Nevertheless, different from traditional evolutionary algorithms, DE uses the scaled differences of vectors to produce new candidate solutions in the population. Hence, no separate probability distribution should be used to perturb the population members. The DE is also characterized by the advantages of having few parameters and ease of implementation. The application of DE on engineering and biomedical studies has attracted a high level of interest, concerning its potential [42].

Basically, DE algorithm works through a particular sequence of stages. First, it creates an initial population sampled uniformly at random within the search bounds. Thereafter, three components namely mutation, crossover and selection are adopted to evolve the initial population. The mutation and crossover are used to create new solutions, while selection determines the solutions that will breed a new generation. The algorithm remains inside a loop until stopping criteria are met. In the following, we explain each stage separately in details.

Like other optimization algorithms, DE starts with a randomly initialized population of parameter vectors, the so-called individuals. Each such individual represents an m-dimensional vector of decision variables. The i^{th} individual of the population for a m-dimensional optimization problem can be denoted as follows:

$$\boldsymbol{x}^i = [x_1^i, \cdots, x_m^i] \in \mathbb{R}^m \tag{2.13}$$

For each individual, the values of the decision variables should be restricted to their lower bounds $\boldsymbol{lb} = [lb_1, \cdots, lb_m]$ and upper bounds $\boldsymbol{ub} = [ub_1, \cdots, ub_m]$. Once initialization search ranges have been determined, DE assigns each individual a value from within the specified range as in Eq.2.14:

$$x_j^i = lb_j + r \times (ub_j - lb_j); i = 1, \cdots, n; j = 1, \cdots, m \tag{2.14}$$

where $r \in [0, 1]$ represents a uniformly distributed random number and n denotes the population size. After initialization, mutation operator produces new solutions by forming a mutant vector with respect to each parent individual (target vector). For each target vector, its corresponding mutant vector can be generated by different mutation strategies. Each strategy employs different approaches to make a balance between the exploration and exploitation tendencies. For the i^{th} target vector, the five most well-known mutation strategies are presented as follows [158]:

$$\boldsymbol{v}^i = \boldsymbol{x}^{r1} + F \times (\boldsymbol{x}^{r2} - \boldsymbol{x}^{r3}) \tag{2.15}$$

$$\boldsymbol{v}^i = \boldsymbol{x}^{best} + F \times (\boldsymbol{x}^{r1} - \boldsymbol{x}^{r2}) \tag{2.16}$$

$$\boldsymbol{v}^i = \boldsymbol{x}^i + F \times (\boldsymbol{x}^{best} - \boldsymbol{x}^i) + F \times (\boldsymbol{x}^{r1} - \boldsymbol{x}^{r2}) \tag{2.17}$$

$$\boldsymbol{v}^i = \boldsymbol{x}^{best} + F \times (\boldsymbol{x}^{r1} - \boldsymbol{x}^{r2}) + F \times (\boldsymbol{x}^{r3} - \boldsymbol{x}^{r4}) \tag{2.18}$$

$$\boldsymbol{v}^i = \boldsymbol{x}^{r1} + F \times (\boldsymbol{x}^{r2} - \boldsymbol{x}^{r3}) + F \times (\boldsymbol{x}^{r4} - \boldsymbol{x}^{r5}) \tag{2.19}$$

Here, $r1, r2, r3, r4, r5 \in [1, \cdots, n]$ are five different randomly generated integer numbers. Furthermore, $F \in (0, 2]$ is a scaling factor affecting the difference vector and $best \in [1, n]$ is the index of the best individual. The presented strategies in Eqs. 2.15 to 2.19 are called DE/rand/1, DE/best/1, DE/rand-to-best/1, DE/best/2 and DE/rand/2, respectively.

In the next step, DE applies a discrete crossover approach to each pair of the target vector and its corresponding mutant vector. The basic version of DE incorporates the binomial crossover defined as follows [158]:

$$u_j^i = \begin{cases} v_j^i & if\ r_j^i \leqslant CR \\ x_j^i & otherwise \end{cases} \tag{2.20}$$

In Eq. 2.20, $CR \in (0, 1]$ is the user-specified crossover rate which determines the probability of mixing between parent and mutant vectors. Also, r_j^i is a randomly picked float number $\in (0, 1]$ that should be regenerated for each individual and each dimension.

Finally, DE adopts a selection mechanism to choose the best individuals according to their fitness for producing the next generation of population. To this end, it compares performance of the trial and target vectors and copies the best one into the next generation; as presented in Eq. 2.21. Here, f is the objective function that should be minimized.

$$x^i = \begin{cases} u^i & if\ f(u^i) \leqslant f(x^i) \\ x^i & otherwise \end{cases} \tag{2.21}$$

2.5 Methodology

Here, we provide a comprehensive discussion on the details of the implementation steps in SGDE. In practice, there are two issues that should be addressed when designing a surrogate-assisted algorithm like SGDE. The first one is how to combine both approximated and original fitness values to prevent the algorithm from being misled by a false minimum introduced by the surrogates (i.e., model management) [76]. Whether building a local or global surrogate model is another issue to be addressed regarding the high dimension of the search space. We tried to tackle the aforementioned problems by incorporating both the local and global surrogate models in different stages of the evolution including population initialization, offspring reproduction and parameter adaptation. A generic framework for the surrogate-assisted SGDE algorithm includes some basic steps:

1. Initial sampling using design of experiments (DoE)
2. Population initialization using stochastic response surface (SRS)
3. Offspring reproduction by means of GEP
4. Configuration selection using local surrogate models
5. Parameter adjustment using a global surrogate model
6. Training set update for the surrogate model
7. DE termination if some stopping criteria are satisfied and going to Step 8; otherwise going to Step 3
8. CMAES executing on the best found solution

The first step samples a population of individuals using the Latin hypercube sampling (LHS) design method [156]. Next, the solutions are evaluated in terms of expensive objective functions. These will be archived for training and building an initial surrogate model. The model and a Stochastic Response Surface (SRS) method [139] are then used to initialize a population for DE. Thereafter, the population is evolved using a new introduced offspring reproduction strategy in SGDE, followed by a parameter adaptation strategy. Next, the training set will be updated. In the next phase, CMAES is also applied to the best known solution by DE. A detailed description of the above components is presented as follows.

2.5.1 Initial Sampling

Generally speaking, optimization algorithms are subjected to *curse of dimensionality* which makes them unsuitable for high dimensional problems. To this fact, DoE methods are used to maximize the amount of information across the design space. This enables us to build a global model of fitness landscapes with a minimum number of samples. Currently, orthogonal array design (OAD), uniform design (UD) and LHS are among the most widely applied fractional factorial DoE methods. The proposed SGDE adopts LHS according to which projections of the generated points onto each variable axis should be uniform. The main advantage of the LHS is that it does not need more samples for more dimensions. Researchers introduced different criteria for LHS to address poor space filling properties. In this work, we adopt a Point-distance criterion. All the solutions obtained by LHS will then be evaluated using the computationally expensive function. Next, these solutions and their fitness values will be archived for training the surrogate model. This is necessary as the RBF model needs to be trained before it can be used for the approximation purpose.

2.5.2 Population Initialization

As a novel and effective component, SGDE adopts the SRS optimization method for population initialization. The obtained solutions by the SRS are then considered as the initial population for the DE algorithm. The main idea is to put forward a computationally cheap initialization method which is able to preserve diversity in more than one promising region. By adopting SRS, only a small part of the computational budget is used to find the promising regions and in most cases we employ the surrogate model. The SRS employs an

optimal response for a surface (output variable) which is influenced by several explanatory variables (input variables). To do so, it uses a collection of mathematical and statistical information from surrogates and exact fitness function [139]. The SRS implementation consists of several steps; as presented in Algorithm 1. Here, f is a continuous function defined on a compact hypercube $[\boldsymbol{lb}, \boldsymbol{ub}] \subseteq \mathbb{R}^m$, m is the number of decision variables, n_0 is the initial number of generated points and Λ contains the obtained solutions by the SRS.

Algorithm 1: Population initialization procedure

Result: n initialized solutions
$\tau \leftarrow [\boldsymbol{x}^1, \cdots, \boldsymbol{x}^{n_0}]$
$t \leftarrow n_0$
while $t \leqslant n$ **do**
  $\Lambda_t \leftarrow \{(x^i, f(x^i)) : i = 1, \cdots, t\}$
  train the surrogate model using Λ_t
  $\Delta \leftarrow$ generate random solutions
  use the surrogate model to evaluate Δ
  $x^{t+1} \leftarrow$ best approximated solution $\in \Delta$
  $t \leftarrow t + 1$
end

Considering the above explanations and Algorithm 1, a few remarks are presented as follows. First, it should be noticed that the initial points come from the LHS. Second, the surrogate model can be RBF, Kriging, RBNN, SVR or any other type of function approximation models. Moreover, the meaning of the word "random" is very specifically related to the original study [139].

2.5.3 Offspring Reproduction

It has been frequently testified that incorporating multiple search strategies can greatly improve the performance of DE [32]. Depending on the characteristics of the problem, they could perform better during different stages of evolution than a single strategy. Based on this finding, SGDE generates competitive offspring for each individual in the population by means of different mutations. Maintaining diversity can help DE to improve the performance of its search operators. Hence, the SGDE is extended to the case of an arbitrary number of mutation strategies. This differs from previous studies that use a predefined set of search operators. The proposed diversification process begins with an initializing step in which the SGDE generates a random population of the chromosomes using the GEP for each individual. These chromosomes denote our mutation strategies and are generated using heuristic information of the DE. Considering the search operators of DE presented by Eqs. 2.15 to 2.19, the chromosomes are based on a mathematical combination of $\boldsymbol{x}^{r1}, \boldsymbol{x}^{r2}, \boldsymbol{x}^{r3}, \boldsymbol{x}^{r4}, \boldsymbol{x}^{r5}, \boldsymbol{x}^{best}, \boldsymbol{x}^i$ and F. Indeed, the introduced diversification schema only changes the way that search operators process information. The new generated chromosomes are then decoded into tree based programs and we perform a breadth-first search method to obtain the mathematical formula. During this process, a specific part of each chromosome might be useless (the useful part is called open reading frame and the other part is referred to as non-coding region). An example of such schema is illustrated in

Figure 2.4. The mentioned mutation strategies are then employed to generate trial vectors for each individual. In this approach, efficiency of the generated configurations for each individual is computed based on the approximated objective value for the generated trial vectors. The GEP takes advantages of these cheap approximated values in order to adaptively evolve mutation strategies.

Subsequently, an effective model management procedure is required to build the surrogate models when the *curse of dimensionality* emerges. To alleviate this difficulty, we involve local surrogate models which are capable of generating more accurate fitness values. Furthermore, they are relatively fast and take into account the most important information of the closest neighbors. To this fact, SGDE builds separate local surrogate models for each trial vector. It builds the local surrogates on the basis of the idea that training set for each trial vector should not lie too far from it. The SGDE selects k-nearest neighbors from the archive in order to build the local surrogate models. The generated surrogate models are then used to evaluate the corresponding trial vectors.

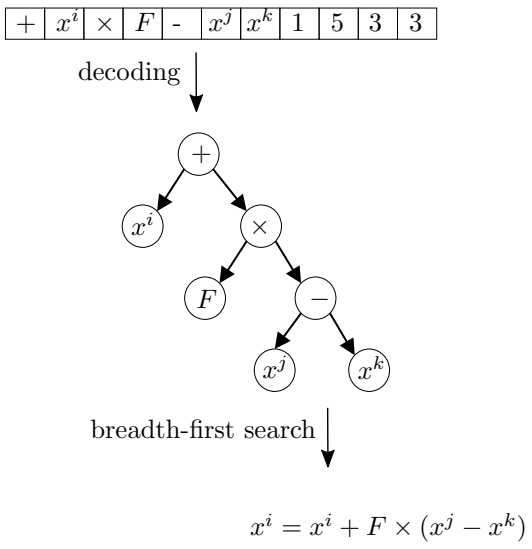


Figure 2.4: A schematic of the proposed GEP based search operators generation.

In SGDE, the estimated objective value is not the only one considered, and several other criteria are used to find the best set of generated trial vectors for the individuals. The rules of the SGDE scheme are given as follows: between any two generated trial vectors for an individual, the solution with better approximated objective value which is evaluated with more accurate surrogate model, and which is also closer to its corresponding individual, is preferred. The accuracy of the models here is calculated using the cross-validation error, while the Euclidean distance is used to measure the distance. To measure the accuracy, the suggested training data set is randomly split into K equal subsets with n data points. Then, each subset is removed in turn from the training data set and the surrogate model is updated using the $K - 1$ remaining data. Each time, all the solutions in the removed

subset are predicted using the model and the cross-validation error is calculated as in [166].

2.5.4 Parameter Adaptation

Besides the search operators, success rate of DE depends on appropriately tuning its control parameters. Although using a trial-and-error scheme seems to be a good solution, the main challenge lies in high computational costs. We use the feedback from the surrogate model to overcome such inconvenience and to guide the parameter adaptation. The proposed approach adaptively optimizes the F and CR parameters by incorporating an embedded standard DE algorithm. First, we train a global RBF kernel based on all the collected data in the archive. Next, a population of solutions is directly optimized using this surrogate for t generations without requiring any expensive function evaluations. Particularly, the optimization process runs multiple times with different configuration. We consider different lower and upper bounds for each individual in the population. So, the DE algorithm here tries to explore the search space around inside only a small interval. More precisely, for each individual a confidence interval $[-\vartheta, \vartheta]$ is adopted. The introduced strategy tries to pre-screen the most promising configurations using a full factorial experiment. Such an experiment allows us to take into account all possible combinations of the parameters. We constructed a design table for a three-level full factorial in two factors (i.e., F and CR). The configuration which leads to the best results will survive.

The adapted parameters have a $lifetime$ which represents the number of generation that a configuration can survive. The $lifetime$ property depends on the quality of the global surrogate model; the higher the accuracy, the longer the parameter will be used. Similar to the previous section, the accuracy here is computed by the cross-validation error. Whenever the $lifetime$ of the parameters ends, they will be re-configured. Accordingly, the $lifetime$ for a given configuration is determined as the multiplication of the cross-validation error ε_{cv} and a constant $\gamma > 1$.

2.5.5 Updating the Surrogate Model

The SGDE use a training data set to archive the solutions evaluated by the exact fitness function. This data set is supposed to be employed for both the global and local surrogate models. For this reason, the training set must be properly updated to ensure that the created models can cover a slightly larger region, but still be most relevant to the current solutions. Furthermore, using all the evaluated solutions will increase the computational time. To tackle this, we adopted a linear decreasing rule which reduces the archive size linearly according to the number of fitness evaluations (i.e., FEs). After each generation t, the archive size Λ_{t+1} is updated as follows:

$$\Lambda_{t+1} = (\frac{min - max}{FEs}) \times t + max \tag{2.22}$$

In Eq. 2.22, The min and max are the minimum and maximum values for the archive size, respectively. Whenever $\Lambda_{t+1} < \Lambda_t$, the worst-ranking individuals will be deleted from the archive.

2.5.6 Applying CMAES

Considering SGDE, it should be remarked that the generated mutation strategies by GEP focus on multimodal rugged search landscapes which is actually a limitation of the reliability of the SGDE. So, we employed CMAES approach to make the SGDE also computationally viable for large scale smooth problems. To do so, it is applied to the best known solution provided by DE. If the solution found by CMAES is better than the solution obtained in the exploration phase by DE, then it replaces the old one; otherwise, it is discarded. This algorithmic choice generalizes the behavior of the algorithm; regarding properties of the fitness landscape.

2.6 Experiments

2.6.1 Benchmark Sets

In this section, performance of the SGDE on protein sequences from PDB using the elaborated models is investigated. The detailed information about these proteins is presented in Table 2.1. To provide a fair comparison procedure, we select amino-acid sequences which have been frequently used to benchmark new algorithms for the PSP. The free energy state of the presented sequences should be minimized. For the AB off-lattice model, the hydrophobic and hydrophilic characteristics of the amino-acids are denoted as follows: D, E, F, H, K, N, Q, R, S, T, W, Y fall into B residues and I, V, L, P, C, M, A, G belong to A residues. The SGDE is implemented in Matlab under Windows 7 operating system. To implement the RBF, we used the SURROGATES toolbox which is a general-purpose library for the surrogate models [119].

2.6.2 Baselines

First, the proposed algorithm is compared with the basic ABC [84], DE, PSO [86] and CMAES optimization algorithms. Thereafter, a comparison study between the SGDE and state-of-the-art algorithms from the literature is conducted. In this comparison, the following algorithms are included: DE_{pfo}, jDE (DE/best/1), L-SHADE, SaDE, CoDE, JADE and EPSDE. The results for the DE_{pfo}, jDE (DE/best/1) and L-SHADE are directly taken from [11].

2.6.3 Experimental Settings

In DE, we set F to 0.5, population size to 100 and CR to 0.9. In PSO, population size is 100, $C1$ and $C2$ coefficients are 1.8, and inertia weight is 0.6. The parameter configurations of ABC and CMAES are set according to [74, 100]. To reduce stochastic behavior of the random population initialization, the results of the algorithms over 30 runs are considered. The parameters of the other competitive algorithms are set according to their original works. Moreover, the number of function evaluations for each conducted experiment is explicitly reported. Furthermore, we adopt the same implementation and parameter configuration

Table 2.1: The protein sequences used in this study along with their properties.

PDB ID	m	Protein sequence
1CB3	21	BABBBAABBAAAB
1BXL	27	ABAABBAAAAABBABB
1EDP	29	ABABBAABBBAABBABA
2H3S	45	AABBAABBBBBABBBABAABBBBBB
2KGU	63	ABAABBAABABBABAABAABABABA BABAAABBB
1TZ4	69	BABBABBAABBAAABBAABBAABAB BBABAABBBBBB
1TZ5	69	AAABAABAABBABABBAABBBBAAB BBABAABBABBB
1AGT	71	AAAABABABABABAABAABBAAABB ABAABBBABABAB
1CRN	87	BBAAABAAABBBBBAABAAABABAA AABBBAAAAAAAABAAABBAB
1HVV	145	BAABBABBBBBBAABABBBABBABB ABABAAAAABBBABAABBABBBABB AABBABBAABBBBBAABBBBBABBB
1GK4	163	ABABAABABBBBABBBABBABBBBA ABAABBBBBAABABBBABBABBBAA BBABBBBBAABABAAABABAABBBB AABABBBBA
1EWH	191	AABABAAAAAAABBBAAAAAABAAB AABBAABABAAABBBAAAABABAAA BABBAAABAAABAAABAABBAABAA AAABAAABABBBABBAAABAABA

for the SRS and RBF as suggested in [139]. The rest of the parameters are tuned and presented in Table 2.2.

2.6.4 Results and Discussion

First, we analyzed performance of the SGDE in order to give an insightful view for the influence of the introduced GEP schema on the algorithm's efficiency. To do so, we conducted 30 runs based on 1CB3 and 1BXL amino-acid sequences using the AB off-lattice model. The stopping condition is determined as 50,000 function evaluations. The configurations of SGDE are equal to those in Table 2.2. The obtained results are shown in Figures 2.5 and 2.6. In these figures, the SGDE* is a version of SGDE which only adopts the mutation strategies of the DE presented in Eqs. 2.15 to 2.19, while SGDE uses a diversified pool of mutation strategies for each individual. The depicted results show that SGDE performs better and obtained more stable results. In contrast to SGDE*, the introduced algorithm evolves a pool of effective trial vector mutation strategies for individuals based on their previous experience. Accordingly, unfavorable mutation strategies of less effective results will be discarded in later generations.

Next, the performance of SGDE for PSP is compared with both the standard and improved algorithms in the literature. The performance metrics are reported in Tables 2.3 and 2.4, while Table 2.5 presents the best solution vectors by SGDE for the problem at hand. In these experiments, the number of function evaluations is set to $200,000$ for the standard algorithms, improved versions and for the SGDE. In SGDE, 1% of the computational budget

Table 2.2: The adopted parameters for the proposed SGDE algorithm.

Parameter	Value
number of generated mutant vector by GEP	10
mutation rate in GEP	15
selection rate in GEP	0.8
head length in GEP	50
function set in GEP	{×,+,-}
population size	100
min	100
max	200
n_0	$2 \times (m+1)$
n	1000
k	15
K	4
γ	1.5

is used by SRS, 50% by the DE and 49% by CMAES. In Table 2.4, $\text{DE}^{\star}_{pfo}$ and $\text{SGDE}^{\dagger}$ denote the results with $100,000$ function evaluations for DE_{pfo} and SGDE, respectively. As previously mentioned, major contribution of the proposed SGDE approach is using the GEP and surrogate modeling to replace in part the original computationally expensive solver. To this fact, results after $100,000$ evaluations are also reported in Table 2.4 to investigate whether SGDE algorithm is able to take advantages of the introduced surrogate-based schema. We considered DE_{pfo} and SGDE because they yield best performances in terms of the solution accuracy and convergence rate.

The values of best, worst, mean, median and standard deviation of the energy functions are presented for the purpose of the comparison. Furthermore, the percentage of improvement (PI) for the SGDE in comparison to each of the other algorithms based on the mean value is also collected. The corresponding results are given in Tables 2.3 and 2.4. In Table 2.3, we considered ABC and PSO as two algorithms with different properties from that of SGDE, and CMAES and DE as the basic components of the proposed algorithm. Regardless of SGDE, it can be seen that ABC provides better mean values for 1CB3, 1TZ4, 1HVV, 1GK4; CMAES for 2KGU, 1TZ5, 1AGT, 1CRN; and finally DE for 1BXL, 1EDP, 2H3S. These results indicate how algorithm scalability affects their performances. For example, it is clear that the performance of DE decreased by increasing the dimension of the problem at hand. Conversely, CMAES shows a better performance in the case of high dimensional problems. From Table 2.3, however, we can see that the enhanced algorithm outperforms all the standard algorithms on both the low and high dimensional problems. The reported PI values also confirm the superiority of the SGDE. It can be explained by the adopted surrogate models and GEP technique which reduce the probability of the exploration around already explored search areas.

Table 2.3: Comparison of the results between the standard algorithms and the SGDE (Std.: standard deviation).

PDB ID	Algorithm	Mean	Std.	Best	PI

Table 2.3: Comparison of the results between the standard algorithms and the SGDE (Std.: standard deviation).

PDB ID	Algorithm	Mean	Std.	Best	PI
1CB3	ABC	-4.296455	0.439499	-5.268011	29.30%
	DE	-1.866129	0.845961	-4.659330	69.29%
	PSO	-3.203742	2.084507	-5.776234	47.28%
	CMAES	-2.503826	2.056343	-7.778162	58.80%
	SGDE	-6.077217	1.754122	-8.369052	
1BXL	ABC	-9.446039	0.655434	-11.268678	35.69%
	DE	-11.294667	1.028064	-12.648686	23.11%
	PSO	-9.710913	2.753267	-11.933259	33.89%
	CMAES	-7.734668	3.440012	-15.362006	47.35%
	SGDE	-14.689404	1.839411	-16.478776	
1EDP	ABC	-6.121525	0.880439	-8.253825	32.59%
	DE	-7.233354	2.695314	-11.316626	20.34%
	PSO	-4.307987	1.113684	-5.711125	52.56%
	CMAES	-4.609754	2.726812	-9.337707	49.24%
	SGDE	-9.080840	1.857780	-13.145400	
2H3S	ABC	-7.341647	0.893442	-10.199716	41.91%
	DE	-7.443750	1.227711	-9.292054	41.10%
	PSO	-4.618970	1.955581	-6.897306	63.45%
	CMAES	-7.368380	4.120026	-14.856206	41.70%
	SGDE	-12.637971	2.861910	-17.303680	
2KGU	ABC	-19.223037	1.562318	-22.130853	50.38%
	DE	-18.511796	6.859623	-25.403276	52.21%
	PSO	-12.471775	5.963385	-21.131556	67.81%
	CMAES	-19.605122	8.453813	-29.880254	49.39%
	SGDE	-38.738349	4.606122	-46.091724	
1TZ4	ABC	-12.582645	1.410564	-15.867712	47.84%
	DE	-8.448939	2.895693	-11.970046	64.97%
	PSO	-5.843409	3.655283	-9.194886	75.78%
	CMAES	-11.486689	4.331165	-18.983567	52.38%
	SGDE	-24.122570	4.058022	-31.503100	
1TZ5	ABC	-15.313003	1.316238	-18.879996	48.56%
	DE	-12.045163	6.590958	-18.610951	59.53%
	PSO	-8.755463	4.135149	-14.293049	70.59%
	CMAES	-15.702593	5.638691	-29.306050	47.25%
	SGDE	-29.766762	4.581026	-39.053634	

Table 2.3: Comparison of the results between the standard algorithms and the SGDE (Std.: standard deviation).

PDB ID	Algorithm	Mean	Std.	Best	PI
1AGT	ABC	-22.568844	1.942112	-25.652326	44.46%
	DE	-13.522272	5.734693	-21.687050	66.72%
	PSO	-18.835705	7.774478	-26.380818	53.65%
	CMAES	-23.411478	8.961818	-34.457463	42.39%
	SGDE	-40.634420	4.219346	-46.229500	
1CRN	ABC	-38.261975	2.804349	-41.290523	40.46%
	DE	-18.963015	9.991839	-35.898240	70.49%
	PSO	-23.902511	3.355806	-27.545842	62.80%
	CMAES	-40.975829	13.444372	-60.939240	36.23%
	SGDE	-64.258945	7.687089	-78.245070	
1HVV	ABC	-21.609320	2.569676	-27.353049	43.76%
	DE	-3.241457	13.398311	-11.795201	91.56%
	PSO	4.497784	6.747108	-1.060249	111.71%
	CMAES	-18.988722	3.968934	-27.915393	50.58%
	SGDE	-38.422210	5.975514	-52.558824	
1GK4	ABC	-27.836994	2.267989	-32.729077	40.75%
	DE	11.778927	8.120520	-1.4981720	125.61%
	PSO	-6.571952	9.798663	-20.691791	86.01%
	CMAES	-25.297559	3.207333	-41.108156	46.16%
	SGDE	-46.984386	3.969936	-57.965434	

Thereafter, we collect the results of the recently proposed algorithms for the PSP in order to provide a more comprehensive study. The best, mean and standard deviation of the algorithms are presented for the purpose of the comparison. Table 2.4 shows the performance of the algorithms for all the sequences over 30 runs. This table denotes the superior accuracy of the SGDE for all the presented test sequences in terms of mean, standard deviation, and the best results. Interestingly, we can also see that SGDE† outperforms other propositions for all the problems when considering the mean values. It is also evident that the introduced cheap algorithms fairly provide comparable results when considering the best obtained solutions. Surprisingly, SGDE† outperforms the other algorithms on the 2H3S, 1TZ4 and 1TZ5 protein sequences by considering the best obtained value. This is due to the fact that the incorporated local and global surrogate models

Table 2.4: Comparison of the results between the improved algorithms and the SGDE (Std.: standard deviation).

PDB ID	Algorithm	Mean	Std.	Best	PI
	DEpfo	-5.588400	1.960300	-8.369000	8.04%
	$DE^{\star}_{pfo}$	-4.539159	2.314210	-8.116150	25.31%
	jDE(DE/best/1)	-3.898800	2.443700	-8.198300	35.85%

Table 2.4: Comparison of the results between the improved algorithms and the SGDE (Std.: standard deviation).

PDB ID	Algorithm	Mean	Std.	Best	PI
1CB3	L-SHADE	-2.791600	2.106800	-8.115100	54.06%
	SaDE	-3.487625	2.306001	-5.933400	42.61%
	CoDE	-5.585840	0.677920	-6.774100	8.09%
	JADE	-5.391650	0.423160	-6.394700	11.28%
	EPSDE	-4.918070	0.383574	-5.822700	19.07%
	SGDE	-6.077217	1.754122	-8.369052	
	SGDE†	-5.765060	1.081309	-8.116200	5.14%
1BXL	DEpfo	-12.610400	2.530600	-16.344300	14.15%
	DE$^{\star}_{pfo}$	-11.862654	2.510254	-15.833700	19.24%
	jDE(DE/best/1)	-12.404700	2.491300	-16.010100	15.55%
	L-SHADE	-10.542800	2.871200	-14.201500	28.23%
	SaDE	-11.208780	0.809479	-12.345200	23.69%
	CoDE	-11.907830	2.605866	-15.447600	18.94%
	JADE	-11.384120	0.765585	-12.732600	22.50%
	EPSDE	-10.530370	0.877880	-12.178400	28.31%
	SGDE	-14.689404	1.839411	-16.478776	
	SGDE†	-14.436030	1.174347	-16.223800	1.72%
1EDP	DEpfo	-8.666600	2.560300	-13.562000	13.03%
	DE$^{\star}_{pfo}$	-8.495471	3.339964	-13.428000	14.75%
	jDE(DE/best/1)	-7.466700	2.937600	-11.988000	25.07%
	L-SHADE	-4.590000	3.217800	-11.697700	53.94%
	SaDE	-5.392990	2.666704	-9.946400	45.88%
	CoDE	-8.087580	3.861207	-13.527000	18.84%
	JADE	-7.120270	0.892819	-8.673800	28.55%
	EPSDE	-6.200950	0.585151	-7.307400	37.77%
	SGDE	-9.964899	2.623943	-14.292856	
	SGDE†	-9.080840	1.857780	-13.145400	8.87%
1H3S	DEpfo	-10.676700	2.751800	-16.503000	15.52%
	DE$^{\star}_{pfo}$	-12.111503	2.982058	-17.287400	4.17%
	jDE(DE/best/1)	-10.793100	2.786400	-16.692000	14.60%
	L-SHADE	-10.383000	2.627300	-15.668700	17.84%
	SaDE	-6.265260	1.385776	-8.454000	50.43%
	CoDE	-9.679260	2.438849	-14.432800	23.41%
	JADE	-8.240400	1.049730	-9.997900	34.80%
	EPSDE	-5.108480	0.924547	-6.755500	59.58%
	SGDE	-12.637971	2.861910	-17.303680	
	SGDE†	-12.539310	2.823182	-17.046000	0.78%
	DEpfo	-35.385000	4.701300	-44.336900	8.66%
	DE$^{\star}_{pfo}$	-28.521353	4.771998	-37.74590	26.37%
	jDE(DE/best/1)	-29.551100	5.374000	-40.503500	23.72%
	L-SHADE	-26.628200	2.907100	-35.070700	31.26%

Table 2.4: Comparison of the results between the improved algorithms and the SGDE (Std.: standard deviation).

PDB ID	Algorithm	Mean	Std.	Best	PI
2KGU	SaDE	-13.514080	1.496921	-16.036900	65.11%
	CoDE	-24.925000	4.086078	-30.124600	35.66%
	JADE	17.218310	0.993447	-19.218100	144.45%
	EPSDE	-8.360970	0.666336	-9.341700	78.42%
	SGDE	-38.738349	4.606122	-46.091724	
	SGDE†	-38.201660	6.536553	-44.284300	1.39%
	DEpfo	-20.436100	5.279800	-30.921100	15.35%
	DE$^{\star}_{pfo}$	-19.293730	4.088801	-25.517600	20.09%
	jDE(DE/best/1)	-16.913500	3.885100	-24.300000	29.94%
	L-SHADE	-16.469300	2.896300	-20.221600	31.78%
1TZ4	SaDE	-8.634780	1.771702	-11.188000	64.23%
	CoDE	-11.740680	4.679727	-18.023000	51.37%
	JADE	-10.376180	1.692243	-13.829800	57.02%
	EPSDE	-1.814920	0.893272	-3.3096000	92.48%
	SGDE	-24.142960	6.107640	-31.503100	
	SGDE†	-24.122570	4.058022	-31.503100	0.08%
	DEpfo	-27.341200	4.084700	-38.186800	8.15%
	DE$^{\star}_{pfo}$	-23.652170	3.674546	-31.400700	20.54%
	jDE(DE/best/1)	-20.365500	3.837800	-30.127900	31.58%
	L-SHADE	20.640300	3.116300	-34.311500	169.34%
1TZ5	SaDE	-14.617780	2.796681	-19.886800	50.89%
	CoDE	-19.165230	3.451774	-24.744500	35.62%
	JADE	-13.169800	1.505524	-15.702900	55.76%
	EPSDE	-4.124440	0.638126	-5.020000	86.14%
	SGDE	-29.766762	4.581026	-39.053634	
	SGDE†	-29.486118	4.296542	-38.626900	0.94%
	DEpfo	-39.026800	5.344600	-50.631100	5.78%
	DE$^{\star}_{pfo}$	-36.684787	6.323074	-48.854200	11.44%
	jDE(DE/best/1)	-30.777000	6.309000	-42.992600	25.70%
	L-SHADE	-29.356400	2.684600	-39.316800	29.13%
1AGT	SaDE	-14.187640	1.487899	-15.680800	65.75%
	CoDE	-28.921060	4.867545	-35.709000	30.18%
	JADE	-19.432300	0.622762	-20.246100	53.09%
	EPSDE	-8.914320	1.135966	-10.824500	78.48%
	SGDE	-41.422991	6.285417	-54.362306	
	SGDE†	-40.634420	4.219346	-46.229500	1.90%
	DEpfo	-60.244400	7.577200	-74.406800	6.25%
	DE$^{\star}_{pfo}$	-53.743007	6.253436	-68.836100	16.36%
	jDE(DE/best/1)	-46.903000	7.424300	-63.713800	27.01%
	L-SHADE	-46.960400	3.768300	-60.237100	26.92%

Table 2.4: Comparison of the results between the improved algorithms and the SGDE (Std.: standard deviation).

PDB ID	Algorithm	Mean	Std.	Best	PI
1CRN	SaDE	-22.044860	1.553122	-24.751700	65.69%
	CoDE	-44.234280	4.972664	-49.020300	31.16%
	JADE	-30.053940	1.572330	-32.762100	53.23%
	EPSDE	-9.730750	1.139082	-12.381900	84.86%
	SGDE	-64.258945	7.687089	-78.245070	
	SGDE†	-60.634070	6.462495	-70.030400	5.64%
	DEpfo	-34.805900	5.292600	-44.726400	9.41%
	$DE^{\star}_{pfo}$	-32.351140	4.849457	-41.921100	15.80%
	jDE(DE/best/1)	-20.954100	7.642400	-31.587800	45.46%
	L-SHADE	-25.491000	1.709000	-28.778700	33.66%
1HVV	SaDE	-13.775810	6.625543	-21.356300	64.15%
	CoDE	-21.148700	4.558295	-26.106000	44.96%
	JADE	-8.771650	1.124914	-10.729700	77.17%
	EPSDE	18.788040	1.614203	15.689900	148.90%
	SGDE	-38.422210	5.975514	-52.558824	
	SGDE†	-38.201660	6.536553	-44.284300	0.57%
	DEpfo	-44.859100	4.722700	-52.065100	4.52%
	$DE^{\star}_{pfo}$	-40.455554	5.350739	-47.983600	13.90%
	jDE(DE/best/1)	-22.321800	7.416900	-35.677900	52.49%
	L-SHADE	-32.908200	2.210800	-40.265500	29.96%
1GK4	SaDE	-19.701000	5.829114	-25.855100	58.07%
	CoDE	-27.696075	6.414348	-35.723300	41.05%
	JADE	-9.812420	1.007794	-11.348200	79.12%
	EPSDE	12.030714	9.152999	1.614203	125.61%
	SGDE	-46.984386	3.969936	-57.965434	
	SGDE†	-46.859100	5.142473	-52.024740	0.27%

help the SGDE to walk around the potentially promising regions and avoid unnecessary computation cost in non-promising regions. Furthermore, the GEP algorithm enables the SGDE to learn the utility of adopting the best mutation strategies which leads to a proper balance between the exploration and exploitation.

In Table 2.4, PI of the different algorithms is also presented. From this table, we can see that the introduced method needs less fitness evaluations to attain a similar or better result. More precisely, if we compute the average improvement of the algorithms over 11 problems as depicted in Figure 2.7, we can see that the overall percentage improvement for the SGDE is always positive.

To evaluate the stability between the algorithms, a rank based analysis is presented. This procedure determines the most effective algorithm for each protein sequence by ranking them from the best to the worst. The rank procedure assigns rank 1 to the algorithm

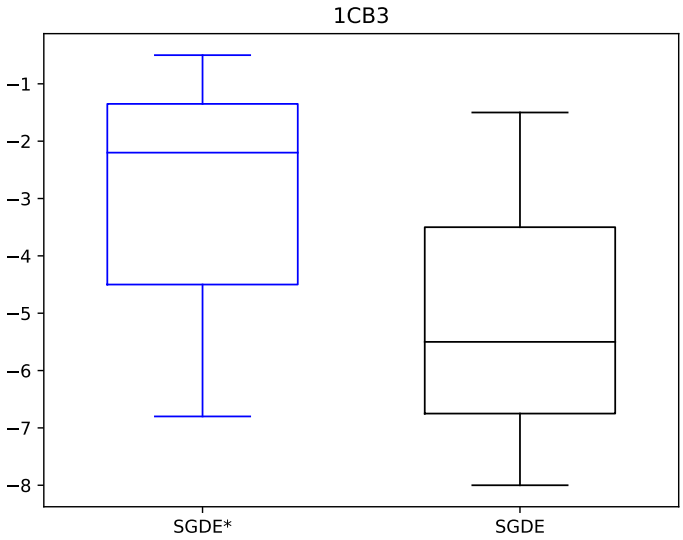


Figure 2.5: The comparison results of SGDE* and SGDE over 30 runs on protein sequences 1CB3.

with the best fitness; rank 2 to the second best, and rank N to the Nth best. Given this procedure, the average ranks based on the mean fitness values are calculated and summarized in Figure 2.8. According to this figure, the algorithms can be sorted in the following order: SGDE, SGDE†, DE$_{pfo}$, DE$^\star_{pfo}$, jDE, LSHADE, JADE, SaDE, CoDE and EPSDE. From this figure, we can say that both the SGDE and SGDE† are less sensitive to the dimensionality of the problem at hand and thus, have proved to be more scalable compared to the competitive algorithms.

Table 2.5: The best obtained solution vectors using the SGDE.

PDB ID	Solution
1CB3	−13.041, 20.817, −39.505, −14.249, 28.612, 3.133, −6.776, 6.401, 33.293, −30.798, −10.004, −30.594, 204.909, 165.391, 183.932, 188.005, 101.259, 14.773, 2.565, 221.216, 190.832
1BXL	−23.058, −101.926, −16.652, 46.170, 8.710, −0.418, 75.287, 24.366, 43.752, 1.303, −2.988, −2.432, −38.211, −0.777, 51.683, 98.770, 5.898, −76.227, −63.225, 162.538, −148.684, −33.216, 278.209, −167.932, −255.219, −154.179, −94.389
1EDP	−22.739, 1.910, −104.101, 21.195, −125.910, 26.263, −3.295, 23.916, 25.536, 21.164, 9.161, −44.040, 31.740, −46.338, −4.520, 15.290, 49.322, −157.821, −25.050, −189.757, −151.354, −49.747, 1.413, 20.335, −15.761, 55.586, −187.339, −53.988, −176.481

Table 2.5: The best obtained solution vectors using the SGDE.

PDB ID	Solution
2H3S	59.655, −26.849, −113.330, 3.700, −56.051, −49.854, 71.446, −27.260, 1.044, −123.514, −5.514, 90.039, −15.741, −26.559, 50.759, −23.017, 45.851, 48.831, 9.401, 50.956, −23.219, 3.534, 35.316, −0.199, −5.270, −124.885, −65.149, −30.528, 192.078, −198.127, 63.566, 127.694, −39.422, 200.616, 71.077, 133.549, 45.635, 196.027, −159.076, 222.841, −108.331, −202.128, 369.663, 269.937, 13.612
2KGU	−159.139, −71.147, −26.228, 5.416, −27.889, 37.277, 85.860, 29.850, 66.600, −220.387, 73.223, 224.554, −81.451, 55.263, −3.473, 43.010, −46.396, 45.797, −288.564, −20.181, −32.074, 14.577, 6.068, −151.986, −0.326, −79.239, −11.343, −258.643, 35.753, −144.949, −5.784, 12.601, −53.979, −56.730, 212.901, 131.871, 96.724, 160.449, 17.666, −231.835, 20.939, −41.439, 125.357, −131.312, −63.064, −55.975, −180.921, −182.716, −62.213, 182.275, −146.784, −256.896, 148.335, 35.609, 51.117, −342.816, 168.226, −52.008, 197.307, −124.738, −10.302, −164.867, −63.083
1TZ4	−165.334, 0.199, −308.832, −19.564, 57.088, −22.242, −33.621, 13.795, 54.533, −181.066, 8.051, −74.348, −43.488, 2.648, −42.501, −236.707, −11.640, −288.063, −13.457, −153.326, 2.644, −58.256, −155.085, 8.710, −24.551, 19.462, −59.251, 81.531, −3.096, −17.812, −255.897, 8.354, −69.865, −2.716, 48.152, 89.646, −2.270, −160.095, −39.007, 6.903, −28.411, −28.702, 50.580, 28.463, −226.413, −23.231, 17.755, −248.482, −170.342, 9.488, 197.418, 162.935, 28.759, −204.977, 5.143, −127.868, −190.456, 77.663, −177.556, 203.181, −242.256, −179.994, −126.389, −166.967, −40.466, 211.163, 7.530, −92.841, 5.068
1TZ5	144.424, 54.747, 102.739, 92.689, −2.814, 94.396, 2.146, 21.189, 69.594, 13.985, −66.838, 50.379, 48.012, 2.767, −126.631, 18.757, −11.099, 50.994, 31.931, −27.937, 3.396, 35.722, −21.505, −63.870, 47.882, −33.406, 16.053, 99.876, 279.718, −19.088, −294.202, −27.874, 62.764, −24.135, −23.670, 126.114, −127.681, −146.080, −52.368, 24.685, 4.347, 306.027, 23.096, 120.475, 203.812, 185.362, 283.888, 187.446, 66.357, −225.057, 126.390, −242.125, −146.168, −16.040, 302.675, −56.468, −65.507, −23.753, −223.905, −216.439, 243.711, 21.638, −228.963, 141.464, −176.279, −22.492, −232.266, 100.986, 202.063
1AGT	−156.030, −82.798, −95.070, −0.464, −112.604, −13.003, 36.340, −124.184, −39.982, 229.694, −3.162, −267.961, −33.234, 2.010, −93.592, 81.454, 331.001, 32.176, −9.160, 35.237, −1.374, −25.395, −128.790, −10.705, −56.745, −48.426, −12.472, 0.918, 66.112, −15.636, 12.279, 23.290, −76.885, −32.544, 184.096, −132.908, −47.248, −127.123, 33.011, 211.996, 59.836, 39.474, 26.187, 112.558, −21.629, 153.042, 223.658, 4.476, 33.030, −125.145, −74.598, −45.447, −149.345, 139.283, 128.786, 5.417, −24.633, −46.156, −207.543, 33.837, 146.291, 96.303, 49.327, 32.750, −172.228, −126.106, 137.126, −43.327, −147.187, 206.192, −35.189

Table 2.5: The best obtained solution vectors using the SGDE.

PDB ID	Solution
1CRN	145.300, 36.488, −27.693, 165.534, 77.902, 354.837, 7.136, 48.594, −80.512, −14.057, 99.159, 112.074, 46.753, 256.470, 90.789, 22.948, 85.355, 83.407, −43.953, −96.154, 11.458, 128.099, 43.359, −72.034, 237.431, −33.780, −47.403, −90.489, 19.382, 18.790, 337.112, 177.606, −106.985, −35.294, 171.970, 50.324, −52.184, 192.523, −188.396, 63.694, 29.464, −81.714, −31.965, 342.749, −181.666, −136.103, −133.733, 28.622, 112.944,155.587, 133.711, 34.217, −194.437, −6.297, −32.510, −59.951, −55.728, −180.195, −8.319, 23.287, 15.044, −165.647, −27.660, −94.603, 158.104, 21.053, 2.937, −46.729, −59.605, −119.248, 151.592, 76.387, 36.969, 64.091, 23.301, 195.288, 50.213, 19.561, 180.144, 55.116, 224.443,202.698, 163.036, 128.596, 71.251, 137.036, 239.625
1HVV	82.557, −4.648, 49.800, −23.404, −28.628, 41.943, 7.134, −166.340, −56.528, −337.274, −31.374, −93.214, 37.106, −117.152, 42.778, 100.133, −14.835, 76.012, −321.270, −72.339, 42.745, −37.322, 45.282, 37.796, −11.162, 54.482, −117.711, −5.714, −143.808, −1.347, 79.931, −10.487, −11.978, −17.393, 44.289, 29.164, 15.536, 23.316, −24.147, 163.034, −1.999, −6.900, 40.658, 42.585, 1.649, −77.651, 26.304, −60.850, −43.554, 9.506, 358.792, 27.782, −17.848, 13.687, 20.478, 6.313, −267.353, 2.239, 7.522, −78.233, −32.929, −7.866, 16.250, 30.633, 2.260, −5.060, −75.798, 11.519, 2.689, 11.681, 13.585, 1.897, 27.862, −113.754, −16.753, 43.384, −66.091, −161.383, −96.891, −187.561, 46.594, 114.072, 121.035, 138.510, 104.586, −37.781, 119.307, 40.934, −132.005, −52.130, −159.752, −124.843, 15.955, 260.259, −5.885, 81.430, 6.415, 161.376, 124.015,193.957, 52.734, 235.418, 21.524, 278.142, −110.396, −79.301, −184.381, −135.690, −213.179, −100.948, −26.899, −129.316, −81.018, −81.180, −187.666, 108.727, 132.176, −201.880, −61.588, −176.929, 93.506, 21.892, −53.959, −106.365, −72.412, −181.517, −165.334, 197.385, −128.767, −13.453, −323.226, −163.631, −59.022, 37.137, 43.509, −299.102, 90.037, 89.780, −14.997, 53.471, −203.178, 143.063, 40.360, −24.016, −16.890

Table 2.5: The best obtained solution vectors using the SGDE.

PDB ID	Solution
1GK4	−21.350, 7.289, −98.280, −44.417, −29.790, 32.793, −45.553, −6.752, 56.575, −8.413, 10.111, −8.803, 14.148, 8.485, 3.998, −45.350, −24.909, −12.958, 1.046, −10.167, 33.426, 9.071, 13.890, −40.705, 69.725, 17.744, −20.340, 18.051, 16.650, −90.021, 7.510, −13.634, 32.259, −0.115, 19.477,−25.760, 3.567, −7.441, 4.887, −5.443, 2.298, −7.960, 8.788, −9.155, 9.726, 34.049, −0.973, −17.697, −2.901, −0.428, 22.487, −52.917, 58.330, −19.208, −1.853, −6.544, −17.736, −11.070, 5.067, 39.871, 89.058, 16.728, 80.819, 5.917, −15.588, −26.382, −34.298, 1.734, −73.843, 12.014, −0.598,−44.611, −13.265, −1.520, 0.664, 4.390, −9.704, 10.029, 3.965, −0.721, 0.382, −3.113, 85.934, −19.991, −78.439, −7.090, 128.539, −9.184, −62.805,−12.132, 91.381, 35.076, −68.442, −24.881, 72.582, 121.197, 6.538, 21.101, −43.487, −148.246, −148.722, −40.254, 24.561, 47.359, 43.872,31.372, −78.728, −108.732, −124.579, −154.733, 12.213, −26.988, 76.124, 58.749, 43.400, −17.033, −145.120, −38.315, 22.566, −78.594, −16.745,85.985, −0.592, −85.335, 16.567, 76.919, −22.160, −24.485, 27.512, 63.799, 83.927, 63.474, −25.497, −187.504, −126.813, −21.404, −123.582, −121.014, −180.669, −167.495, −46.515, −159.078, 17.356, 16.533, −104.856, −41.776, 70.995, −11.817, 105.383, 162.064, 58.494, 79.789, −21.554,76.475, 35.576, 44.884, −15.714, −128.459, −23.287, 35.668, 85.378, −16.573, −11.516

Besides the above mentioned performance measures in Tables 2.3 and 2.4, a nonparametric test, called Friedman, followed by a Conover method, is also conducted to determine whether the obtained results by SGDE are significantly different from the other approaches. To do so, the mean values obtained from 12 algorithms over the 11 protein sequences are subjected to this test with 0.05 as the level of significance. The results indicate that with 95% certainty SGDE provided significantly better results for these problems.

The convergence curves of the improved algorithm over 30 runs per sequence are presented in Figure 2.9. The x-axis presents the number of function evaluations $\times 20,000$, and the y-axis is the logarithmic value of the mean energy value. Here, a bias value of 100 is added to make the energy values positive. From this figure we can see the proposed SGDE algorithm shows continuously fast convergence rate for the presented problems. The advantage of combining machine learning techniques and DE is revealed in this convergence plot. Take 1EDP as an example, all the algorithms are trapped in local optima while SGDE successfully escaped from the local optima and found the global optimum. The main point is that the introduced surrogate models provide a measure of uncertainty associated with the DE configurations. This uncertainty can effectively be used to construct more promising strategies during the optimization process.

Thereafter, we compared the results by the DE_{pfo}, jDE and L-SHADE on the high dimensional sequence 2EWH after 800,000 function evaluations, and the best offered solution by the SGDE after 400,000 evaluations to further validate its approximation strategy. According to our experiments, SGDE was capable of obtaining the best energy value compared

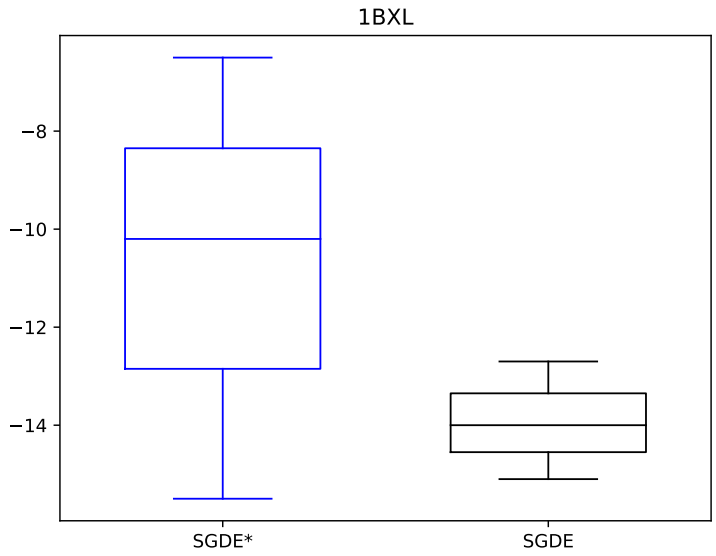


Figure 2.6: The comparison results of SGDE* and SGDE over 30 runs on protein sequences 1BXL.

to the other competitive algorithms. The folding results are reported in Table 2.6.

Moreover, we benchmark the performance of the SGDE for the real-world protein sequences. The results for 3 runs each with 1,000,000 function evaluations are collected in Table 2.7. The SGDE† uses only 500,000 evaluations. This simulation is computationally expensive. In this experiment, the total energy value (kJ/mol) of the obtained structure is used to assess the quality of the solutions from the optimization aspect, while the Root Mean Square Deviation (RMSD) is adopted to measure the similarity between the equivalent atoms in the obtained model and the native structure. We used the well-known BLAST server to provide the secondary structure of the target protein.

Table 2.6: Comparison of the results between the improved algorithms and SGDE for high-dimensional protein sequences 2EWH using simplified model (Std.: standard deviation).

Algorithm	Mean	Std.	Best
DE_{pfo}	-144.90	12.84	-171.63
jDE (DE/best/1)	-88.830	20.29	-129.88
L-SHADE	-104.96	4.930	-118.15
SGDE	-149.15	10.75	-201.05

This study investigates how approximation strategy of surrogate models can be used to replace in part the original computationally expensive PSP solver which may take from several minutes to several hours. For such problem, time overhead of training and building the surrogate models is insignificant compared to evaluating the exact fitness function which

Table 2.7: Comparison of the results between the improved algorithms and SGDE for high-dimensional protein sequences 1GK4 using all-atom model (Std.: standard deviation).

Algorithm	Run1	Run2	Run3	RMSD
DE_{pfo}	-769.36	-874.33	-657.32	0.05
jDE(DE/best/1)	-468.36	-569.33	-496.32	0.07
L-SHADE	-154.65	-256.32	-365.36	0.09
SaDE	150.14	130.15	94.31	0.18
JADE	21.63	12.36	-0.65	0.17
EPSDE	1001.12	987.65	698.78	0.23
SGDE	-6036.14	-5964.15	-7973.82	0.03
$SGDE^{\dagger}$	-2338.02	-1834.36	-1920.99	0.02

can happen when you are working on a new fold. In this case, a high proportion of the processing time involved in running is spent in function evaluation and surrogate models can be used to ease the computational burden. The surrogate models themselves are often expensive and are not recommended for the simple problems. Hence, this section takes a more detailed look on the importance of surrogate models for the PSP in regard to execution time. Generally speaking, the main purpose of the surrogate based methods is to find optimal solutions within very few expensive evaluations. So, we are interested in the obtained computation times for the $SGDE^{\dagger}$.

The most direct of this section is to show that the surrogate models can successfully reduce the computational costs for the PSP problems. Toward this goal, the execution (CPU) times for the ab-initio and all-atom models are presented in Tables 2.8 and 2.9, respectively. We did not report the results for the DE_{pfo} because it uses different setup, coding language, compiler and computational architecture. In Tables 2.8 and 2.9, the self time value show the total time in seconds spent inside an algorithm, excluding time spent for the fitness evaluations. In the case of ab-initio model, we record the time for two low-dimension problems (1CB3,1AGT) and a high-dimension problem (1GK4). The results of 1AGT using the all-atom model are also collected in Table 2.9. In this table, the values are rounded down. The simulations performed under Windows 7 operating system on an Intel(R) Core i7-6700HQ CPU and 8 GB of RAM. As can be seen from the results in Table 2.8, $SGDE^{\dagger}$ needs a huge computational time compared to the other algorithms. However, it is not a big surprise due the fact that surrogate based methods such as SGDE are not developed for computationally cheap problems. Having this in mind, one can see that $SGDE^{\dagger}$ significantly reduced the computational time for the 1GK4 problem. The effectiveness of this approach is further demonstrated in Table 2.9, where $SGDE^{\dagger}$ convergence about 26h faster for the 1GK4. The results also reveal the fact that even by considering SGDE, computational complexity of the algorithm become negligible compared to time overhead resulting from the expensive evaluations.

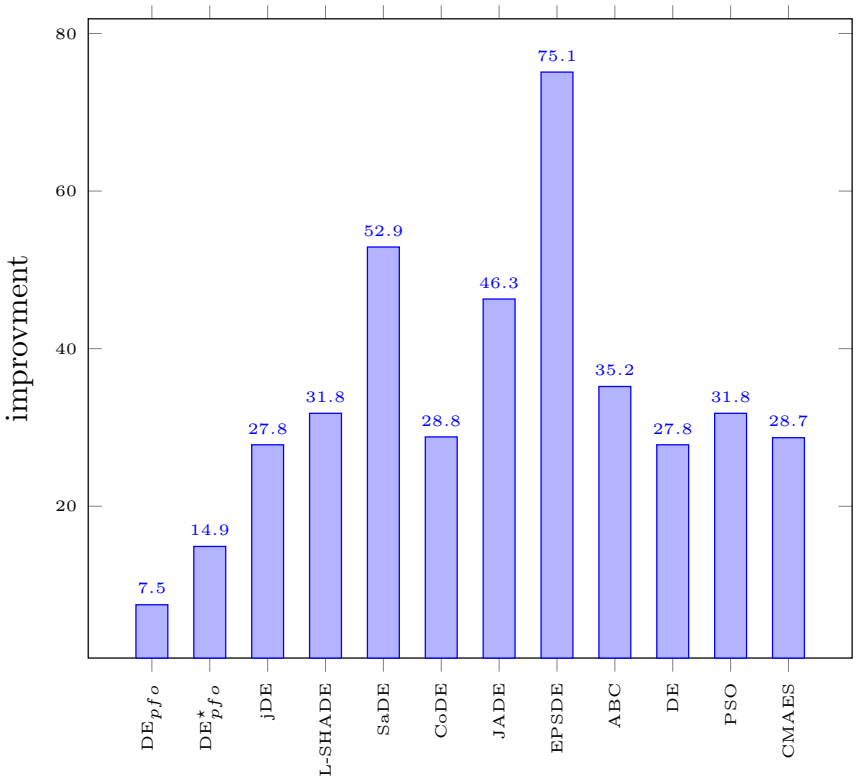


Figure 2.7: The percentage of improvement for SGDE over the competitive algorithms.

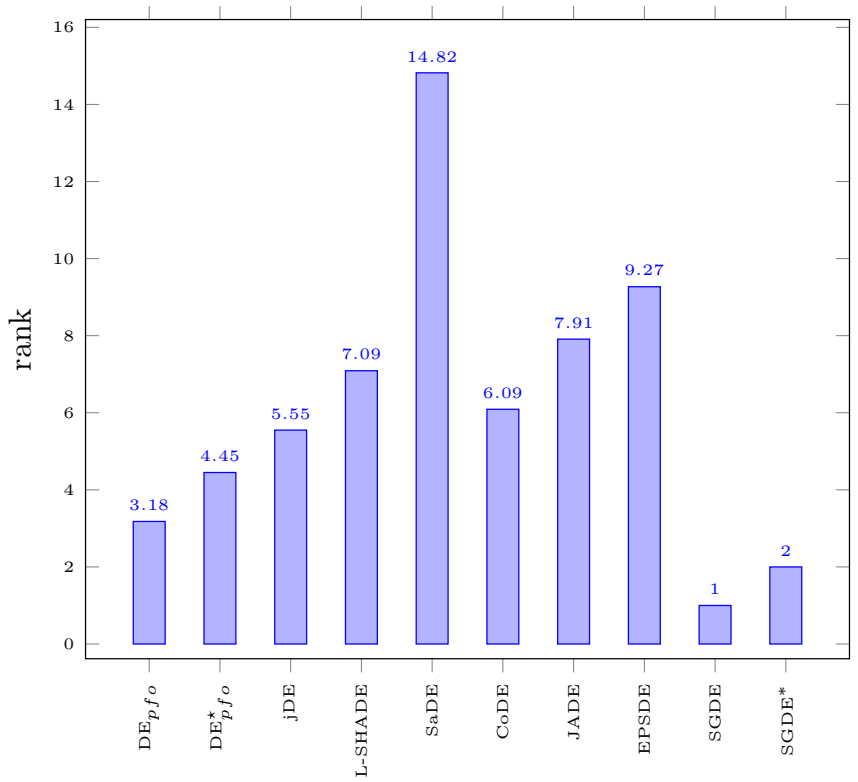
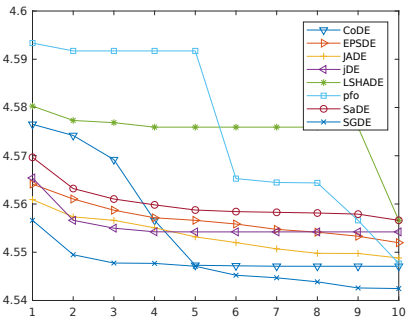


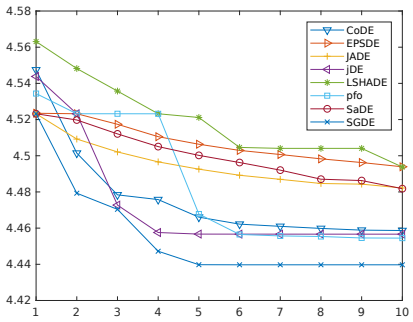
Figure 2.8: Ranking results of improved algorithms based on mean values.

Table 2.8: The reported computational times (in seconds) based on 3D AB off lattice model.

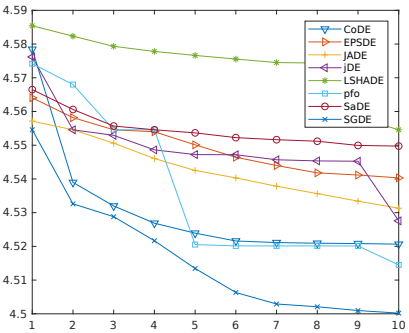
PDB ID	Algorithm	Evaluation time	Self time	Total time
1CB3	jDE(DE/best/1)	7.86	0.72	8.58
	L-SHADE		1.21	9.07
	SaDE		8.04	15.90
	CoDE		0.88	8.74
	JADE		0.71	8.57
	EPSDE		4.37	12.23
	SGDE		59.1	66.80
	SGDE†	3.93	24.1	28.05
1AGT	jDE(DE/best/1)	65.73	0.92	66,65
	L-SHADE		1.56	67,29
	SaDE		8.39	74,12
	CoDE		0.86	66,59
	JADE		0.91	66,64
	EPSDE		4.92	70,65
	SGDE		80.03	145,8
	SGDE†	32.87	43.46	76.33
1GK4	jDE(DE/best/1)	325.60	0.93	326,53
	L-SHADE		3.61	329,21
	SaDE		10.25	335,85
	CoDE		0.85	326,45
	JADE		2.16	327,76
	EPSDE		5.38	330,98
	SGDE		100.76	426,36
	SGDE†	162.8	74.16	236.96



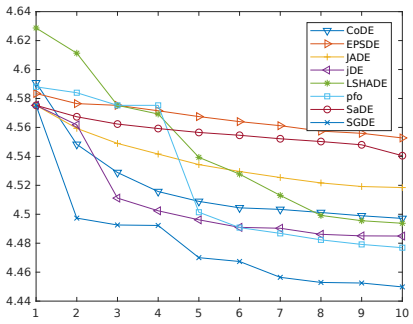
(a) 1CB3



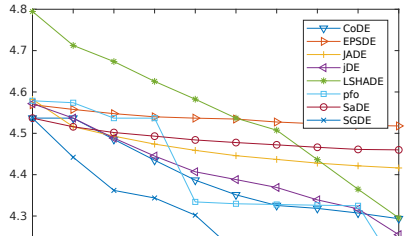
(b) 1BXL



(c) 1EDP



(d) 2H3S



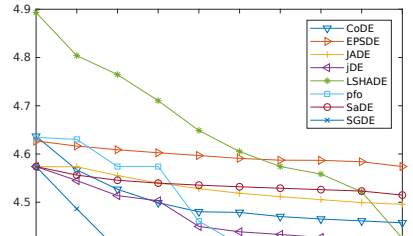


Table 2.9: The reported computational times (in minutes) based on the all-atom model for 3 runs.

Algorithm	Evaluation time	Self time	Total time
jDE(DE/best/1)		219	3361
L-SHADE		234	3376
SaDE		281	3423
CoDE	3142	227	3369
JADE		221	3363
EPSDE		249	3391
SGDE		321	3463
SGDE†	1571	291	1862

2.6.5 When Surrogate Models Do Not Help

A lot of things can go wrong. But some of them are more likely to be broken than others. First, notice that the main purpose of applying surrogate modeling is to replace an expensive-to-evaluate function by a simple response surface model. Consequently, it does not make any sense to apply surrogate models for a cheap-to-evaluate simulation function. This is due to the fact that the computational demands of building and training the surrogate models are also growing as their accuracy and complexity keeps increasing. In addition, you should remember that a sensitivity analysis is a pre-requisite for the surrogate modeling. Sensitivity, in this context, is a measure of the contribution of an independent variable to the total variance of the dependent data. This provides a systematic approach to identify how the variability in the computational model output is associated with the model input parameters. Starting with a model before applying sensitive analysis might provide more noise than useful information to the optimization process.

2.6.6 Implementation Notes

There is an overhead in model training phase which is implementation-specific and can be reduced with better-engineered code. We used a Matlab implementation for the conducted experiments which increases the required computational time compared to C, C++, or MEX-C. This thesis discourages coding the surrogate models in slow programming languages like Matlab. Relatively, it should be noted that modern advances in computing power increasingly rely on parallelization rather than faster processors. Hence, we recommend to take advantages of parallelization in their surrogate based implementations [132].

2.7 Practical End Use of Algorithm

We tried to incorporate the surrogate models during the PSP process. The main direction was to investigate how surrogate models can be used to evaluate more conformational search by means of cheap surrogate models. As a practical end use of algorithm, it would be interesting to introduce the surrogate models into the well-known Rosetta [125] predication

tools in order to provide more details on the use of surrogate models for the PSP. Furthermore, we would provide user-friendly web interfaces in order to generate reliable models using the proposed approach. They allow non-expert users to generate 3D models without the need to install and learn complex molecular modeling software and they have increasing impacts on both basic research and drug development. As pointed out and demonstrated in a series of recent publications [58], an established web-server gives a step-by-step guide on how to use the proposed algorithms to get the desired results without the need to follow the complicated mathematic equations. Particularly, it would be even more useful if the users can testify their new proposition through this web interface. The fully automated server for PSP problem has been continuously developed since 20 years ago [58]. Actually, many practically useful web-servers have increasing impacts on medical science, driving medicinal chemistry into an unprecedented revolution [36]. Hence, we would make efforts to provide a web-server for the new structure prediction method presented in this chapter.

2.8 Chapter Summary

In this chapter, we proposed a new extension of DE to accelerate the convergence rate of the standard algorithm for the computationally expensive PSP problem. The introduced SGDE verifies convergence conditions by adopting the surrogate modeling and GEP techniques. Moreover, it provides an improved search process which guides solutions by using CMAES. We evaluate the performance of SGDE as a specific algorithm for solving high dimensional real-world PSP problems using both ab-initio and an all-atom model. The SGDE is compared with ABC, DE, PSO, CMAES, DE_{pfo}, jDE (DE/best/1), L-SHADE, SaDE, CoDE, JADE and EPSDE. In this regard, six performance metrics are used: best fitness value, mean value of solutions, standard deviation, percentage of improvements, convergence rate and runtime complexity of the algorithms. Tables 2.3 and 2.4 presented the results of competitive algorithms based on the ab-initio model. These results clearly showed that SGDE significantly outperformed other algorithms in terms of the solution accuracy and robustness. The Friedman statistical test followed by a Conover method is in agreement with previous observations. The results also suggest that SGDE is less sensitive to the increases in dimensionality of the sequences and thus, has proved to be scalable. Moreover, it has been illustrated that SGDE provides a high enhancement in terms of the convergence speed. The same conclusions can be drawn for the all-atom model. Overall, experimental results suggest that the adopted surrogate models lead to a high convergence rate using a limited computational budget.

Chapter 3

Application II: Offline Algorithm Configuration

> Your assumptions are your windows on the world. Scrub them off every once in a while, or the light won't come in.
>
> Isaac Asimov

State-of-the-art metaheuristics algorithms often expose many parameters that should be configured to improve their empirical performance. Manual tuning of such parameters is synonymous with tedious experiments which tend to lead to unsatisfactory outcomes. Accordingly, researchers developed several frameworks to tune the parameters of a given algorithm over a class of problems. Until very recently, however, these approaches are not testified and applied to many-objective optimization problems. This study formulates a multi-objective algorithm configuration (MAC) method. In MAC, we take into account the importance of a given configuration by building a conditional probability graph. In this light, the introduced algorithm aims to explore more important variables using an undirected fully-connected graph. Experimental results reveal that MAC performs better in comparison with state-of-the-art F-Race and SMAC algorithms.

3.1 Motivation

There is no doubt that metaheuristic algorithms have gained immense popularity in recent years. The adoption of these algorithms to unseen $\mathcal{NP}$-hard problems, however, is severely hampered by choosing a set of optimal parameters associated with them. The learning rate in stochastic gradient descent or the mutation rate in the genetic algorithm are examples of these parameters. In particular, we can point out parameters of metaheuristics whose configurations have a high impact on their overall performance on a given class of instances. These configurations are correlated in non-intuitive ways which makes it difficult and tedious to tune them manually.

Automatic algorithm configuration deals with optimizing parameters of an algorithm so as to perform well across a broad range of instance types. In this regard, standard optimization algorithms like metaheuristics may need hundreds of evaluations to locate a near-optimal solution which is a major challenge to their successful application. This is primarily due to the expensive computational cost associated with them, which often

consume many minutes to even days of CPU time. In this context, the advantages of so-called model-based algorithms become clear [69]. They construct computationally cheap-to-evaluate surrogate models in order to provide a fast approximation of the expensive fitness evaluations during the search process. By leveraging surrogate models, the computational cost can be greatly reduced since the time overhead of training and building surrogate models is insignificant compared to evaluating the exact fitness function. To this fact, state-of-the-art algorithms such as SMAC [69] and F-RACE [17] have focused on model-based optimization.

To the best of our knowledge, automatic configuration methods have not been applied to many-objective problems and researchers were more interested on single-objective and multi-objective cases. In contrast to conventional multi-objective approaches, many-objective optimization poses a great challenge due to the ineffectiveness of Pareto dominance, inefficiency of recombination operation, rapid increase of computational time and parameter sensitivity.

3.2 Problem definition

A general definition of the algorithm configuration problem can be presented by a tuple $< I, \Theta, \Lambda, \zeta >$ as follows:

$$\theta^* = \arg \max_{\theta \in \Theta} u(\theta), \text{where } u(\theta) = f(\theta | I, P_I, P_\zeta, t) \tag{3.1}$$

where the parameters are:

— I: a set of problem instances which is given by a distribution P_I over admissible instances

— Θ: a set of all possible combinations of values of p_i

— Λ: an algorithm which should solve the problem class I, with input configurations $\theta = (p_1, \cdots, p_k) \in \Theta$. Here, $\Lambda(\theta)$ is the instance of algorithm Λ configured with θ

— $\zeta(\theta, i, t) = \zeta(\Lambda(\theta), i, t)$: assigns a cost value to each configuration θ when running $\Lambda(\theta)$ on instance $i \in I$ for time t. It could be modeled as $\zeta \sim P_\zeta(\zeta | \theta, i, t)$

The hyperparameter approach should then try to find configuration $\theta^* \in \Theta$ such that $\Lambda(\theta)$ yields the best utility u. To sum up, workflow of the automatic algorithm configuration problem is illustrated in Figure 3.1.

We introduce the hypervolume as the utility function u for formulating the many-objective problem. Assume that $\boldsymbol{f(x)} = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), ..., f_M(\boldsymbol{x}))$ denotes all the measured objectives on hyperparameter $\boldsymbol{x}$. In this context, we are interested to find a set of promising solutions so as to simultaneously minimize/maximize M objectives over the design space $\boldsymbol{\mathcal{D}}$ and objective space $\boldsymbol{f(\mathcal{D})}$ using the concept of Pareto optimality.

Definition 1 (Domination): *We denote the induced domination relation:* $\boldsymbol{x} \preceq \boldsymbol{x}'$ *iff* $\exists m \in \{1, ..., M\} : f_m(\boldsymbol{x}) < f_m(\boldsymbol{x}')$ *and* $\forall m \in \{1, ..., M\} : f_m(\boldsymbol{x}) \leq f_m(\boldsymbol{x}')$

Definition 2 (Pareto optimal): *A solution* $\boldsymbol{x} \in \mathcal{D}$ *is called Pareto optimal (or efficient), if it is non-dominated:* $\nexists \boldsymbol{x}' \in \mathcal{D}$*:* $\boldsymbol{x}' \preceq \boldsymbol{x}$

The many-objective algorithms return a set of solutions and a key question is how

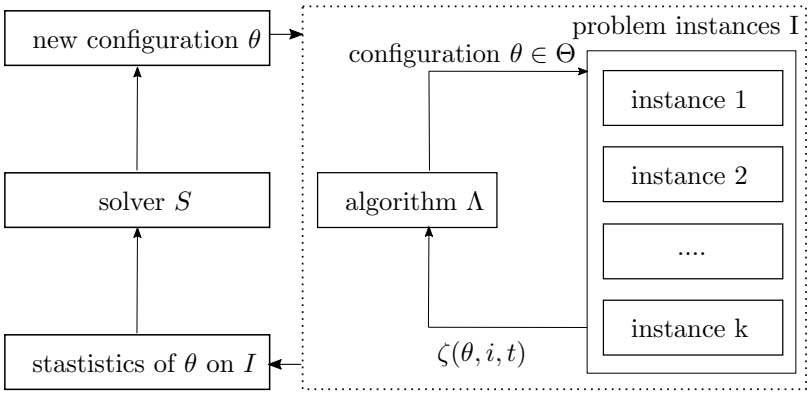


Figure 3.1: Workflow of automatic algorithm configuration

to compare the performance of the algorithms by their respective outputs. Indeed, this has been an active subject of work leading to a set of quality indicators based on Pareto front approximation. In this research, the well-known Hypervolume indicator is adopted to compare the performance of the algorithms. To summarize, some basic concepts are also depicted in Figure 3.2.

Definition 3 (Pareto front): *The Pareto front associated with a MO problem is a set of points in* $\mathbb{R}^M$ *that are all Pareto optimal.*

Definition 4 (Pareto front approximation): *An approximation of the Pareto front is a set* $A = \{\forall c, c^{'} \in A : c||c^{'}\} \in 2^{|\boldsymbol{f(\mathcal{D})}|}$, *where* $c||c^{'}$ *is satisfied when neither* $c \preceq c^{'}$ *nor* $c^{'} \preceq c$. *The set of all the aforementioned approximations are represented by* $\Phi_{\boldsymbol{f(\mathcal{D})}}$.

Definition 5 (Quality Indicator): *We define a quality indicator as* $I : \Phi_{\boldsymbol{f(\mathcal{D})}} \rightarrow \mathbb{R}$ *which assigns a fitness value to each approximating set.*

Definition 6 (Hypervolume indicator): *Assume that* $r \in \mathbb{R}^M$ *is a reference point,* $A \in \Phi_{\boldsymbol{f(\mathcal{D})}}$ and $B^+(A, r) = \{z \in \mathbb{R}^M | \exists c \in A : c \preceq z \preceq r\}$. The hypervolume is $I_H(A, r) = V(B^+(A, r))$, where V denotes the volume.

3.3 Related Works

Sequential Model-based Algorithm Configuration (SMAC) [69], Spearmint [152], F-RACE [17] and Tree-structure Parzen Estimator (TPE) [14] are examples of well known methods for automatic configuration task. A large class of such methods is characterized by modeling a conditional probability $p(y|\varphi)$ of a $m-$dimensional configuration φ, given n observations s with the corresponding evaluation metrics y. SMAC adopted a random forests model and Expected Improvement (EI) to compute $p(y|\varphi)$. It applies a multi-start local search and selects resulting configurations with locally maximal EI. The exploration property of SMAC is enhanced by the fact that EI conditioned on points with large uncertainty and low values of predictive mean. Similarly, TPE et al. [14] defined a configuration algorithm based on tree-structure Parzen estimator and EI. To tackle the *curse of dimen-*

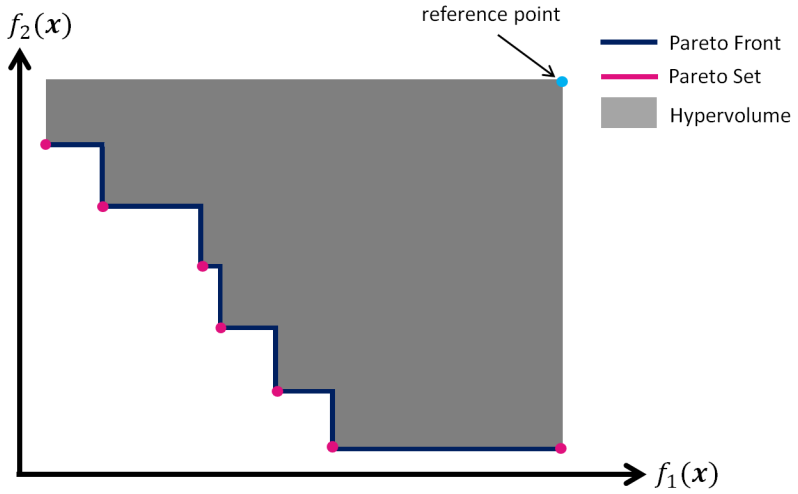


Figure 3.2: An illustration of Pareto front, Pareto set and hypervolume indicator for $M = 2$, with both objectives being minimized

sionality, TPE assigns particular values of other elements to the configurations which are known to be irrelevant. Ilievski et al. [72] proposed a deterministic method which employs dynamic coordinate search and radial basis functions (RBFs) to find most promising configurations. By using the RBFs [126] as surrogate model, they mitigated some of the requirements for inner acquisition function optimization. In another work [153], the authors put forward neural networks as an alternative to Gaussian process for modeling distributions over functions. They show that their introduced method is competitive with state-of-the-art GP-based approaches while it scales linearly with the data size rather than cubically. Blot et al. [19] introduced a multi-objective extension of the well-known ParamILS configuration framework and they demonstrate that it gives promising results on several challenging bi-objective scenarios. Interestingly, *Google* introduced *Google Vizier* [59], an internal service which incorporates Batched Gaussian Process Bandits along with the EI acquisition function.

3.4 Methodology

This section discusses in detail the main components of the proposed MAC method. In brief, MAC consists of two main phases:

— **Exploration:** The algorithm tries to learn probabilistically about the relevance of configurations and the model's performance during the optimization process. In other word, it expects to find reasons why a collection of past solutions is superior to others. To do so, MAC is equipped by a linkage learning component which periodically acquires information about the problem at hand to find most informative configurations. The aforementioned schema encodes the underlying dependencies between variables using an undirected graph, where nodes denote configurations and edges show the probability that two nodes are relevant. We adopted the idea of Eigenvector centrality feature selection [142] to learn the factor graph.

— **Exploitation:** The collected information from the previous step are then processed and used to generate new solutions. The introduced informed component guides the algorithm toward the search space that are likely to contain the promising solutions. This orthogonal technique prevents MAC to uniformly consider all configurations and bias the search process toward the good configurations.

We extend the idea of stochastic RBF [139] to be suitable for the algorithm configuration task. It is a model-based algorithm that cycles from emphasis on the objective to emphasis on the distance using a weighting strategy. Compared to the evolutionary algorithms like genetic algorithm, stochastic RBF need less computational time by virtue of surrogate modeling techniques. On the other hand, it mitigates some of the requirements for inner acquisition function optimization in comparison with well-know efficient global optimization (EGO) algorithm [78]. Hence, we focused on proposing a new algorithm configuration approach based on stochastic RBF. A generic framework for MAC includes some basic steps which can be stated as follows:

1. Generate a set of initial configurations θ^i $(i = 1, 2, ..., n)$ using design of experiments (DoE) and compute the cost value for each configuration
2. Build an initial surrogate model based on the sampled configurations θ^i in the first step
3. Find the current best configuration $conf_{best}$
4. Generate a set of random perturbations ρ based on exploration/exploitation modes
5. Generate a set of new configurations $confs_{new}$ around $conf_{best}$ using ρ
6. Use the surrogate model to select the best configuration $conf_{new}$
7. Evaluate $conf_{new}$ using exact cost function
8. Update the surrogate model based on $conf_{new}$
9. Check the stopping criteria: if some stopping criteria are satisfied go to Step 10; otherwise go to Step 3
10. Post-process the results

3.4.1 Initial Design

In MAC, the first step involves generating a set of random configurations θ^i $(i = 1, 2, ..., n)$ (i.e., initial population). Here, the algorithm might possibly miss a considerable portion of the promising area due to the high dimensionality of the configuration space (it should be noticed that we have a small and a fix computational budget and increasing size of the initial population cannot remedy the issue). Furthermore, it is crucial for a model-based algorithm to efficiently explore the search space so as to approximate the nonlinear behavior of the objective function. For these reasons, as with many model-based algorithms, MAC adopts DoE methods to partially mitigate high dimensionality of the search space. Among them, MAC uses the Latin Hypercube Sampling (LHS) [124] to provide a uniform cover in the search space using a minimum number of individuals. The main advantage of LHS is that it does not require an increased initial population size for more dimensions.

3.4.2 Build an Approximation Model

As the next step, we evaluate all the generated configurations θ^i $(i = 1, 2, ..., n)$ to build an approximate model of the cost function. This computationally cheap-to-evaluate model can provide a fast approximation of the expensive fitness evaluations during the search process. MAC tries to model conditional probability $p(y|\varphi)$ of a $d-$dimensional configuration φ given n observations $\mathbf{S}$ with the corresponding cost metrics $\mathbf{y}$:

$$\mathbf{S} = \left[\theta^{(1)}, ..., \theta^{(n)}\right]^{\mathrm{T}} \in \mathbb{R}^{n\times d}, \theta = \{\theta_1, ..., \theta_d\} \in \mathbb{R}^d \tag{3.2}$$

To do so, it offers surrogate models which are a set of mathematical tools for predicting the output of an expensive objective function. Particularly, they are designed to predict the fitness function value for any unseen configuration $\hat{\theta}$ according to computed data points (θ^i, y^i). Given a set of distinct configurations $\theta^1, ..., \theta^n \in \mathbb{R}^d$ with known values y^i, the RBF interpolant form is then computed as below [139]:

$$\tilde{f}(\hat{\boldsymbol{\theta}}) = \sum_{i=1}^{n} \lambda_i \phi(\left\|\hat{\theta} - \theta^i\right\|) + p(\hat{\theta}),\ \hat{\theta} \in \mathbb{R}^d \tag{3.3}$$

In Eq. 3.3, $\left\|.\right\|$ is the Euclidean norm, $\lambda_i \in \mathbb{R}$ for $i = 1, ..., n$, $p \in \prod_m^d$ denotes the linear space of polynomials in d variables of degree which is less than or equal to m, and ϕ is a RBF with one of the *surface splines* ($\phi(r) = r^k$ where $k \in \mathbb{N}$ is an odd number, or $\phi(r) = r^k log(r)$ where k is an even number), *multiquadratic* ($\phi(r) = (r^2 + \gamma^2)^k$ where $k > 0$ and $k \notin \mathbb{N}$), *inverse multiquadratic*($\phi(r) = (r^2 + \gamma^2)^k$ where $k < 0$ and $k \notin \mathbb{N}$) and Gaussians ($\phi(r) = e^{-\gamma r^2}$) forms. Here, $r \geq 0$ and $\gamma > 0$.

Following [139], MAC selected the *surface splines* form with $k = 3$ as the RBF. Having this in mind, we can compute a matrix $\Im \in \mathbb{R}^{n\times n}$ by $\Im_{i,j} = \phi(\left\|\theta^i - \theta^j\right\|)$; $i, j = 1...n$. Assume that $\hat{m}$ is the dimension of the linear space $\prod_m^d$ such that $m \geq = \lfloor k/2 \rfloor$. Accordingly, we have another matrix $\mathbf{P} \in \mathbb{R}^{n\times\hat{m}}$ such that: $P_{ij} = p^{(i)}(\theta^{(i)})$, $i = 1..n; j = 1..\hat{m}$. The approximated model can then be obtained by solving the system as presented in Eq. 3.4, where $\mathbf{c} = (c_1, ..., c_{\hat{m}})^T \in \mathbb{R}^{\hat{m}}$.

$$\begin{pmatrix} \Im & \mathbf{P} \\ \mathbf{P}^\top & 0 \end{pmatrix} \begin{pmatrix} \gamma \\ \mathbf{c} \end{pmatrix} = \begin{pmatrix} \mathbf{y} \\ 0_{\hat{m}} \end{pmatrix} \tag{3.4}$$

3.4.3 Exploration

The original stochastic RBF method generates a set of candidate points by adding random perturbations ρ to the best obtained solution (i.e., configuration) to guide the search process. This trial-and-error procedure does not take into account the interactions between the generated configurations and the obtained objective values. We note that the performance of stochastic RBF depends on this *random* points and a more informed scheme can be beneficial to enhance the robustness of the algorithm. Indeed, this is the same desired property in *optimal contraction theorem* [31] which states an optimal optimizer should dynamically considers useful information about the problem at hand. Motivated

by this finding, MAC incorporates an adaptive control strategy which keeps a historical memory of the ρ perturbations to guide the generation of future configurations. In the exploration phase, MAC generates a diverse set of random perturbations ρ and tends to increase global search to prevent algorithm from being trapped in a local minimum. We adopted Student's t-distribution to do so, which is a symmetric and bell-shaped family of distributions like the normal distribution. In contrast, however, it has heavier tails which let MAC explores the points that fall far from the distribution's mean. At each iteration t, MAC archives the generated perturbations for the exploitation phase.

3.4.4 Exploitation

After half of the iterations, MAC employs a feature selection algorithm method to acquire information about the performance of each of those randomly generated perturbations in the previous phase. It uses this information to dynamically make a balance between exploration and exploitation. In other words, MAC transforms the task of learning the optimal feature in feature selection algorithms into the search for an efficient and adaptive optimization behavior. This enables MAC to take into account the underlying correlations between the generated perturbations and domain-specific search knowledge of the problem.

Following [141], MAC creates an undirected graph $G =< V, E >$ according to which nodes represent the generated random perturbation $\rho^{(t)}$ and edges denote relationships among pairs of nodes. All the archived perturbation $\rho^{(t)}$ are ranked in descending order according to their associated cost values: the first best 50% solutions are labeled as *promising* and the other solutions are labeled as *non-promising*. This consideration addresses the imbalanced training set and prevent of biasing against the minority class.

Given the above-mentioned training set, an adjacency matrix A is associated with G in order to define relationships between the nodes. The graph G is represented through an adjacent matrix A, where each element $a_{i,j}$ shows pairwise relations among feature distributions. The $a_{i,j}$ elements are defined as follows:

$$a_{i,j} = \alpha\sigma_{i,j} + (1-\alpha)c_{i,j}; 1 \leqslant i, j \leqslant t \tag{3.5}$$

In Eq. 3.5, α is a scaling factor $\in [0, 1]$, $\sigma_{i,j} = max(\sigma_i, \sigma_j)$ where σ_i denotes the standard deviation over the ρ and $c_{i,j}$ is a kernel. To compute the $c_{i,j}$, first the Fisher criterion should be applied [141]:

$$f_i = \frac{|\mu_{i,1} - \mu_{i,2}|^2}{\sigma_{i,1}^2 + \sigma_{i,2}^2} \tag{3.6}$$

In Eq. 3.6, discriminate classes *promising* and *non-promising* are labeled as 1 and 2, respectively. Also, $\sigma_{i,c}^2$ and $\mu_{i,c}$ are the mean and standard deviation of the i-th feature for class c. The k can be obtained as $k = (f.m^\top)$ where the mutual information m is [141]:

$$m_i = \sum_{y \in Y} \sum_{z \in \rho^{(i)}} p(z, y) log(\frac{p(z, y)}{p(z)p(y)}) \tag{3.7}$$

In Eq. 3.7, Y shows class labels and p denotes the joint probability distribution. Now,

MAC computes the eigenvalues υ and eigenvectors v of A. The obtained weigh for generating the new configurations is equal to the eigenvector associated to $\eta_0 = \max_{\eta \in \upsilon}(abs(\eta)))$.

3.5 Experiments

3.5.1 Benchmark Sets

We considered CEC'2018 benchmark [35] suite to compare the competitive configuration algorithms. Accordingly, we investigate the performance of the introduced MAC method using two different many-objective algorithm configuration scenarios. The first scenario is designed to optimize the configuration space of NSGA-II [47] defined by three control parameters, while the second one adopts MOPSO [37] which is defined by six control parameters. The adopted algorithms are introduced to measure search performance of the MAC under different dimensions. The NSGA-II is a low dimensional configuration problem, while MOPSO is a medium dimension problem. The considered configurations are presented in Tables 3.1 and 3.2.

Table 3.1: The considered configurations of NSGA-II for a D-dimensional problem

Name	Type	Range	Default values [50]
population size	integer	[100,500]	100
crossover rate	continuous	[0.1,1]	1
mutation rate	continuous	[0.1,1]	$1/D$

Table 3.2: The considered configurations of MOPSO for a D-dimensional problem

Name	Type	Range	Default values [150]
population size	integer	[100,500]	100
inertia weight	continuous	[0.1,1]	0.9
C1	continuous	[0.1,2]	1.8
C2	continuous	[0.1,2]	1.8
V	continuous	[0.1,1]	0.6
mutation rate	continuous	[0.1,1]	$1/D$

3.5.2 Baselines

Experiments are conducted based on random search, SMAC and F-RACE methods. Empirical evidence reveals that random search can outperform Grid search within a small fraction of the computation time. Furthermore, SMAC and F-RACE are two state-of-the-art automatic configuration frameworks and to our best knowledge this is the first study which reports their performance for many-objective problems.

3.5.3 Experimental Settings

It should be mentioned that the number of objectives is set to $M = 5$. Our experimental procedure follows two steps, namely *training* and *test*. In the *training* step, each evaluation involves running the NSGA-II/MOPSO on the training problem instances MaF1, MaF2, MaF5 and MaF6-10 (the size of training and test instances is small due to small number of CEC'2018 benchmark problems) for 10 runs. After finishing the automatic configuration step, the parameters obtained from the *training* step are applied to rest of the problems in order to validate the performance of the optimized configurations on the unseen test instances. In the case of NSGA-II and MOPSO, number of fitness evaluations is set to be $max(1.0e{+}5, d \times 1.0e{+}4)$, where m is the default dimensionality of the problem.

For each algorithm, stopping criteria are satisfied if the algorithm exceeds 5×200 evaluations, or if the computational time reaches 5×24 hours. We used hypervolume (HV) in order to compare the proximity and diversity of the obtained results. The HV indicator should be maximized during the configuration process.

3.5.4 Results and Discussion

The obtained results are summarized in Tables 3.3- 3.4 and Figure 3.3. In these tables, NSGA-II and MOPSO denote the obtained results by using the default configurations. The best results are indicated in boldface.

Table 3.3: Average Hypervolume values for final test fronts of NSGA-II algorithm

Problem	Random Search	SMAC	F-RACE	NSGA-II	MAC
MaF1	0.0132	**0.0141**	0.0109	0.0072	0.0135
MaF2	0.0440	0.0448	0.0407	0.0342	**0.0449**
MaF5	32400.0000	**32759.7820**	29971.3091	20900.0000	27200.0000
MaF6	**0.0093**	**0.0093**	**0.0093**	0.0092	**0.0093**
MaF7	1.8471	1.8914	1.7851	1.7035	**1.9127**
MaF8	**4.7763**	4.7708	1.5404	3.9464	4.7749
MaF9	7.5106	7.9908	7.4298	3.7968	**8.0242**
MaF10	2500.0000	**2810.4974**	2470.0709	2464.2300	2470.0000
MaF11	6080.0000	**6102.3614**	6070.2643	5970.0000	6100.0000
MaF12	3710.0000	3869.2988	3657.2979	2825.0000	**3880.0000**
MaF13	0.4376	0.4446	0.4470	0.2080	**0.4547**
MaF14	0.1464	0.1464	0.0732	0.1464	**0.1513**

Arguably, NSGA-II is one of the well-known methods which mimics the same developmental process in the standard GA: Selection, reproduction and evaluation. It is worth tuning the population size, mutation and crossover probabilities of the NSGA-II to find reasonable settings for the problem at hand. A small population size will lead to the early convergence problem, while using a large population increases the computational cost. The configurations tuning of NSGA-II becomes even more challenging due to the fact that there is a correlation between its control parameters. For example, mutation is more effective on smaller population sizes while crossover is likely to benefit from large populations. All the mentioned reasons make the NSGA-II a challenging algorithm for benchmarking the

Table 3.4: Average Hypervolume values for final test fronts of MOPSO algorithm

Problem	Random Search	SMAC	F-RACE	MOPSO	MAC
MaF1	0.0106	**0.0116**	0.0154	0.0030	0.0107
MaF2	**0.0417**	0.0396	0.0389	0.0315	0.0407
MaF5	29100.0000	**33163.6115**	32753.9500	16600.0000	32900.0000
MaF6	**0.0093**	**0.0093**	**0.0093**	0.0089	**0.0093**
MaF7	1.5039	**1.7304**	0.6107	1.0984	1.6691
MaF8	4.3301	4.5889	4.6987	4.3521	**4.7207**
MaF9	7.9267	8.7605	8.7001	2.5895	**9.1728**
MaF10	1440.0000	1513.8214	1485.7750	1465.0000	**1570.0000**
MaF11	5540.0000	5471.3664	5570.2050	4915.0000	**5600.0000**
MaF12	3190.0000	3143.7161	3181.9100	1440.0000	**3450.0000**
MaF13	0.0475	0.3558	0.3705	0.0219	**0.3710**
MaF14	**0.1464**	**0.1464**	**0.1464**	0.1369	**0.1464**

performance of the MAC. However, Table 3.3 shows the automatic algorithm configuration methods enhanced the performance of the NSGA-II over the problems. Meanwhile, it is worth mentioning that MAC exhibits more promising performance than other competitive methods. The same situation could happen for the considered MOPSO algorithm. As it can also be seen from Table 3.4, MAC finds considerably better configurations in terms of HV indicator.

Figure 3.4 provides additional details by showing the behavior of the considered methods over 4 different problems. This figure shows how the introduced methods adopt their tuning behavior for NSGA-II and MOPSO algorithms. Altogether, with respect to the performed experiments, we can say that MAC and other frameworks have achieved promising results. From the illustrated correlation matrix (results are recorded by MAC) in Figure 3.3, it can also be concluded that well-tuning the algorithms even with very small training instances can enhance the search performance of the many-objective approaches.

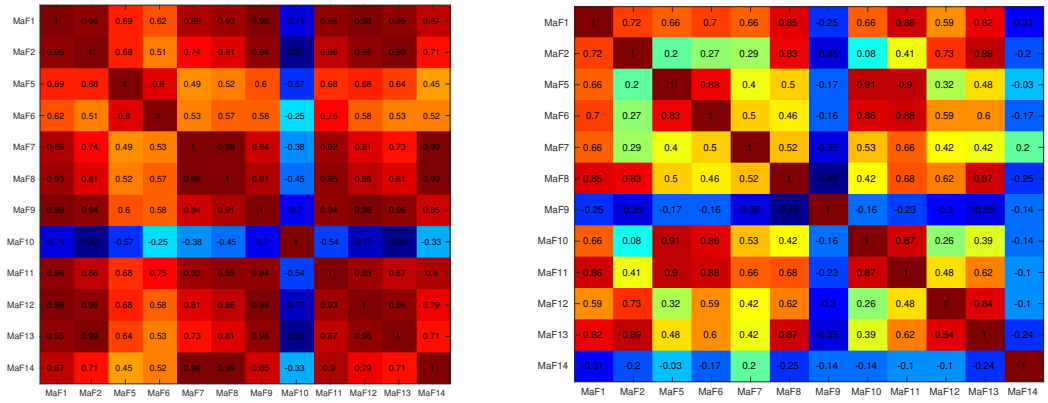
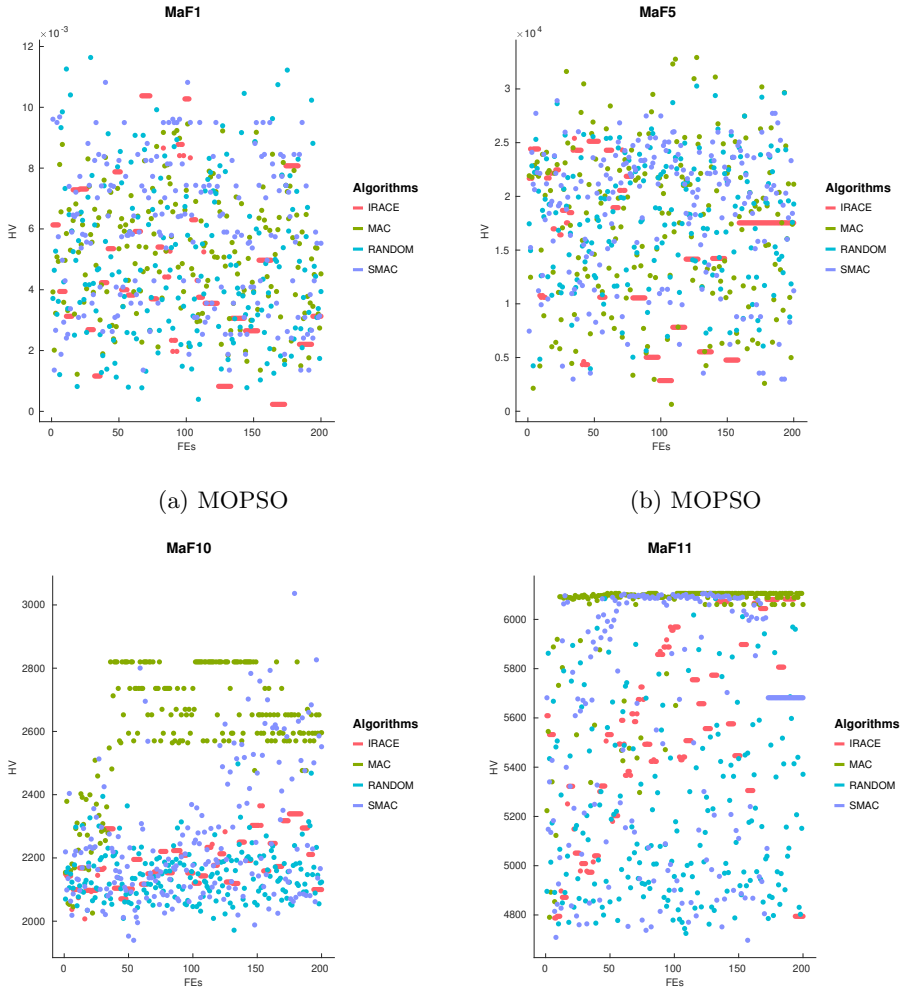


Figure 3.3: The obtained correlations between the configurations of different problems for the NSGA-II (left) and MOPSO (right).



(a) MOPSO (b) MOPSO

(c) NSGA-II (d) NSGA-II

Figure 3.4: HV values v.s. number of function evaluations (FEs) of different methods for optimizing 6 parameters of MOPSO and 3 parameters of NSGA-II. One dot represents HV value of an algorithm at the corresponding evaluation number

3.5.5 When Feature Selection Does Not Help

We did not notice a significant overhead in our experiments by applying feature selection. However, sometimes features selection might not be beneficial due to the low configurations/features correlations. In such a case, the feature selection strategies are likely to introduce noise into your optimization process. We highly recommend to examining feature selection strategies on your problem instead of blindly go with an approach; which might save you plenty of time.

3.5.6 Implementation Notes

This study formulates a many-objective algorithm configuration method (MAC) which is available for the Matlab and Python. MAC can be easily installed. It can be used

to directly configure the algorithms in PlatEMO, Platypus and PyGMO packages. Also, like SMAC and F-RACE it can be linked with other frameworks such as jMetal by means of a Wrapper function. MAC supports automatic generation of LaTeX tables, applying statistical pairwise comparison, and graphical visualization. The MAC can be executed in parallel on multiple cores.

3.6 Practical End Use of Algorithm

We believe that MAC could be very useful in real-world applications due to the fact that it is not easy for manually tuning the parameters of many-objective algorithms by considering multiple performance criteria. We are intended in using many-objective versions of model-based algorithms for parameter tuning tasks. It would be interesting to investigate how many-objective methods can cover a range of trade-offs in comparison with single-objective ones.

3.7 Chapter Summary

In this study, we present a framework for automatic algorithm configuration of many-objective optimization methods. The introduced MAC incorporated the idea of feature selection into the stochastic RBF method using an undirected graph. The MAC is proposed in the interest of integrating the optimization methods and machine learning techniques. The application of MAC to very recent CEC'2018 benchmarks compared to 3 state-of-the-art competitors show efficient performance.

Chapter 4

Application III: Numerical Optimization

> We are an impossibility in an impossible universe.
>
> Ray Bradbury

Metaheuristic algorithms have seen unprecedented growth thanks to their successful applications in fields including engineering and health sciences. In this work, we investigate the use of a deep learning (DL) model as an alternative tool to do so. The proposed method, called MaNet, is motivated by the fact that most of the DL models often need to solve massive nasty optimization problems consisting of millions of parameters. Feature selection is the main adopted concepts in MaNet that helps the algorithm to skip irrelevant or partially relevant evolutionary information and uses those which contribute most to the overall performance. The introduced model is applied on several unimodal and multimodal continuous problems. The experiments indicate that MaNet is able to yield competitive results compared to one of the best hand-designed algorithms for the aforementioned problems, in terms of the solution accuracy and scalability.

4.1 Motivation

The need for optimization has received a lot of attention in different application areas. Metaheuristics are one of the fastest growing fields aimed at solving different complex and highly non-linear real-world problems by inspiration from the process of natural evolution or physical processes [82, 135]. In metaheuristics, we often have a population of candidate solutions that strive for survival and reproduction. In every iteration, different search operators are applied to the candidate solutions and then the population will be updated based on its success in achieving the goal. Over the last decade, there has been an explosion in the development of a variety of extensions to further enhance the performance of metaheuristics. However, there are no clear guidelines on the strengths and weaknesses of alternative methods such as the deep learning models for developing more enhanced optimization algorithms.

The deep learning approaches use a hierarchy of features in conjunction with several layers to learn complex non-linear mappings between the input and output layer. As opposite to traditional machine learning methods that use handmade features, the important features are discovered automatically and are represented hierarchically. This is known

to be the strong point of deep learning against traditional machine learning approaches. Accordingly, these models have been described as universal learning approaches that are not task specific and can be used to tackle different problems arise in different research domains [4]. In this chapter, we propose a simple, yet effective approach for numerical optimization based on the deep learning. The proposed MaNet adopts a Convolutional Neural Network (CNN); which is a regularized version of fully-connected neural networks inspired from biological visual systems [94]. The "fully-connectedness" of CNNs enables them to tackle the over-fitting problem and it is reasonable to postulate that they may outperform classical neural networks for difficult optimization tasks.

Some of the recent works mainly aim at providing optimal solutions within a very limited computational time [153], while others [6, 104] primarily focus on getting better heuristic solutions. These success stories of DL motivated us to investigate the ability of a moderate model so as to make a balance between the solution accuracy and computational time. Altogether, these are the same desired properties in MAs and our work is a step towards investigating the usefulness and strong potential of this research direction.

4.2 Problem Definition

Formally, optimization algorithms seek to find a parameter vector x^* so as to minimize a cost function $f(x) : \mathbb{R}^D \to \mathbb{R}$, i.e. $f(x^*) \leq f(x)$ for all $x \in \Omega$, where $\Omega = \mathbb{R}^D$ is the search domain and D is the dimension of the problem. There are no a prior hypothesis about f and optimization algorithms should treat them as black-box functions. This motivated the development of MAs which do not take advantages of problem structure.

4.3 Related Works

The idea of solving optimization problems using neural networks has an old history which has seen a number of advances in recent years [5, 6, 87, 104, 153]. In [153], authors developed a Bayesian optimization method, called as DNGO, based on deep neural networks for hyperparameter tuning of large scale problems with expensive evaluation. The main idea is to combine large-scale parallelism with an optimization method to provide an approximate model of the real cost function. They show that DNGO scales in a less dramatic fashion compared to the Gaussian process, while maintains its desirable flexibility and characterization of uncertainty. OptNet [5] is another method proposed for learning optimization tasks by the virtues of DL, sensitivity analysis, bilevel optimization, and implicit differentiation. The authors highlighted the potential power of OptNet networks against existing networks to play mini-Sudoku. In [6], researchers investigated automating the design of an optimization algorithm by long short-term memory deep networks on a number of tasks. Their results outperform hand-designed competitors for simple convex problems, neural network training and styling images with neural art. Similarly, Li and Malik [104] put forward a deep learning method for automating algorithm design process. They formulate the problem as a reinforcement learning task according to which any can-

didate algorithm is represented by a policy and the goal is to find an optimal policy. To verify this finding, the authors conducted a set of experiments using different convex and non-convex loss functions correspond to several machine learning models. The obtained results clearly suggest that the automatically designed optimizer converges faster compared to hand-engineered optimizer.

4.4 Methodology

This section presents a new optimization method, called MaNet, to explore the possibility of adopting a lightweight deep learning architecture for continuous optimization tasks. In the following, it is assumed that the reader is familiar with the basic concepts of evolutionary computation and deep neural networks. In case the reader needs a refresher on any of those topics, author recommends Part II as well as [94].

The MaNet is designed to have the common properties of the MAs: providing a sufficient good solution with incomplete or imperfect information. It starts the optimization procedure with a set of randomly generated solutions as genotype. During training the network, MaNet applies the network training components directly on the genotype, while decodes a genotype into a phenotype (i.e., individuals in MAs) only in the last layer. It finds an optimized solution by iteratively improving an initial solution with regard to its cost function. Among different DL models, CNNs trained with an extension of stochastic gradient descent is used to build the MaNet. The CNNs have been central to the largest advances in computer vision [94] and speech processing [66]. A CNN is a DL method that uses convolutional layers to filter redundant or even irrelevant input data to increase the performance of the network [64]. This consideration also reduces the dimensionality of the input data and speeds up the learning process in the CNNs. Besides, it allows CNNs to be deeper networks with fewer parameters. Altogether, these properties could make CNNs a potential tool for solving optimization problems; especially when we take into account the history behind the application of feature selection [133] and problem scale reducing [149] in the optimization domain.

The architecture of a CNN consists of an input and an output layer, as well as one or more hidden layers. The hidden layers are typically composed of convolutional layers, fully connected layers, normalization layers and pooling layers. The number of hidden layers could be increased depending on the complexities in the input data, but at the cost of more computational expensive simulations. From the mathematical perspective, convolution layers provide a way of mixing input data with a filter so as to form a transformed feature map. Fully-Connected layers learn non-linear combinations of the high-level features by connecting neurons in one layer to neurons in the previous layer, as seen in multi-layer perceptrons neural networks (MLPs). Moreover, normalization layers are adopted to normalize the data to a network and to speed up learning. This includes batch normalization [146], weight normalization [145], and layer normalization [98] techniques. Batch normalization is applied to the input data or to the activation of a prior layer, weight normalization is applied to the weights of the layer and layer normalization is applied across the features. The pooling layers are usually inserted in-between successive convolutional layers to further

reduce the number of parameters in the network. A CNN network can have local or global pooling layers that may compute a max or an average.

Inspired by the aforementioned components in CNNs, the MaNet is designed to train a model so as to solve an optimization problem (Fig. 4.1). The existing feature selection and dimensionality reduction policies in CNNs help MaNet to find complex dependencies between the parameters. The MaNet starts optimization by generating a set of random $n \times m$ inputs for the model (i.e., the raw pixel values of the image). So, each individual solution is represented by a matrix rather than a vector. During training the network, convolutional layers transform the initial population layer by layer to a final feasible solution. This large part genotype representation enables the optimizer to keep genetic information that was necessary in the past as a source of exploration, as well as a playground for extracting new features that can be advantageous in the exploitation.

The MaNet multiplies the initial population with a two-dimensional array of filters that are connected to every disjoint region. The output of multiplying the filters with initial population forms a two-dimensional output array called as "feature map". They are obtained by convolution process upon the initial population with a linear filter, without applying a non-linear function or applying feature normalization methods. Similar to other DL models, the filters/kernels in MaNet are learned using the back-propagation algorithm for each specific optimization task. This is the novel aspect of DL techniques that filter weights are learned during the training of the network and are not hand designed. Accordingly, CNNs are not limited to image data and could be used to extract a variety types of features. Thank to this characteristic, MaNet will be forced to extract the features that are the most important to minimize the loss function for the problem at hand the network is being trained to solve. In each convolution layer, we have some predefined hyperparameters that can be used to modify the behavior of the model: the filter size and the number of filters. The first one simply denotes the dimensions of the filter when applying the convolution process, while the second one determines the number of different convolution filters.

In MaNet, multiple convolution layers are stacked which allows convolution layers to be applied to the output of the previous layer, results in a hierarchically set of more decomposed features. Finally, a Dense layer (or fully-connected) with linear activation function will be used to form the final solution vector. As it can be seen from Fig. 4.1, MaNet has a very simple structure and can benefit from the advantage of having a fast network training process [1]. Indeed, it has only 3,742 trainable parameters compared to state-of-the-art models [151] which have millions or billions of parameters. This could facilitate the application of MaNet for optimization tasks where a small amount of data (i.e., population) is available.

1. Netron Visualizer is used to illustrate the model. The tool is available online at: https://github.com/lutzroeder/netron

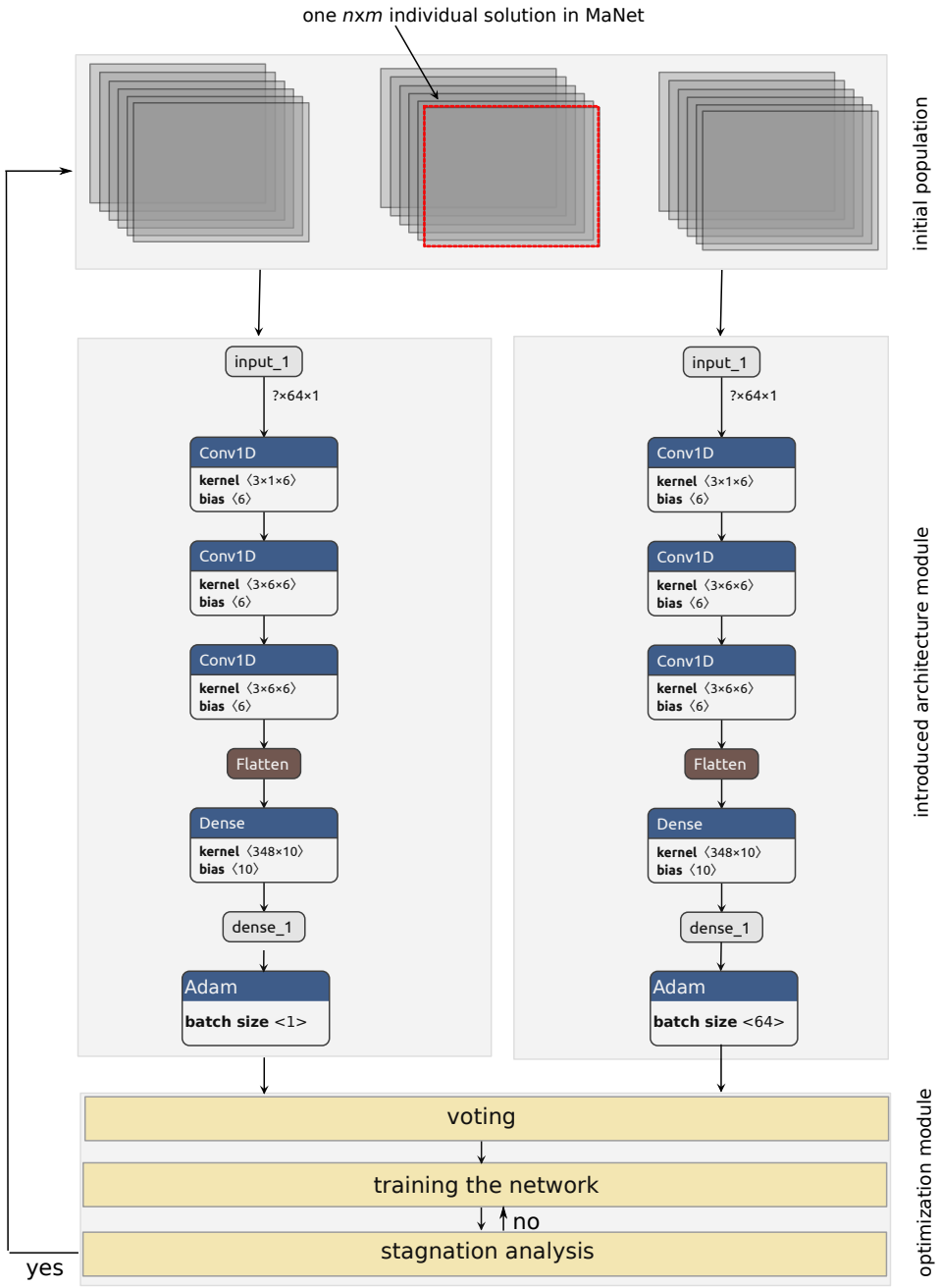


Figure 4.1: An overview of the proposed optimization architecture. The MaNet is composed of three convolution layers and one Dense layer (or fully connected layer). In each layer, the number of filters and the filter size are 6 and 3, respectively. The activation function for all the layers is proportional to their inputs.

As it can be seen, the MaNet is composed of two similar architectures which are subjected to different optimization procedures. The first one uses a batch size of one and the other uses 64 as its batch size. The batch size is a hyperparameter of gradient descent that should be tuned for each optimization task. To do so, MaNet integrates a reinforcement

strategy inspired from SDCS [135]. Technically speaking, SDCS is a simple metaheuristic algorithm which toggles continually between two snap and drift modes to enhance reinforcement and stability. Based on this idea, MaNet introduces a self-adaptive strategy to tune the batch size hyperparameter. More precisely, it is looking to see if the best cost function stops improving after some number of epochs, and if so then it restarts the optimization process and continuous the search by the architecture which obtained a higher overall performance so far. Finally, it is worth mentioning that the initial population will remain unchanged during training the network and the algorithm will evolve a set of filters. The goal of MaNet then, is to transfer the initial population on one end to evolved solutions on the other hand. This is one of the main differences between MaNet and evolutionary algorithms.

4.5 Experiments

4.5.1 Benchmark Sets

We use a set of 9 benchmark functions given in CEC 2017 [9] to evaluate the performance of the proposed algorithm [2]. The considered problems are widely used in the optimization community and are challenging for any optimization approach. This work uses several problems that can be classified into unimodal (F1 and F3) and multimodal (F4-10) minimization functions with different properties including separable, non-separable, rotated, ill-condition and shifted [3]. The aforementioned problems are adopted on the GPU so as to be linked with machine learning libraries. We refer the reader to the detailed principle about the definition of CEC2017 benchmark functions as defined in [9]. To verify the algorithm scalability, 30-dimensional and 50-dimensional problems are used. All functions should be minimized and have a global minimum at $f(x) = 0$. The results are reported according to their distance from the optimum. We trained MaNet on each problem by using the parallel power of 9 NVIDIA Tesla K20m GPU cards.

4.5.2 Baselines

It has been shown that various extensions of the differential evolution (DE) [158] algorithm are always among the winners of the CEC competition. Having this is mind, we used jSO [26] algorithm for the purpose of comparison which is the second ranked algorithm in CEC2017 competitions for the single objective optimization track. The algorithm is shown to outperform LSHADE [163] (the winner of the CEC2014) and its new extension for CEC2016 (iL-SHADE [25]).

2. The codes for CEC problems and the jSO algorithm are publicly available at: http://www.ntu.edu.sg/home/EPNSugan/index_files/CEC2017/CEC2017.htm

3. F2 has been excluded by the organizers because it shows unstable behavior especially for higher dimensions [9]

4.5.3 Experimental Settings

All the results are taken from the original study. In order to make a fair comparison, all the experiment conditions are the same. The number of function evaluations is $10,000 \times D$, where D is the problem dimension [9]. To tackle the negative effects of the random initial configurations, each algorithm were run 51 times [9]. The initial population is generated randomly within the search bounds $[-100, 100]$. The parameters of the jSO are the same as reported in the original study [26]. In MaNet, we have 3 convolution layers which are sequentially connected to each other. In each layer, the number of filters and the filter size are 6 and 3, respectively. The MaNet is a CNN model and needs a lot of input data to be well trained and so the population size is fixed to $n = 5,000$. Moreover, m is considered to be 64 for all the problems. The MaNet will be optimized using the Adam algorithm [88].

4.5.4 Results and Discussion

Tables 4.1 and 4.2 present best, worst, mean and standard deviation (Std.) results of the MaNet and jSO on 9 problems over 51 runs. Table 4.1 reports the results for 30 dimensional problems, while Table 4.2 shows the performance of the competitive algorithms for 50 dimensional cases. In these tables, a statistical test is also presented to assess the significance of performance between the results of the jSO and MaNet.

Table 4.1: The obtained results by MaNet and jSO for 30 dimensional problems over 51 runs [9]. The results for jSO are directly taken from the original paper [26].

		Best	Worst	Mean	Median	Std.	
F1	MaNet	3.71e+02	1.33e+03	7.94e+02	8.02e+02	2.03e+02	-
	jSO	0.00e+00	0.00e+00	**0.00e+00**	0.00e+00	0.00e+00	
F3	MaNet	3.69e+04	7.10e+04	5.85e+04	5.85e+04	6.46e+03	-
	jSO	0.00e+00	0.00e+00	**0.00e+00**	0.00e+00	0.00e+00	
F4	MaNet	1.46e-05	3.99e+00	**5.88e-01**	6.79e-04	1.41e+00	+
	jSO	5.86e+01	6.41e+01	5.87e+01	5.86e+01	7.78e-01	
F5	MaNet	0.00e+00	1.99e+00	**5.85e-01**	1.34e-07	6.59e-01	+
	jSO	3.98e+00	1.32e+01	8.56e+00	8.02e+00	2.10e+00	
F6	MaNet	0.00e+00	0.00e+00	**0.00e+00**	0.00e+00	0.00e+00	=
	jSO	0.00e+00	0.00e+00	**0.00e+00**	0.00e+00	0.00e+00	
F7	MaNet	3.26e+01	3.41e+01	**3.33e+01**	3.33e+01	3.91e-01	+
	jSO	3.61e+01	4.31e+01	3.89e+01	3.91e+01	1.46e+00	
F8	MaNet	0.00e+00	4.97e+00	**2.29e+00**	1.99e+00	1.15e+00	+
	jSO	4.97e+00	1.30e+01	9.09e+00	8.96e+00	1.84e+00	
F9	MaNet	0.00e+00	0.00e+00	**0.00e+00**	0.00e+00	0.00e+00	=
	jSO	0.00e+00	0.00e+00	**0.00e+00**	0.00e+00	0.00e+00	
F10	MaNet	1.09e+04	1.13e+04	1.11e+04	1.11e+04	1.19e+02	-
	jSO	1.04e+03	2.04e+03	**1.53e+03**	1.49e+03	2.77e+02	

The results of the Wilcoxon rank sum test are reported at the 95% confidence level. In these tables, + shows that MaNet significantly outperforms the jSO with 95% certainty; -

Table 4.2: The obtained results by MaNet and jSO for 50 dimensional problems over 51 runs [9]. The results for jSO are directly taken from the original paper [26].

		Best	Worst	Mean	Median	Std.	
F1	MaNet	3.67e+02	2.06e+03	1.39e+03	1.46e+03	3.71e+02	-
	jSO	0.00e+00	0.00e+00	**0.00e+00**	0.00e+00	0.00e+00	
F3	MaNet	9.80e+04	1.42e+05	1.23e+05	1.25e+05	8.88e+03	-
	jSO	0.00e+00	0.00e+00	**0.00e+00**	0.00e+00	0.00e+00	
F4	MaNet	3.10e-06	1.53e-03	**8.22e-04**	9.96e-04	4.46e-04	+
	jSO	1.32e-04	1.42e+02	5.62e+01	2.85e+01	4.88e+01	
F5	MaNet	1.99e+00	1.09e+01	**6.15e+00**	5.97e+00	2.20e+00	+
	jSO	8.96e+00	2.39e+01	1.64e+01	1.62e+01	3.46e+00	
F6	MaNet	0.00e+00	0.00e+00	**0.00e+00**	0.00e+00	0.00e+00	=
	jSO	0.00e+00	0.00e+00	**0.00e+00**	0.00e+00	0.00e+00	
F7	MaNet	5.49e+01	5.65e+01	**5.58e+01**	5.59e+01	3.62e-01	+
	jSO	5.75e+01	7.42e+01	6.65e+01	6.66e+01	3.47e+00	
F8	MaNet	1.99e+00	8.95e+00	**5.41e+00**	5.97e+00	1.99e+00	+
	jSO	9.95e+00	2.41e+01	1.70e+01	1.70e+01	3.14e+00	
F9	MaNet	0.00e+00	0.00e+00	**0.00e+00**	0.00e+00	0.00e+00	=
	jSO	0.00e+00	0.00e+00	**0.00e+00**	0.00e+00	0.00e+00	
F10	MaNet	1.86e+04	1.88e+04	1.87e+04	1.87e+04	6.25e+01	-
	jSO	2.40e+03	3.79e+03	**3.14e+03**	3.23e+03	3.67e+02	

indicates that the jSO is significantly better than MaNet; and = shows there is no statistical different between the two compared algorithms. The significant results are given in bold. For further validation, convergence graphs of jSO and MaNet for 30 dimensional functions F4 and F8 are given in Figure 4.2.

As can be seen from Tables 4.1 and 4.2, jSO gives more accurate solutions for the unimodal benchmarks F1 and F3 for both 30-dimensional and 50-dimensional cases. Moreover, with the exceptions of F10, MaNet has equal or significantly better performance on all the multimodal benchmark functions. In fact, the results indicate that MaNet significantly outperforms the jSO on 4 functions (F4-F8), obtains an equal performance on 2 functions (F6 and F9), and has worst results on 3 test cases (F1, F3 and F10). Furthermore, we can see that MaNet is a robust algorithm according to the reported standard deviation results. In addition, these experimental results have confirmed that MaNet is not very sensitive to the increment of dimension and is scalable. Considering Figure 4.2, it can be seen also that MaNet has a more rapid convergence rate than the jSO algorithm for function F4 and F8. In MaNet, we assume that not selection, but rather the combination of different filters is the main source of evolution and that is the reason for having unstable convergence behavior on these functions.

Altogether, these promising results have confirmed that MaNet has a competitive results in comparison with one of the best designed algorithm for the CEC2017 problems. This is quite interesting because MaNet doesn't borrow any search strategy or components from the previously proposed methods for the CEC problems; including CMAES [114], DE,

jADE [180], SADE [129], SHADE [162], L-SHADE [163], i-LSHADE [25] and jSO.

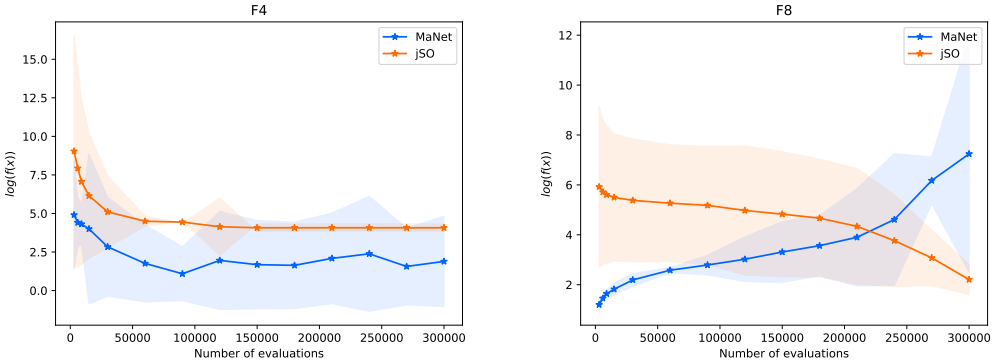


Figure 4.2: Convergence graphs of the jSO and MaNet for 30 dimensional functions F4 and F8 over 51 runs

As a future work, we are intended to apply the proposed MaNet to all the problems over all the dimensions. Besides, we have to find a way in order to adjust the learning rate hyperparameter for each problem. From Fig. 4.2 one can see that a high learning rate in Adam causes the network to generate large numbers for F8 and the updates are going to be just as large. After that, we would like to apply the proposed methodology to more complicated real-world optimization problems.

4.5.5 When Convolutional Neural Networks Does Not Help

The CNN models need to make numerous decisions about how the network is configured, connected, initialized and optimized. For example, a large value for learning rate will cause the training to diverge, while too small will allow the noise inherent in training to overwhelm the gradient estimates. In some way, they are suffering from the same problems in metaheuristics.

4.5.6 Implementation Notes

We used the Keras Open Source Neural Network library that supports most TensorFlow features. Going forward, we recommend that users consider switching to Pytorch. Keras doesn't handle low-level computation. Instead, it uses another library to do it, called the "Backend". So Keras is a high-level API wrapper for the low-level API. Pytorch, on the other hand, is a lower-level API focused on direct work with array expressions. It has gained immense interest in the last year, becoming a preferred solution for academic research, and applications of deep learning requiring optimizing custom expressions. Moreover, Pytorch has better debugging capabilities as compared to the Keras. We found it very difficult to implement the CEC2017 benchmarks and to customize the layers in Keras.

4.6 Practical End Use of Algorithm

The presented idea of directly solving difficult optimization problems using CNN neural networks can be easily applied to discrete and even multi-objective optimization. Moreover, we encourage to use the CNN models for automated algorithm design so as to help to develop and implement better algorithms. This can be very useful in situations where what have been learned on primary problem instances are exploited to improve generalization on the other similar problem instances; by virtue of further accelerating convergence rate.

4.7 Chapter Summary

This chapter introduced a new optimization algorithm based on the deep learning models in order to provide an improved search process. The proposed method verifies convergence conditions by using a convolutional neural network. The simple structure of the MaNet along with feature selection and dimension reduction strategies result in an architecture at a relatively low computational cost. The MaNet optimizer is evaluated using unimodal and multimodal optimization benchmarks from CEC2017 test suite. The obtained results are statistically analyzed and compared with state-of-the-art jSO algorithm. Evaluations confirm that the introduced MaNet optimization model has a competitive performance in terms of the final solution accuracy and scalability compared to one of the best designed algorithms for the problem at hand.

Chapter 5

Application IV: Automated Algorithm Design

> If I have seen further it is by standing on the shoulders of Giants.
>
> Isaac Newton

Neural architecture search (NAS) has recently drawn considerable attention to automatically build and evaluate low-latency networks for perceptual tasks. Here, we present a study on the performance of NAS for automated algorithm design. The space of network architectures is represented using a directed acyclic graph (DAG) and the goal is to find the best architecture so as to optimize the objective function for a new, previously unknown task. Different from proposing very large networks with GPU computational burden and long training time, we focus on searching for lightweight implementations to find the best architecture. The experiments reveal that NAS can achieve competitive results when compared to hand-designed algorithms; given enough computational budget.

The proposed method has several moving parts including convolution neural networks (CNN's) search space, search strategy, and search acceleration. We show how these different components could be adopted to achieve a better performance within a limited computational time. Finally, we claim that NAS can be used in other research fields and encourage further works in this domain. Our contributions can be summarized as follows:

— The NAS is formally defined tailored to global optimization. On top of that, a search strategy is used to perform NAS over a *cell-based* search space. We show that the well-known NAS approaches can be further enhanced by considering the key properties of the optimization problems.

— A set of experiments are conducted to investigate the performance of the proposed method. Our contribution achieves competitive performances for CEC 2017 [172] benchmarks and also protein structure optimization [134] compared to the state-of-the-art handcrafted algorithms.

— The transfer learning [111] and ensemble learning [48] concepts are used to show how the optimization process can be accelerated.

5.1 Motivation

Optimization algorithms have witnessed prevailing success in different application areas [134, 135, 167, 182]. Formally, they help to find a parameter vector x^* in order to

minimize an objective function $f(x) : \mathbb{R}^D \rightarrow \mathbb{R}$, i.e. $f(x^*) \leq f(x)$ for all $x \in \mathbb{R}^D$, where D denotes the dimensionality of the problem. There is no a priori hypothesis about f and it should be treated as a black-box entity.

Heuristic algorithms offering guidance based on the problem-domain knowledge for optimizing function f. In order to design a new heuristic algorithm, having a team of human experts with a longstanding experience within the specific domain is necessary. Usually, this is a very complex process performed with trial and error. As a consequence, the development of hyper-heuristics which do not take advantages of the problem structure accelerated [22, 144]. They are high-level methods that operate on the search space of heuristics rather than of solutions. Over the last decade, a variety of hyper-heuristic approaches have been proposed to automate the development of optimization methodologies on their own without having to rely on researchers' expertise [27].

The majority of popular hyper-heuristics are deployed according to the basic components of the hand-designed evolutionary algorithms [122]. We often have a population of candidate solutions that strive for survival and reproduction. However, there are no clear guidelines on the strengths and weaknesses of the proposed components arise in other research filed such as machine learning; for developing more enhanced optimization algorithms. In this study, we claim the mentioned contribution by porting existing NAS methods from image classification to optimization domain.

NAS is currently one of the fastest growing topics in machine learning aims to automate the neural network architecture design for various tasks such as semantic segmentation [121], object detection [169], and image classification [112]. This optimization process has focused on discovering better modeling accuracy, building architectures with lower computational complexity, or both of them. Search space, search strategy (or policy), and search speed-up methods are three main components in NAS approaches [53]. Already by now, outstanding results have been achieved using NAS that are superior to the expert-designed architectures [188].

Motivated by the recent successes of NAS, we propose to extend NAS studies to the global optimization domain. That is, we build and train neural networks to efficiently and adaptively solve optimization problems. The critical contribution of this study is to tackle two major challenges that are known to the direct application of NAS for stochastic optimization. First and foremost, the widely-used search speed-up methods in NAS such as parameter sharing [113] might not be suitable for the optimization problems. Although the parameter sharing avoids training each architecture from scratch, it would lead to unstable and suboptimal solutions as discussed in [102]. In some cases, this could end up with solutions with worse performance compared to the conventional methods in the literature; considering the complex and highly non-linear real-world optimization problems. Second, the existing NAS methods introduce a domain-specific bias with search spaces tailored to particular applications [53, 183]. For example in computer vision, NAS is defined to search for the convolutional and fully connected layers in CNNs; without fine-tuning the hyperparameters in the learning algorithm. However, this may prevent finding superior architectures that go beyond the classification tasks. Integrating prior knowledge about typical properties of optimization problems can characterize the complexity of the search

space more properly and simplify the search. Even so, the search space is huge and may contain more than 10^{15} different architectures [188].

Particularly, we are interested to adopt the proposed NAS methods from the computer vision domain. These deep neural models use a hierarchy of features in conjunction with several layers to learn complex non-linear mappings between the input and output layers. As opposite to traditional methods which use handmade features, the important features are discovered automatically and are represented hierarchically. This is known to be the strong point of CNNs against traditional approaches. Accordingly, these models have been described as universal learning approaches that are not task specific and can be used to tackle different problems arise in different research domains [4, 131]. They are regularized version of fully-connected neural networks inspired from biological visual systems [94]. The "fully-connectedness" of CNNs enables them to tackle the over-fitting problem and it is reasonable to postulate that they may outperform classical neural networks for difficult optimization tasks [131].

5.2 Problem Definition

We aim to find a topology that minimizes a considered objective function $f(x)$ over a neural search space $\mathcal{D}$ with the available computational budget $\mathcal{T}$. Formally, this is equivalent to search for a superior neural architecture $\mathcal{A}^* \in \mathcal{D}$:

$$\mathcal{A}^* = arg \min_{\mathfrak{A} \in \mathcal{D}} cost(\mathfrak{A}, f, w, \mathcal{T}) + \xi \tag{5.1}$$

where w is the learned weights of $\mathfrak{A}$ and ξ is a penalty function. The measure of violation in ξ is nonzero when the number of edges ϑ in the DAG graph is > 9 or when there is no path from the input to the output layer; and is zero in the other cases. The mathematical function associated with ξ is:

$$\xi = \begin{cases} (\vartheta - 9) \times \eta_1 & if\ \vartheta > 9 \\ \kappa \times \eta_2 & otherwise \end{cases} \tag{5.2}$$

In Eq. 5.2, κ denotes the number of single nodes and η_2, η_1 are the penalty coefficients. Before applying any search regime, the representative DAG graph for arbitrary $\mathcal{A}$ network should be encoded in its genotype form. We adopt a very general encoding: the first 21 binary genes $\in \{0, 1\}$ are used to represent the edges in the graph, while other 5 genes $\in \{0, 1, 2\}$ are used to represent the type of the operation. Also, the last gene denotes the batch size hyperparameter.

5.3 Related Works

Several works introduced deep neural networks to find optimum solution for the optimization tasks. DNGO [153] uses deep neural networks for hyperparameter optimization of large scale problems with expensive evaluations. The key point is to take the advantages of large-scale parallelism to provide an approximate model of the real objective function. The

scalability of DNGO is successfully verified against Gaussian process. Moreover, OptNet [5] proposed to learn optimization tasks by incorporating the deep networks, bi-level optimization, and sensitivity analysis. In another study, researchers put forward MaNet optimization algorithm based on the CNN models [131]. MaNet uses feature selection to skip irrelevant or partially relevant information and uses those which contribute most to the overall performance. The experiments indicate that MaNet is able to yield competitive results compared to one of the best hand-designed algorithms for the CEC 2017 problems [172], in terms of the solution accuracy and scalability.

Overall, all these studies make it likely that an optimization algorithm based on neural networks can find solutions that substantially outperform the state-of-the-art optimization methods. We thus found it important to go beyond hand-designed algorithms for optimization tasks by applying NAS to this less explored domain. Among different models, NAS equipped with CNN models is used in this research. A CNN is a deep learning model that interleaves convolutional layers to filter redundant or even irrelevant input data to increase the performance of the network [64]. This consideration also reduces the dimensionality of the input data and speeds up the learning process in the CNNs. Besides, it allows CNNs to be deeper networks with fewer parameters.

5.4 Methodology

Our immediate aim is to examine properties of the NAS on optimization problems. Such an implementation involves search space definition, search strategy, and search speed-up in parallel on GPUs; as illustrated in Figure 5.1. We combine the efficiency of multi-GPU systems with NAS to balance computational efficiency and the solution quality. Due to its distributed nature, we are able to deploy large-scale number of deep networks while learning different problems. Empirically, we show that the introduced method obtains better results with reductions in search complexity. The proposed methodology is related to several prior works, mainly including DARTS [113], NAS-Bench-101 [176], ASHA [109], randomNAS [105] and [175]. The performance of the introduced method confirms concerns raised in this study that state-of-the-art results can be obtained by using the NAS. In the following, it is assumed that the reader is familiar with the basic concepts of optimization algorithms, CNNs, and artificial neural networks.

5.4.1 Preliminaries

The state-of-the-art NAS methods can be parametrized by: (i) search space; (ii) search strategy; and (iii) search speed-up methods [53]. For simplicity, we review some ubiquitous approaches regarding each aspect. Note that we are interested in CNN models, and so approaches about recurrent neural networks are out of the scope of this work.

Search space aims to define the feasible neural architectures based on the applications and computation requirements. In *linear-structured* NAS [28], search space is defined as a sequence of m layers, so as the input of the L_i layer is fed by previous layer L_{i-1} (Figure 5.2 top). Accordingly, the number of layers, type and hyperparameters associated with each

layer will form the possible architectures search space. *Multi-branch* methods, on the other side, allow the researchers to build the complex architectures with significantly more degrees of freedom. These methods are motivated by the Residual networks [64] and DenseNets models [68] which introduced skip connections (Figure 5.2 bottom).

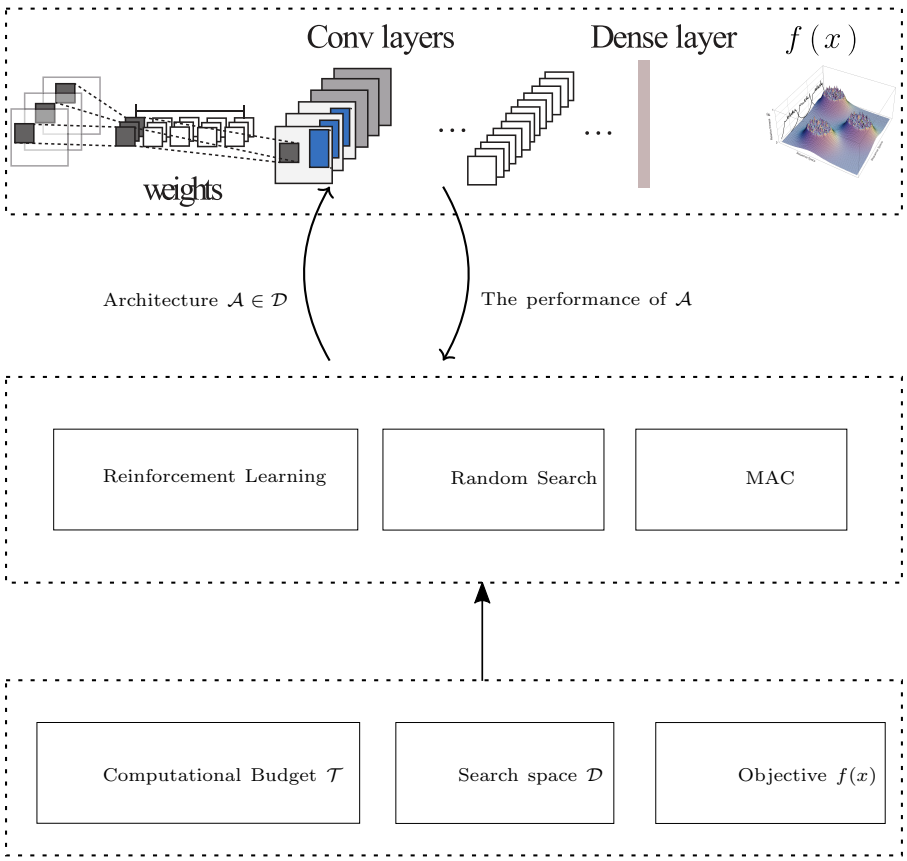


Figure 5.1: Illustration of the abstract layers for the NAS and the optimization phases. First, the computational budget, search space, and the objective function f should be specified. Thereafter, we will generate and train neural architectures in parallel on several GPUs so as to solve the problem at hand. In this study, reinforcement learning, random search, and MAC search strategies are applied.

In the same direction, *cell*-based approaches aim to formulate the search space by stacking several copies of the discovered *cells*, which significantly reduced the size of the search space since *cells* have less layers than final architectures. In *micro search*, the whole architecture is built by stacking the *cells* in a predefined manner [178], while in *macro search* they can be combined arbitrarily [29].

Search strategy is then used to explore the above-mentioned space of neural architectures. Typical NAS approaches apply reinforcement learning [187], evolutionary algorithms [52], Bayesian optimization [80] and random search [137] to reduce the computational costs, improve the performance, or obtain a trade-off.

Search speed-up strategies lead to accelerated NAS methods for training the neural architectures, which sometimes need thousands of GPU days for NAS [187]. The lower

fidelity [137], learning curve [10], weight inheritance [52], and weight sharing [174] are among the most recent approaches.

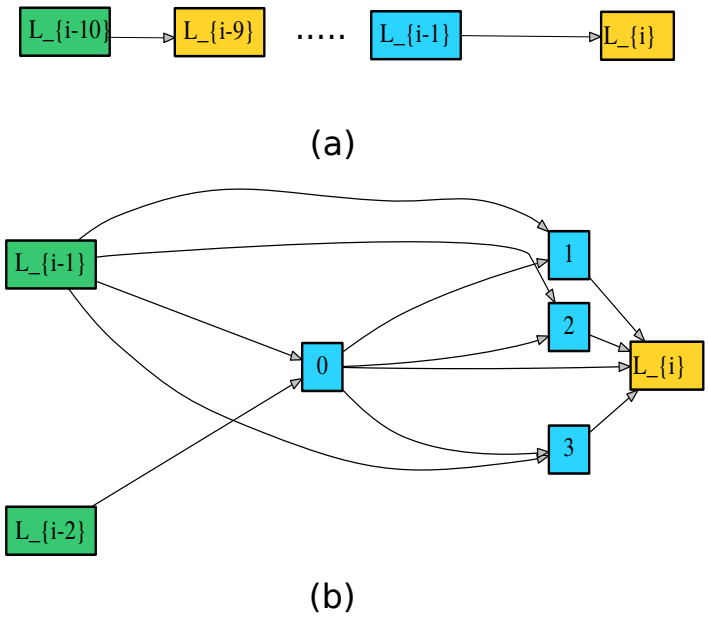


Figure 5.2: Illustration of the *linear-structured* (a) and *multi-branch* (b) architecture search spaces. In the first case, NAS only allows data to flow in one direction: from a lower-numbered layer L_i to a higher numbered layer L_{i+1}. The *multi-branch* methods, however, allow to use multiple branches and skip connections. Note colors are used to represent different kind of operations at each layer.

5.4.2 Search Space

The *input layer*, *intermediate layers* (e.g., convolution layer, a pooling layer, a Dense layer, etc.), and *output layer* are the basic building blocks of the CNN models. The inputs to *intermediate layers* are fed by a previous layer, thus forming a network. It is necessary to define the topology of this network before deploying a CNN model in the context of the optimization problem we are trying to solve. Some network structures might lead to the final networks that are highly memory demanding and time-consuming [61]. The key point so is to define a search space that makes optimal use of computational resources and reduces the probability of generating sub-optimal network architectures.

Following [113], we limit the NAS search space by factorizing each architecture into *multi-branch cells*. This representation is characterized by the number of layers m, choice of operation $O = [o_1, o_2, ..., o_n]$ for each layer L, and θ_{L_o} which denotes the associated hyperparameters for the operation o at the L-th layer. At a higher level, the entire network is defined by m *cells* connected sequentially, which can be seen in Figure 5.3. A *cell* is composed of one *input layer*, several intermediate nodes, and one *output layer*. Specifically, the *input layer* is connected to the output of the previous *cell* layer. Moreover, the *output layer* aggregates the representations from all the intermediate nodes. The layouts of the

intermediate nodes can be defined using a directed acyclic graph (DAG), where a node contains the results from a previous operation and an edge e_{ij} shows some operation o that transforms the feature map from node I_i to I_j. Thus we have intermediate nodes:

$$I_j = \sum_{i<j} o_{i,j}(I_i) \tag{5.3}$$

where $o_{i,j}$ denotes the selected operation from lower indexed node i to higher node j. At each layer L, one of three possible operations *convolution*, *max pooling* and *average pooling* from CNN models can be chosen. In the following, each of these layers are characterized so as to fully parameterize the neural architecture space.

A typical CNN network is composed of a series of convolutional and pooling layers. Generally speaking, convolutional layers provide a way of capturing the dependencies in their input by applying different filters. For a given filter and the input data, the convolution operation takes entries with size $p \times p$ of the input and multiplies by the filter. The sum of the entries is then the first entry of the so-called feature map. The weights of the filters are adopted during the training process, while the number of filters and their size should be configured. Given the filter weights, we create a sliding window $p \times p$ that goes by step size s through the vertical and horizontal dimensions of the input data. The hyperparameters of convolutional layers are: a) the number of filters $n \geq 1$, b) size of the slide window $p \times p \geq 1$, and c) the stride step size $s \geq 1$. For each filter, a fixed weight will be used across the entire input. A one layered CNN with $n = 10$ filters of size 5×5 and 10 biases has $5 \times 5 \times 10 + 10 = 260$ parameters, while a fully connected network for a $K = (P \times M)$ image with 250 neurons has $(250 \times K + 1)$ parameters. This is the main advantage of CNNs which makes them more efficient in terms of memory and complexity; compared to fully connected neural networks.

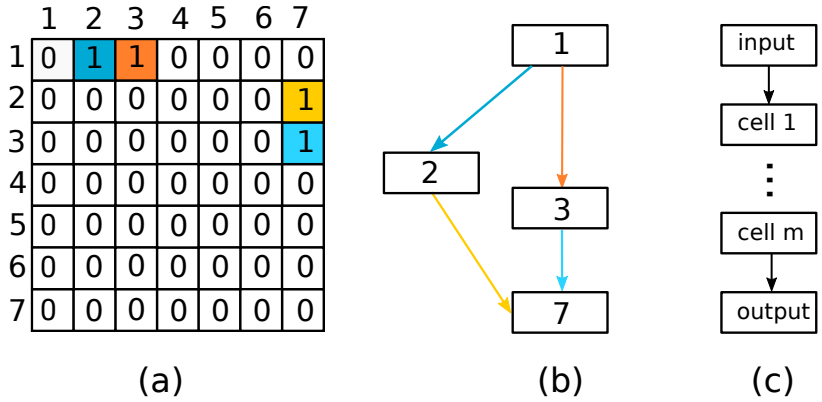


Figure 5.3: An overview of the *cell*-based search schema: (a) The adjacency matrix used to represent the DAG. Here, 1 and 7 indexes belong to the *input* and *output* layers of the defined *cell*, followed by the *intermediate nodes* 2-6, (b) The obtained *cell* structure, where operations at each vertex are denoted by a different color, and (c) The derived final architecture with m *cells*.

Similar to convolutional layers, pooling provides another way to reduce the dimension of

a layer. Although they can be replaced by the convolutional layers, they provide a simpler way by summarizing a $p \times p$ area of the input with certain fixed weights. For an average pooling layer with n feature maps, we should have a convolutional layer with n filters of size $p \times p$ and stride s. The ith filter has the values as in Eq. 5.4 for the dimension i, and zero for the other dimensions $i \in [1, n]$.

$$\begin{pmatrix} \frac{1}{p^2} & \cdots & \frac{1}{p^2} \\ \vdots & \ddots & \vdots \\ \frac{1}{p^2} & \cdots & \frac{1}{p^2} \end{pmatrix} \tag{5.4}$$

The same thing can be considered in max pooling, where the maximum value within the window is taken with filter weights 1. It is not necessary to pool over the whole input and we can pool over a window with stride step size s. So, we have only two hyperparameters p and s for the pooling layer. To sum up, the functionality of a simple convolution layer and max pooling layer is depicted in Figure 5.4.

Besides the mentioned operations, the CNN learning algorithm itself contains a set of hyperparameters. The batch size is a hyperparameter of the learning algorithm that controls the number of data that will be propagated through the network. An appropriate batch size can increase the accuracy of the learning algorithm when training a neural architecture. We found that fine-tuning this hyperparameter can have a significant impact on the performance of the NAS (we will show that this consideration is preferred over traditional NAS methods that use a fixed value). So, two typical choices of batch size $\in \{1, 32\}$ are used in this study.

We now quantify the size of our search space to determine the magnitude of the proposed NAS method. A comparison of the search space complexity for state-of-the-art NAS methods is given in Table 5.1. The space of the *cell* networks contains all DAG graphs on v nodes, where each node denotes one operation with $p = 3$ and $s = 1$. In this work, the number of operations is limited to: a) one convolutional layer, b) one max pooling layer, and c) one average pooling layer. Moreover, the maximum number of nodes in each *cell* is supposed to be ≤ 7. Also, the maximum number of edges is limited to 9. Considering 21 possible edges in DAG adjacency matrix, 3 operations for each intermediate node, and 2 different values for batch size, $2^{21} \times 3^5 \times 2 \approx 1.0 \times 10^9$ total models exist in this search space. The created models do not apply $ReLU$ activation function or batch normalization between depthwise and pointwise layers. To match shapes in convolutional layers, strided 1×1 convolution projections are applied as necessary. Furthermore, the output of the intermediate blocks are concatenated.

5.4.3 Search Strategy

Up to our best knowledge, this chapter describes the first attempt to extend NAS for optimization domain. Accordingly, 1) random search, 2) reinforcement learning, and 3) Bayesian optimization strategies are used to leverage the observations of previous studies, respectively. As shown in Algorithm 2, they all operate in a similar fashion: in each iteration i a step vector $\Delta\mathfrak{A}$ is computed by means of an update formula ϖ. Thereafter, this formula is updated using Φ to guide the search process more effectively. For the reinforcement

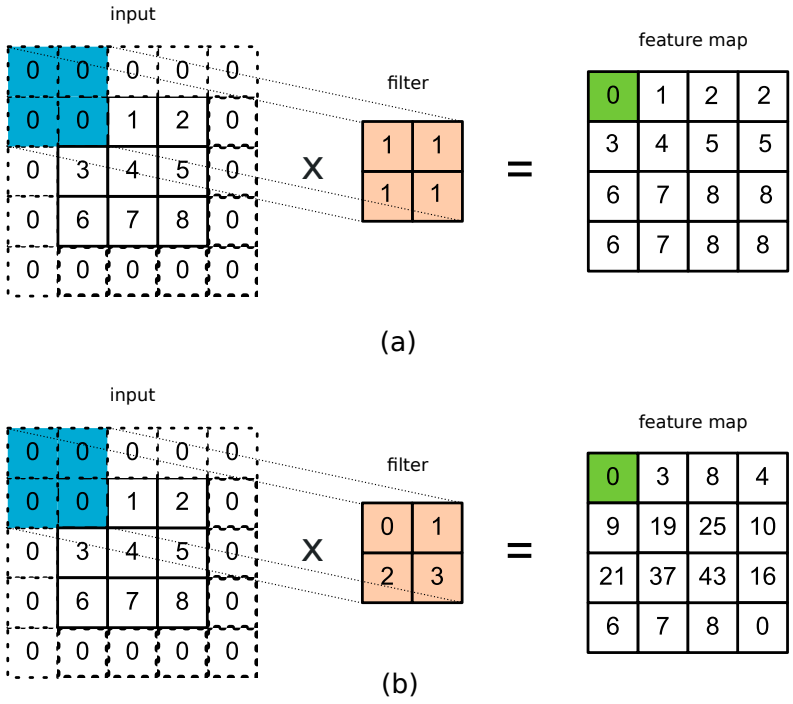


Figure 5.4: Application of a single max pooling layer (a) and a convolutional layer (b) with 1 filter of size 2×2 with stride $s = 1$ to input data of size 5×5. In (a), the maximum value after element-wise multiplication is taken, while in (b) the summation is computed. This figure also shows how the max pooling layer can be replaced by a convolutional one.

learning and Bayesian methods, Φ utilizes some history of the generated architectures and their associated performance evaluated at the current and past iterations. For example, the update formula is based on recurrent neural networks in the second method, while reinforcement learning is used to update the aforementioned network according to the past information.

1-Random search is the most simple yet effective [105] baseline in this study. We used our implementation according to which the generated candidates are drawn from a uniform probability distribution and are independent of the samples that come before it. This property makes it well suited to highly parallel systems. Moreover, random methods are flexible in that they can be applied to both the continuous and discrete search space; in contrast to Bayesian approaches based on Gaussian processes [90] and gradient-based approaches [113].

2-Reinforcement learning is another approach that is proposed to search for good architectures. Following [187], a recurrent network trained by reinforcement learning is used to generate better architectures; as times goes on.

The recurrent networks are able to use information learned from prior steps while generating new architectures. Let's suppose we would like to search for a *cell* topology, the agents action is to generate new architecture, while its reward is based on the performance

Table 5.1: Comparisons of the search space complexity between the introduced and state-of-the-art NAS methods.

Search Method	Number of Layers	Search complexity
EDNAS [97]	6	1.04×10^9
PNAS [112]	5	10^{12}
NASNet [188]	5	10^{28}
RENAS [33]	5	3.1×10^{13}
EPNAS [127]	5	5×10^{14}
STACNAS [102]	4	10^{18}
current study	5	1.0×10^9

Algorithm 2: General structure of the NAS methods

Input: NumIterations, SearchSpace, objective f
Output: BestCost
$\mathfrak{A}_1 \leftarrow$ RandomSolution(SearchSpace)
for $i \in \{2, ..\text{NumIterations}\}$ **do**
  $\text{Cost}_{i-1} \leftarrow \text{BestSolution}(\mathfrak{A}_{i-1}, f)$
  $\Delta\mathfrak{A} \leftarrow \varpi(\{\mathfrak{A}_j, f\}_{j=1}^{i-1})$ → $\varpi(.) = \begin{cases} 1\text{-random distribution} \\ 2\text{-recurrent network} \\ 3\text{-feature selection} \end{cases}$
  $\mathfrak{A}_i \leftarrow \mathfrak{A}_{i-1} + \Delta\mathfrak{A}$
  $\varpi \leftarrow \Phi(\varpi, \mathfrak{A}_i, f)$ → $\Phi(.) = \begin{cases} 1\text{-not applicable} \\ 2\text{-reinforcement learning} \\ 3\text{-surrogate model} \end{cases}$
end
BestCost $\leftarrow$ min(Cost)

of the trained architecture on our optimization problem. In Figure 5.5, it is shown how a recurrent network can be used to sample new architectures as a sequence of tokens. The list of the predicted actions $a_{1:T}$ by the controller will be used to generate a new architecture. This is the key feature of recurrent nets which allows us to operate over sequences of vectors in the input, the output, or both. [1]. Hence, the controller should only maximize its expected reward $J(\theta_c)$ which directly depends on the parameters of the recurrent net θ; as presented in [171]. We can use reinforcement learning as follows:

$$\bigtriangledown_{\theta_c} J(\theta_c) = \frac{1}{M} \sum_{k=1}^{M} \sum_{t=1}^{T} \bigtriangledown_{\theta_c} logP(a_t | a_{t-1:1}; \theta_c) R_k \tag{5.5}$$

In Eq. 5.5, M denotes the number of sampled architectures in one batch, T is the dimension of the problem, and R_k is the performance of the k-th neural network architecture after being trained.

3-Bayesian methods are widely used for hyperparameter optimization, but their application to NAS has been limited due to the fact that they mainly employ Gaussian process and they are not appropriate for high dimensional NAS [53]. In this work, we adopt a single-

1. For more details please see: `http://karpathy.github.io/2015/05/21/rnn-effectiveness`

objective version of MAC [132] as the Bayesian search strategy for NAS. Accordingly, we generate a set of random architectures in the early iterations in parallel, while we adopt a feature selection strategy to generate more promising candidates in the later stages of development. This is a key property in MAC that helps us to make a balance between the solution quality and the computational time. [2]

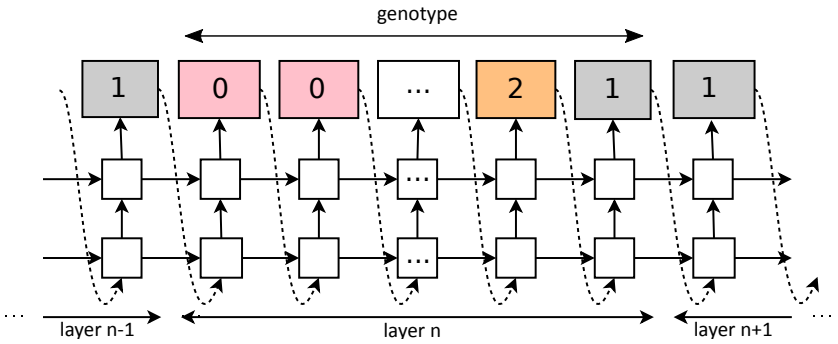


Figure 5.5: Application of a recurrent neural network for sampling a new *cell* topology [187]. The generated networks in the next time steps are influenced by what the network has learned from the past.

After half of the iterations, a response surface model is created to provide a fast approximation of the expensive evaluations for the later stages of evolution. Given a set of configurations $\mathfrak{A}_1, ..., \mathfrak{A}_N \in \mathbb{R}^{\mathcal{D}}$ with known performance $\mathbf{y}$, the radial basis function (RBF) interpolant for $\hat{\mathfrak{A}}$ is then computed as below [140]:

$$\tilde{y}(\hat{\mathfrak{A}}) = \sum_{i=1}^{N} \lambda_i \phi(\left\|\hat{\mathfrak{A}} - \mathfrak{A}_i\right\|) + p(\hat{\mathfrak{A}}),\ \hat{\mathfrak{A}} \in \mathcal{D} \tag{5.6}$$

Here, $\mathfrak{A}_i$ is the associated solution representation for i-th architecture model. Also, $\left\|.\right\|$ is the Euclidean norm, $\lambda_i \in \mathbb{R}$ for $i = 1, ..., N$, $p \in \prod_m^d$ denotes the linear space of polynomials in d variables of degree which is less than or equal to m, and ϕ is a RBF kernel. Following [140], MAC selected the *surface splines* $\phi(r) = r^k$ form with $k = 3$ as the RBF. Having this in mind, we can compute a matrix $\Im \in \mathbb{R}^{N \times N}$ by $\Im_{i,j} = \phi(\left\|\mathfrak{A}_i - \mathfrak{A}_j\right\|)$; $i, j = 1, ..., N$. Assume that $\hat{m}$ is the dimension of the linear space $\prod_m^d$ such that $m \geq = \lfloor k/2 \rfloor$. Accordingly, we have another matrix $\mathbf{P} \in \mathbb{R}^{n \times \hat{m}}$ such that: $P_{ij} = p_{(i)}(\mathfrak{A}_{(i)})$, $i = 1, ..., N; j = 1, ..., \hat{m}$. The approximated model can then be obtained by solving the system as presented in Eq. (5.7), where $\mathbf{c} = (c_1, ..., c_{\hat{m}})^T \in \mathbb{R}^{\hat{m}}$.

$$\begin{pmatrix} \Im & \mathbf{P} \\ \mathbf{P}^\top & 0 \end{pmatrix} \begin{pmatrix} \gamma \\ \mathbf{c} \end{pmatrix} = \begin{pmatrix} \mathbf{y} \\ 0_{\hat{m}} \end{pmatrix} \tag{5.7}$$

After half of the iterations, the randomly generated perturbations ρ are also ranked in descending order according to their contribution to the validation accuracy: the top 50%

2. The application of MAC to very recent benchmarks against Random search, SMAC and F-RACE state-of-the-art competitors, at most contributes to top performances. These results encourage us to utilize MAC in this study.

with *promising* label and the others with *non-promising* label. Given this training set, MAC applies a feature selection strategy to obtain weight for generating the new configurations; as elaborated in [132].

5.4.4 Search Speed-up

The above mentioned architecture $\mathfrak{A}_i$ should be trained on each function f which can take a very long time. However, we would like to solve our optimization task with roughly the same wall-clock time needed for an evolutionary algorithm. To tackle this challenge, we help ground the used methods by introducing two considerations: 1) early-stopping and 2) identifying isomorphic computational graphs. In the first case, we stop training the model once its performance stops improving on the fitness function f. Accordingly, we define a threshold ψ to whether consider a function value at some epoch as improvement or not. If the difference of improvement compared to the previous training epoch is below ψ, it is quantified as no improvement. This very simple paradigm can prevent the architecture to spend the computational time on sampling in non-optimal regions. Moreover, it only needs a few modifications and is easy to implement. In particular, the idea is to 1) assign a small computational budget r to the sampled architecture, 2) train the architecture, 3) increase the budget for architecture by a factor of η, and repeat until the maximum budget of ι for the architecture is reached or its performance stops improving.

The second consideration is inspired by [175, 176] according to which we check isomorphic computational *cells* before evaluating the generated model. Of course, there is no guarantee to perfectly find all pairs of non-isomorphic *cells*, but this will work in many cases. The key point is to reduce the size of the search space by detecting the *cells* which have different genotype but encode the same computation. This strategy could significantly reduce the size of our defined search space. Moreover, it can help the RL and MAC from being misled to a false optimum by using the information from the model.

5.4.5 Putting It All Together

Already by now, we described how NAS can be applied to generate different architectures within the defined search space. In this subsection, we would like to put all of the details together to show how such encoded architecture $\mathfrak{A}$ can be employed to solve an optimization problem. Since we assume that not all readers are fully familiar with machine learning, we used the optimization terminologies as a common language to describe all the aspects.

The first principle is that in CNNs you present your input image and train your model to make predictions on unseen data. This implies that our population is represented by a set of $NumSol$ random $n \times n$ inputs for the model (i.e., the raw pixel values of the image). So, as opposite to evolutionary algorithms, each individual solution $x \in \mathbb{R}^D$ is represented by a matrix with any arbitrary size $n \times n \geq D$ rather than a vector. During the training of the network, the defined convolutional operations transform this $n \times n$ matrix, layer by layer, to a final feasible solution $x \in \mathbb{R}^D$. This large part genotype representation enables the optimizer to keep genetic information that was necessary in the past as a source of exploration, as well as a playground for extracting new features that can be advantageous

in the exploitation. The second important point to note is that CNNs only modify their weights and the input data are kept fixed, while evolutionary algorithms modify their initial population. They adopt gradient descent to update these weights based on the backpropagation of error algorithm. Here, the distance from optimal $f(x^*) - f(x)$ is used to calculate the model error.

The training process of a neural network using Adam optimizer [89] for an objective function $f(x)$ is summarized in Algorithm 3. These steps are simplified and we just tried to provide intuition into the training process. In Algorithm 3, $\Delta_w E_t(w_{t-1})$ denotes the partial derivatives of E_t with respect to w at time step t, α is the learning rate, and β_1, β_2 hyper-parameters are the exponential decay rates of the m_t and v_t moving averages, respectively [89].

Algorithm 3: Optimizing objective f using Adam

Input: NumSol, n, f, α, β_1, β_2, $\mathfrak{A}$
Output: Best
$m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $t \leftarrow 0$
$w_t \leftarrow$ RandomWeights($\mathfrak{A}$)
Best $\leftarrow inf$
repeat
  $x \leftarrow \mathfrak{A}(w_t, NumSol, n)$
  Best $\leftarrow \min(\text{Best}, f(x_i)); i = 1, ..., \text{NumSol}$
  $t \leftarrow t + 1$
  $E_t(w_{t-1}) = \frac{1}{\text{NumSol}} \sum_{i=1}^{\text{NumSol}} f(x_i) - f(x^*)$
  $g_t \leftarrow \Delta_w E_t(w_{t-1})$
  $m_t \leftarrow \beta_1 \times m_{t-1} + (1 - \beta_1) \times g_t$
  $v_t \leftarrow \beta_2 \times v_{t-1} + (1 - \beta_2) \times g_t^2$
  $\hat{m}_t \leftarrow \frac{m_t}{(1-\beta_1^t)}$
  $\hat{v}_t \leftarrow \frac{v_t}{(1-\beta_2^t)}$
  $w_t \leftarrow w_{t-1} - \frac{\alpha \times \hat{m}_t}{(\sqrt{\hat{v}_t} + \epsilon)}$
until *stopping criteria are met*;

Let's see how forward propagation step can be used to generate a 9-dimensional solution vector x. Suppose that we have a model with input matrix $n \times n = 5 \times 5$, $NumSol = 1$, one convolution layer, and one max pooling layer. If we use a $k \times k = 2 \times 2$ filter with stride size $s = 1$ and padding size $p = 0$, the output of the first layer will be of size $o(\mathfrak{A}, conv) = \frac{n-k+2p}{s} + 1 = \frac{5-2+2\times 0}{1} + 1 = 4$. Thereafter, we apply the max pooling with the same hyperparameters and we have $o(\mathfrak{A}, max) = \frac{o(\mathfrak{A}, conv)-k}{s} + 1 = \frac{4-2}{1} + 1 = 3$. We can see how the neural network $\mathfrak{A}$ is used to reduce the dimensionality of the input; $\mathfrak{A} : \mathbb{R}^{5\times 5} \rightarrow \mathbb{R}^{3\times 3}$. The output of the max pooling layer which is $x \in \mathbb{R}^9$ vector then forms our genotype for computing the error value.

5.5 Experiments

In this section, we conduct a pipeline of experiments to answer the following questions:

– How effective is the optimization with neural networks?

- What are the influences of the different NAS search strategies?
- Is the proposed methodology is scalable?
- How much efficiency is gained from using a trained network to solve another similar problem?
- How much efficiency is gained from using an ensemble network to solve another similar problem?
- What are the influences of transferring the learned knowledge for solving several problems; to a new similar task.

To provide a fair comparison, same settings, computational resources and budgets are adopted for all the results. The experiments are performed by using the parallel power of graphics cards [3].

5.5.1 Benchmark Sets

The elaborated NAS methodology is applied to learn architecture for two different optimization problems. We conduct experiments based on CEC 2017 benchmarks to assess the performance of the RS, RL and MAC strategies. In the same section, the results are compared against the progressive neural architecture search (PNAS) [112]. We used a set of 9 particularly challenging unimodal (F1 and F3) and multimodal (F4−F10) functions [4]. These problems are first implemented on GPU and are then linked with Tensorflow machine learning library [5]. All CEC test functions should be minimized within the search ranges $[-100, 100]^{\mathrm{D}}$.

Next, the introduced method is compared with state-of-the-art hand-designed algorithms for real-world PSP sequences from protein data bank [6]; given in Table 5.2. The AB off-lattice model is used to define the problem in continuous search space according to which we can predict the secondary structure of a protein using its amino acids sequence. This secondary conformation is characterized by the bond angles $[\theta_2, \theta_3, \theta_4, \cdots, \theta_{n-1}]$, where n is the number of bonds and $\theta_i \in (-180, 180]$. The optimization task then is to minimize the free energy of a protein sequence as:

$$\sum_{i=1}^{n-2} \frac{1 - cos\theta_i}{4} + \sum_{i=1}^{n-2} \sum_{j=i+2}^{n} \left[r_{ij}^{-12} - C(\zeta_i, \zeta_j) \times r_{ij}^{-6} \right] \tag{5.8}$$

where r_{ij} denotes the distance between i-th and j-th monomer; as given in [157].

5.5.2 Baselines

First, the performance of NAS for automated algorithm design on numerical benchmarks CEC 2017 are evaluated. The experimental results for 50 and 100 dimensional problems are

3. Operating system: GNU Linux, CPU: Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz, Tesla K40c , Main memory: 16 GB, GPU memory, 12 GB, Programming language: Python

4. Function F2 has been excluded by the organizers because it shows unstable behavior especially for higher dimensions [172].

5. `https://www.tensorflow.org`

6. `https://www.rcsb.org`

Table 5.2: The details of protein sequences used in experiments

No.	Length	PDB ID	Sequence
1	13	1BXP	ABBBBBBABBBAB
2	13	1CB3	BABBBAABBAAAB
3	16	1BXL	ABAABBAAAAABBABB
4	17	1EDP	ABABBAABBBAABBABA
5	18	2ZNF	ABABBAABBABAABBABA
6	21	1EDN	ABABBAABBBAABBABABAAB
7	21	1DSQ	BAAAABBAABBABABBBABBB
8	24	1SP7	AAAAAAAABAAABAABBAAAABBB
9	25	2H3S	AABBAABBBBBABBBABAABBBBBB
10	25	1FYG	ABAAABAABBAABBAABABABBABA
11	25	1T2Y	ABAAABAABBABAABAABABBAABB
12	26	2KPA	ABABABBBAAAABBBBABABBBBBBA
13	29	1ARE	BBBAABAABBABABBBAABBBBBBBBBBB
14	29	1K48	BAAAAAABBAAAABABBAAABABBAAABB
15	29	1N1U	AABBAAAABABBAAABABBAAABBBAAAA
16	29	1PT4	AABBABAABABBAAABABBAAABBBAAAA

reported. The results are evaluated against the state-of-the-art jSO algorithm. The extensions of the differential evolution are always among the winners of the CEC competition. The algorithm is shown to outperform LSHADE [163] (the winner of the CEC 2014) and its new extension for CEC 2016 (iL-SHADE [25]) which motivated us to consider jSO [23] algorithm for the purpose of comparison.

Next, we shift our focus to investigate the performance of NAS for PSP. All the experiments are repeated for 30 independent runs using the AB off-lattice model. The human-designed IFABC [101], LSHADE [163] and SGDE [134] algorithms for protein structure optimization are considered. The best, worst, mean and standard deviation of the results are reported. To provide a fair comparison, the parameters of the competitive methods are set according to the original works.

5.5.3 Experimental Settings

All the experiments are performed using $\mathcal{T} = 100,000 \times D$ fitness evaluations. As it is explained before, the generated neural architectures should be trained so as to solve the problem at hand. In NAS, gradient-based Adam optimizer is used to update the networks' weights with a learning rate of 0.001, $\eta = 0.01$. The other hyperparameters are set according to [89]. For PNAS, we followed the training procedure used in [112]. However, all the normalization layers are removed and the activation functions are set to None; which is the same consideration that we applied to our methodology. The parameter details of PNAS are presented in Table 5.3, while the operation space is given as follows [112]: a) 3×3, 5×5, and 7×7 depthwise-separable convolutions, b) 1×7 followed by 7×1 convolution, c) identity, d) 3×3 average pooling, e) 3×3 max pooling, and finally 3×3 dilated convolution.

Regarding the search methods, the maximum number of epochs to train the model is set to 200 epochs. We used the original settings for both the RL and MAC methods. In RL, a two-layer RNN controller with 35 hidden units is presented. The Adam with a

learning rate of 0.1, weight decay of $1.0e-04$, and momentum of 0.9 is adopted [187]. The reward used for updating the controller is the mean error value that is propagated in a single batch. In MAC, the number of generated trial samples at pre-evaluation step is $10,000$. Accordingly, MAC builds and trains a surrogate to find the most promising architecture using the surrogate modeling to replace in part the original computationally expensive solver; which is training the neural network for each candidate.

Table 5.3: The parameter configuration of PNAS

Description	Configuration
Maximum number of epochs to train the model	200
Dimension of the embeddings for each state	20
Number of epochs to train the controller	30
Number of children networks to train	8
Learning rate for the child models	0.001
Number of cells in RNN controller	100
Batch size of the child models	32
Number of blocks in each cell	3
Activation function	None

5.5.4 Results and Discussion

This section reports the empirical evaluation of the NAS search methodology on 9 standard benchmark functions from the global optimization literature. In this section, we aim to assess the approach's exploration performance and so the comparisons are based on the best obtained results over 15 runs. The upper and lower bounds for all the test functions are the same and the input data to the neural networks are 500 randomly sampled $n \times n = 32 \times 32$ matrix. We refer the reader to the original material for a description of the test functions and their properties [172].

Table 5.4: The performance of different search methods for NAS on CEC 2017 30 dimensional test cases. A pair-wise comparison between the MAC search strategy and the other competitive methods is also presented using Wilcoxon signed-rank test with $\alpha = 0.05$.

No	MAC	RS	RL	PNAS
F1	$1.13118e-02$	$1.40977e-02$	$1.94292e-03$	$1.47442e+05$
F3	$1.27183e-03$	$1.78565e+04$	$1.02218e-01$	$5.95693e+04$
F4	$2.62499e-06$	$1.06736e-05$	$7.37121e-06$	$3.03500e+00$
F5	$3.98197e+00$	$3.97984e+00$	$9.94979e-01$	$1.68726e+02$
F6	$2.05215e-01$	$1.26378e-01$	$1.07848e+00$	$5.29290e+01$
F7	$3.40319e+01$	$3.42406e+01$	$3.41101e+01$	$2.13789e+02$
F8	$4.97486e+00$	$9.98603e-01$	$6.97262e+00$	$1.46752e+02$
F9	$1.70125e-11$	$2.80843e-11$	$4.41499e+00$	$6.62968e+02$
F10	$1.39714e+03$	$2.05124e+03$	$1.97163e+03$	$3.27547e+03$
$+\backslash=\backslash-$		$5\backslash 2\backslash 2$	$5\backslash 2\backslash 2$	$9\backslash 0\backslash 0$

In Table 5.4, the results for 30 dimensions are shown. We can see that the MAC search strategy preforms well for most of the problems, suggesting that a substantial speed up can

be provided by the value of the additional information available from its surrogate model. Meanwhile, we can see that RS performs surprisingly well on F8 ,whereas RL outperformed the other algorithms on F5. Compared to PNAS, all the extensions of the introduced methodology show competitive results with nearly an order of magnitude reduction in the objective function value. We argue that the incorporated batch size hyperparameter is the main reason for this performance improvement. We further analyze this performance by conducting a Wilcoxon signed-rank test between the MAC and the other search methods. In Table 5.4, symbol '+' denotes that the *null* hypothesis is rejected and MAC obtained a superior performance, symbol '−' shows an inferior performance, and '=' suggests no statistical difference between the pair-wised algorithms.

In Figure 5.6, convergence rate of the MAC, RS and RL search methods are shown, where the horizontal axis indicates the average objective value found by the competitive methods as a function of evaluations. We can see that MAC surpasses the performance of the RS beyond the first stages of evolution for both F4 and F10 functions. Regarding the final results, however, RS results in faster learning and eventually converges to a better NAS settings for F4. Moreover, the advantage of RL can be seen on $F10$, although it does not perform as well on the other test functions. This undesirable performance could be alleviated if the hyperparameters of RL is chosen to be configured properly for each problem; in which case the advantage of MAC and RS parameter-free methods can be better manifested. Having this in mind, all the following experiments will be done using the MAC search strategy.

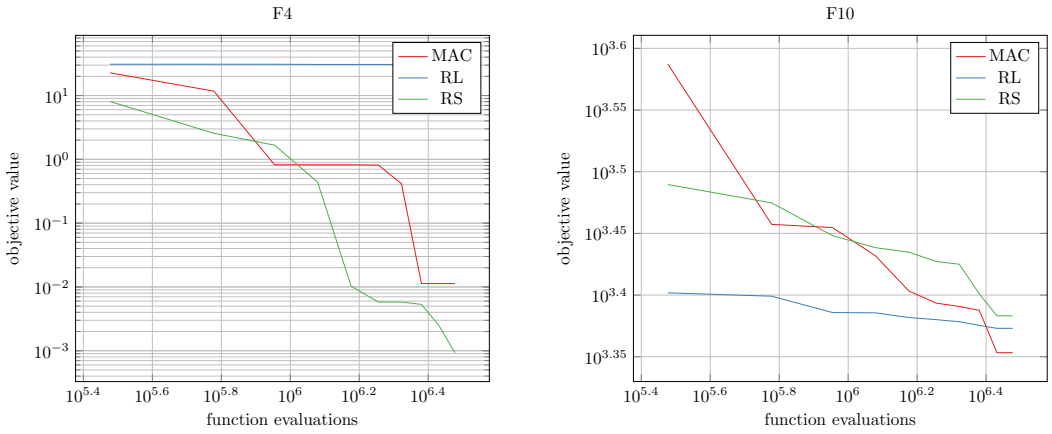


Figure 5.6: Illustration of the convergence results obtained for minimizing F4 (left) and F10 (right) 30 dimensional functions. We show the average objective value found by the competitive methods as a function of evaluations.

In Table 5.5, the experimental results for 50 and 100 dimensional problems are reported. The results are evaluated against the state-of-the-art jSO algorithm [7]. The extensions of the differential evolution are always among the winners of the CEC competition. The algorithm is shown to outperform LSHADE [163] (the winner of the CEC 2014) and its new extension for CEC 2016 (iL-SHADE [25]) which motivated us to consider jSO [23] algo-

7. The code for jSO is publicly available at: https://github.com/P-N-Suganthan/CEC2017-BoundContrained.git

rithm for the purpose of comparison. In order to make a fair comparison, all the experiment conditions are the same as mentioned before. The obtained performance for the unimodal function F1 indicates that jSO gives a more accurate range for 50 and 100 dimensional cases, although NAS has a more reasonable performance on 100 dimension. Conversely, it can be clearly observed that NAS has provided better results for F4 test problems. Furthermore, it can be seen that jSO converges closer to global optimum for 50 and 100 dimensional function F6. Concerning the other functions, the introduced NAS methodology achieved the lowest minimum values. The results also indicate that the NAS method can be also less sensitive to the increases in dimension as well as jSO evolutionary algorithm.

Table 5.5: The performance of NAS and jSO on CEC 2017 for 50 and 100 set cases using Wilcoxon signed-rank with $\alpha = 0.05$.

No	NAS (50)	jSO (50)	NAS (100)	jSO (100)
F1	$1.27460e+01$	$0.00000e+00$	$3.63306e-03$	$0.00000e+00$
F3	$1.66479e+04$	$0.00000e+00$	$1.79154e+04$	$0.00000e+00$
F4	$0.00000e+00$	$2.85127e+01$	$5.72086e-02$	$1.97357e+02$
F5	$4.63491e+00$	$1.19395e+01$	$1.76348e+01$	$2.48740e+01$
F6	$3.39670e-02$	$0.00000e+00$	$4.73965e-02$	$0.00000e+00$
F7	$4.69719e+00$	$6.10567e+01$	$1.04077e+01$	$1.28151e+02$
F8	$5.86165e+00$	$1.19395e+01$	$9.17544e+00$	$2.68639e+01$
F9	$0.00000e+00$	$0.00000e+00$	$0.00000e+00$	$0.00000e+00$
F10	$6.19778e+02$	$2.32882e+03$	$1.34090e+03$	$7.23914e+03$
$+\backslash=\backslash-$		$5\backslash3\backslash1$		$5\backslash3\backslash1$

Altogether, these promising results: (1) provide evidence for applying the neural networks on high dimensional optimization problems; and (2) show that the performance of modern machine learning components for designing new optimization algorithms can be highly significant. For example, the obtained results on F7, F9, and F10 functions are reported for the first time in this study. This is quite interesting because the NAS doesn't borrow any search components from the previously proposed methods for the CEC problems.

In Figure 5.7, the radar chart of the obtained solutions using NAS for 30 dimensional data in the form of a plot is presented. We can see that the best found solutions for the CEC problems are rather different or disparate from each other in the search space. However, it can be seen that the generated neural networks are able to learn the correlation between the decision variables. Accordingly, we can say that the introduced NAS methodology is not a simple biased search method that generates solutions only around a specific area in the search space.

In Tables 5.6 and 5.7, we presented the obtained results by transferring information from previous runs on the higher dimension problems. We show how the trained neural architectures for 30 dimensional problems can generalize well on higher dimensions 50 and 100. Our task is to take the best pre-trained architecture for each problem and transfer its weights to a higher dimension problems; rather than searching and training a model from

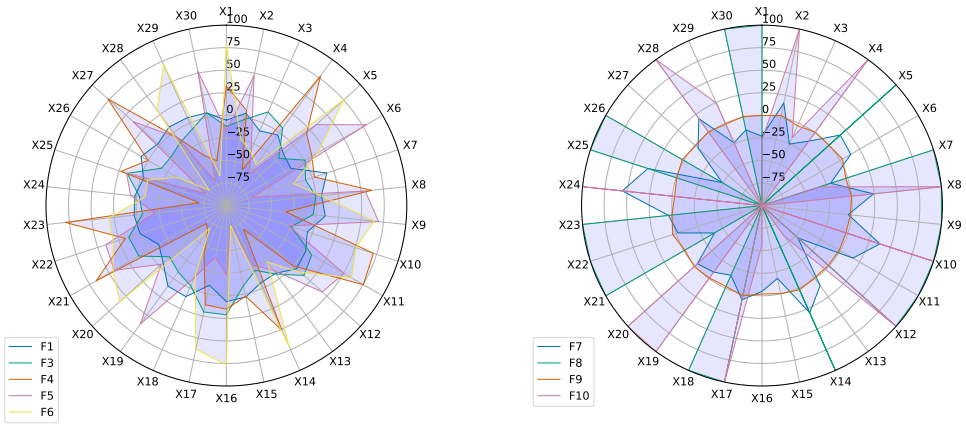


Figure 5.7: Radar chart comparing the variation of the best obtained solutions by NAS on F1, F3-F10 for 30 dimensional problems. Each axis represents a quantity for a different dimension. This chart shows how the designed neural architectures for different problems found the solutions that are rather different or disparate from each other in the search space.

scratch. In the following experiment, batch size will default to 1. To have a fair comparison, five different scenarios are considered in this study. In NAS-1, we freeze the weights of the trained layers of the network so as to use previously learned weights that are hidden throughout all the layers. Thereafter, we change and re-train the last fully-connected Dense layer according to the new dimension. For NAS-2, however, all the weights are re-initialized randomly and architecture should learn everything from scratch. In jSO-2, the solution vectors are initialized with the best found solution for the 30 dimensions, while the algorithm is still able to change the initialized parameters. Alternatively, we freeze these parameters in jSO-3, which reduces the dimensionality of the problem and helps the algorithm to have a better search efficiency [8]. We would like to explore the possibility of speeding up the search process so as to avoid the computational overhead of re-optimizing for large scale computationally expensive problems. To answer this, we repeat the experiments with a cutoff of 1000 evaluations over 15 different runs. In Table 5.6, we can see that the proposed NAS-1 significantly outperformed the randomly initialized architectures NAS-2 and also all the extensions of the jSO; by switching the roles of the classical evolutionary algorithms and learning distributions of the good solutions. The results for NAS-1 and NAS-2 suggest that not only the architecture, but also the learned weights of the network are crucial to the success of accelerating the optimization process. Notably, NAS-1 yielded more scalable performance for the 100 dimensional problems. Overall, we can say the neural architectures addressed the practical limitation of learning from previous similar problems with different dimensions; compared to evolutionary and surrogate-assisted algorithms. Under this setting, we will be able to transfer the parameters of a neural network from a cheap-to-evaluate problem to another similar one in order to accelerate the search process. We will show that

8. Note that the jSO has a population reduction schema and it is not possible to transfer the final population to a new optimization task. More importantly, we notice that the population converges to the best found solution for all the CEC problems.

these results can similarly generalized to the protein problems.

Table 5.6: Performance of NAS and jSO on 50-dimension CEC 2017 test set. The results are obtained by exploiting performance of NAS and jSO on 30-dimension benchmarks.

No	NAS-1	NAS-2	jSO	jSO-2	jSO-3
F1	$2.96902e+03$	$2.41835e+11$	$1.62804e+11$	$4.66952e+10$	$1.734390e+10$
F3	$1.81075e+05$	$2.31981e+05$	$2.35311e+05$	$8.10353e+04$	$7.270594e+04$
F4	$1.48287e+02$	$4.51499e+02$	$4.74709e+04$	$6.82198e+03$	$2.358366e+03$
F5	$2.90176e+02$	$1.95285e+03$	$9.00615e+02$	$5.13882e+02$	$4.671343e+02$
F6	$6.30965e+01$	$1.40318e+02$	$1.25437e+02$	$1.32398e+01$	$8.421186e+00$
F7	$8.26953e+02$	$6.28353e+03$	$3.49137e+03$	$9.54540e+02$	$8.665667e+02$
F8	$4.08987e+02$	$1.21316e+03$	$8.77639e+02$	$5.41581e+02$	$3.926943e+02$
F9	$1.14396e+04$	$6.78905e+04$	$6.81788e+04$	$1.34382e+04$	$4.873473e+03$
F10	$6.74823e+03$	$1.43720e+04$	$1.43100e+04$	$1.46106e+04$	$1.470171e+04$
$+\backslash=\backslash-$		$9\backslash0\backslash0$	$9\backslash0\backslash0$	$7\backslash2\backslash0$	$6\backslash3\backslash0$

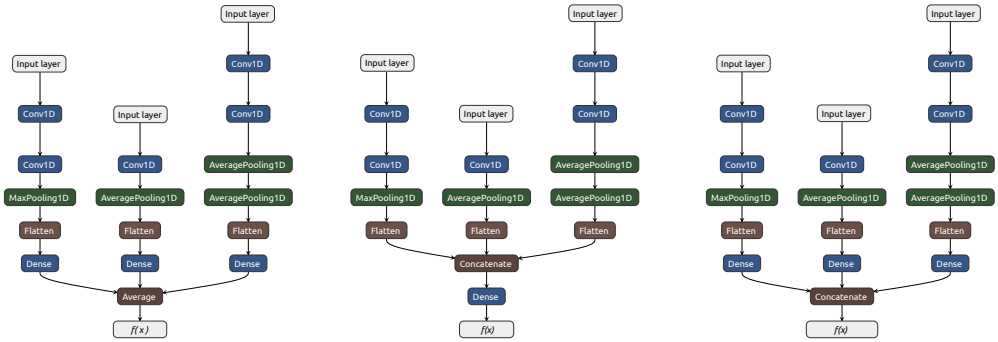
Table 5.7: Performance of NAS and jSO on 100dimension CEC 2017 test set. The results are obtained by exploiting performance of NAS and jSO on 30-dimension benchmarks.

No	NAS-1	NAS-2	jSO	jSO-2	jSO-3
F1	$1.28832e+04$	$5.47524e+11$	$4.99146e+11$	$2.62497e+11$	$2.85013e+11$
F3	$3.49607e+05$	$5.29566e+05$	$7.70880e+05$	$5.80964e+05$	$5.45895e+05$
F4	$1.71629e+02$	$5.51000e+02$	$1.77581e+05$	$8.21492e+04$	$5.60870e+04$
F5	$6.55163e+02$	$4.59244e+03$	$2.27627e+03$	$1.72975e+03$	$1.69408e+03$
F6	$6.10705e+01$	$1.60571e+02$	$1.52327e+02$	$6.19906e+01$	$6.60414e+01$
F7	$1.36100e+03$	$1.52395e+04$	$1.10122e+04$	$7.34962e+03$	$7.27311e+03$
F8	$1.07242e+03$	$2.80098e+03$	$2.24957e+03$	$1.88354e+03$	$1.88620e+03$
F9	$2.23360e+04$	$1.34184e+05$	$1.79658e+05$	$1.13661e+05$	$9.97511e+04$
F10	$1.46885e+04$	$2.76060e+04$	$3.23123e+04$	$3.25155e+04$	$3.16535e+04$
$+\backslash=\backslash-$		$9\backslash0\backslash0$	$9\backslash0\backslash0$	$9\backslash0\backslash0$	$9\backslash0\backslash0$

In Figure 5.8, three different ensemble learning strategies are illustrated [9]. Accordingly, we conduct experiments to show the performance of multiple models instead of a single one for obtaining the best possible results to solve the 50 and 100 problems under a limited budget. In machine learning, this is called ensemble learning and tends to yield better performance by averaging the results over multiple models. In Figure 5.8a, the main principle is to fit a set of weak learners in parallel and to combine them following deterministic averaging. The term bagging is used to describe a family of such ensemble methods [48]. Figure 5.8b illustrates ensemble stacking model where the extracted information from the models are combined. This blending process can be defined using one or more additional layers. In this study, we only going to use one Dense layer for simplicity. Figure 5.8c shows a new hybrid ensemble method that we proposed specifically for optimization tasks. In this strategy, we build and train an ensemble model based on the average value of the outputs (i.e., objective function values) in homogeneous learners.

The already elaborated methods are used to investigate whether we can improve the results in Tables 5.6 and 5.7. The main hypothesis is to combine the weak learners so

9. Netron Visualizer is used to illustrate the model. The tools is available online at: `https://github.com/lutzroeder/netron`



(a) Bagging (b) Stacking (c) Hybrid

Figure 5.8: Easy visualization of different ensemble learning methods using weak learners. In this particular case, the results of three models are aggregated to yield a better performance. The bagged ensemble model is constructed by averaging the final results from each leaner, while the idea of stacking model is to mix different weak learners by training a meta-model. Besides these two, hybrid model is a new approach we introduced for optimization tasks.

as to improve the results returned by the base models. The ensemble learning is applied to the obtained models for 30 dimensional problems for solving 50 and 100 dimensions. Similar to the previous experiment, batch size will default to 1 for all the weak learners. We applied the Bagging, Stacking, and Hybrid schemes on the pre-trained architectures for 30 dimensional problems over 15 runs; rather than using and re-training only the best model. The weights of the trained layers for all the models are frozen. Moreover, we change and re-train the last fully-connected Dense layer according to the new dimension. For the Bagging and Stacking methods, we repeat the experiments with a cutoff of 1000 evaluations over 15 different runs, while the hybrid model needs 15×1000 evaluations.

From Tables 5.6 and 5.8 we can see that the Bagging method outperforms the NAS-1 on functions F1, F4, and F6-F8. Furthermore, the Stacking model gives better results on F4, F7-F8, and F10. Needless to say, the results illustrate the significant improvement of the hybrid model.

In Tables 5.7 and 5.9, the similar results for 100 dimensions are recorded which offer the flexibility of the ensemble methods in proportion to the new higher scale. Overall, the obtained results demonstrate the usefulness of the three ensemble methods in the context of the optimization.

Figure 5.9 gives an intuitive understanding of how the number of models for the Bagging and Stacking methods is related to the accuracy of the generated ensemble model. We can see that there is an improvement in the accuracy when the number of employed models increases. However, sometimes the performance drops to a lower value compared to the one with fewer models. This is a very important factor to be kept in mind that there is no evidence of having more models necessary means a higher ensemble performance.

Table 5.8: Performance of NAS on 50-dimension CEC 2017 test set using ensemble learning. The results are obtained by aggregating the trained models over 15 runs on 30-dimension benchmarks.

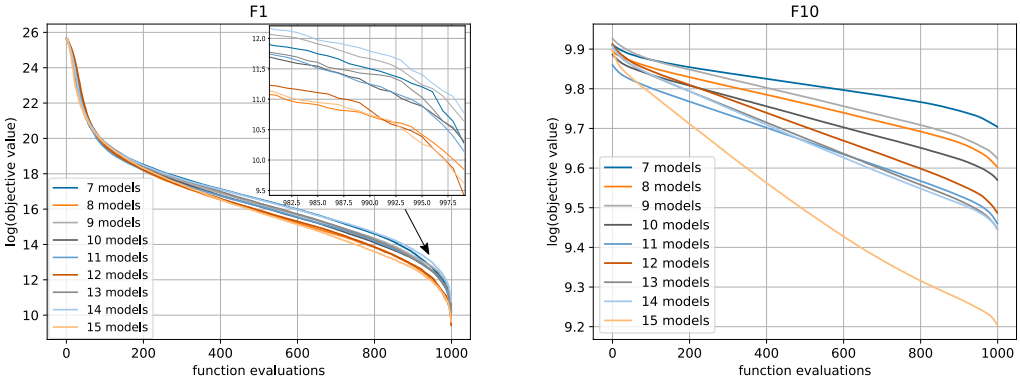
No	Bagging	Stacking	Hybrid
F1	$1.63821e+03$	$4.78148e+03$	$4.37989e+02$
F3	$1.11977e+06$	$1.42817e+06$	$1.66900e+05$
F4	$7.74409e+01$	$1.09783e+02$	$3.35049e+01$
F5	$3.14418e+02$	$5.67149e+02$	$1.01177e+02$
F6	$6.19302e+01$	$1.26289e+02$	$4.60574e+01$
F7	$2.97057e+02$	$4.99652e+02$	$2.63861e+02$
F8	$1.86261e+02$	$2.37593e+02$	$1.52186e+02$
F9	$1.22938e+04$	$1.31054e+04$	$1.06279e+04$
F10	$7.02342e+03$	$5.72794e+03$	$4.99745e+03$

Table 5.9: Performance of NAS on 100-dimension CEC 2017 test set using ensemble learning. The results are obtained by aggregating the trained models over 15 runs on 30-dimension benchmarks.

No	Bagging	Stacking	Hybrid
F1	$5.59750e+03$	$6.52806e+03$	$1.52856e+02$
F3	$2.76794e+10$	$1.09759e+11$	$3.46635e+05$
F4	$1.71032e+02$	$1.89967e+02$	$1.69293e+02$
F5	$3.00000e+02$	$9.05549e+02$	$2.05242e+02$
F6	$6.04775e+01$	$1.25322e+02$	$4.90530e+01$
F7	$2.30870e+03$	$1.21078e+03$	$6.59674e+02$
F8	$9.06463e+02$	$3.11334e+02$	$4.35899e+02$
F9	$2.08151e+04$	$4.37464e+04$	$1.82395e+04$
F10	$1.51215e+04$	$1.27997e+04$	$1.19021e+04$

5.5.5 Results on Protein Structure Prediction

In Table 5.10, we shift our focus to investigate the performance of NAS for PSP. Interestingly, NAS achieves superior results compared to state-of-the-art algorithms designed by human experts. These results show the versatility and robustness of the search-generated neural architectures. Overall, Table 5.10 suggests that NAS strikes better trade-off in every metrics.



(a) (b)

Figure 5.9: This figure illustrates and compares performance of the (a) Bagging and (b) Stacking ensemble methods using different number of models. In most cases, we can say that the larger is the number of models, the more enhanced are the results.

Table 5.10: Performance comparison of NAS and state-of-the-art algorithms for real-world protein sequences. The best, worst, mean, and standard deviation (Std) are reported over 30 runs.

Protein	Algorithm	Best	Worst	Mean	Std
1BXP	IFABC	$-2.00830e+00$	$-1.53750e+00$	$-1.73486e+00$	$9.16579e-02$
	LSHADE	$-2.37310e+00$	$-1.71710e+00$	$-2.10369e+00$	$2.61914e-01$
	SGDE	$-2.30300e+00$	$-1.26950e+00$	$-2.18265e+00$	$2.59131e-01$
	NAS	$-2.49023e+00$	$-2.07551e+00$	$-2.35068e+00$	$1.44632e-01$
1CB3	IFABC	$-3.01540e+00$	$-2.41200e+00$	$-2.65486e+00$	$1.42985e-01$
	LSHADE	$-4.17940e+00$	$-3.88990e-01$	$-3.10716e+00$	$8.44159e-01$
	SGDE	$-5.09030e-01$	$-3.88990e-01$	$-4.77019e-01$	$5.30837e-02$
	NAS	$-4.19828e+00$	$-3.40068e+00$	$-3.85454e+00$	$2.80526e-01$
1BXL	IFABC	$-6.81900e+00$	$-5.27940e+00$	$-6.07831e+00$	$2.97476e-01$
	LSHADE	$-8.34140e+00$	$-6.01370e+00$	$-7.8585e+00$	$6.67609e-01$
	SGDE	$-8.11630e+00$	$-5.93430e+00$	$-6.65916e+00$	$5.86852e-01$
	NAS	$-8.61305e+00$	$-7.18074e+00$	$-8.19518e+00$	$4.44214e-01$
1EDP	IFABC	$-4.77300e+00$	$-3.36490e+00$	$-3.78407e+00$	$2.70870e-01$
	LSHADE	$-6.95040e+00$	$-2.99190e+00$	$-4.78441e+00$	$1.24773e+00$
	SGDE	$-6.23270e+00$	$-4.53410e-01$	$-1.28434e+00$	$1.21523e+00$
	NAS	$-6.95038e+00$	$-1.23753e+00$	$-5.37093e+00$	$1.52153e+00$
2ZNF	IFABC	$-5.82240e+00$	$-4.49080e+00$	$-5.06992e+00$	$3.44016e-01$
	LSHADE	$-7.08230e+00$	$-4.15840e+00$	$-5.37076e+00$	$6.01159e-01$
	SGDE	$-7.13800e+00$	$-4.65980e+00$	$-5.42427e+00$	$7.59187e-01$
	NAS	$-7.41105e+00$	$-3.45382e+00$	$-5.93711e+00$	$9.51282e-01$
1EDN	IFABC	$-7.26590e+00$	$-4.35930e+00$	$-5.06827e+00$	$5.00148e-01$
	LSHADE	$-8.81030e+00$	$-2.47920e+00$	$-4.80588e+00$	$1.71807e+00$
	SGDE	$-8.81030e+00$	$-3.54610e+00$	$-5.84384e+00$	$1.26685e+00$
	NAS	$-8.14153e+00$	$-3.76380e+00$	$-6.04727e+00$	$7.84394e-01$
1DSQ	IFABC	$-5.98690e+00$	$-4.22690e+00$	$-4.78016e+00$	$4.24072e-01$
	LSHADE	$-7.43270e+00$	$-4.39070e+00$	$-5.41422e+00$	$1.02956e+00$
	SGDE	$-7.43270e+00$	$-3.60300e+00$	$-6.18808e+00$	$1.32262e+00$
	NAS	$-7.43270e+00$	$-5.31426e+00$	$-6.85287e+00$	$8.35519e-01$
1SP7	IFABC	$-1.67640e+01$	$-1.42670e+01$	$-1.51513e+01$	$6.53529e-01$
	LSHADE	$-2.17800e+01$	$-1.61490e+01$	$-1.91514e+01$	$1.17516e+00$
	SGDE	$-2.17790e+01$	$-1.64290e+01$	$-1.92621e+01$	$1.70078e+00$
	NAS	$-2.17526e+01$	$-1.69024e+01$	$-1.98864e+01$	$1.40123e+00$
2H3S	IFABC	$-6.04400e+00$	$-4.64080e+00$	$-5.23771e+00$	$3.16578e-01$
	LSHADE	$-6.31350e+00$	$-4.15670e+00$	$-4.45938e+00$	$6.03639e-01$
	SGDE	$-6.42170e+00$	$-4.15670e+00$	$-4.45630e+00$	$6.36553e-01$
	NAS	$-7.56085e+00$	$-3.42569e+00$	$-4.96418e+00$	$1.08122e+00$
1FYG	IFABC	$-1.01220e+01$	$-8.94270e+00$	$-9.44181e+00$	$2.62877e-01$
	LSHADE	$-1.29300e+01$	$-7.46260e+00$	$-1.00965e+01$	$1.71148e+00$
	SGDE	$-1.38600e+01$	$-1.01240e+01$	$-1.19298e+01$	$1.08118e+00$
	NAS	$-1.40123e+01$	$-7.08564e+00$	$-1.19396e+01$	$1.51319e+00$

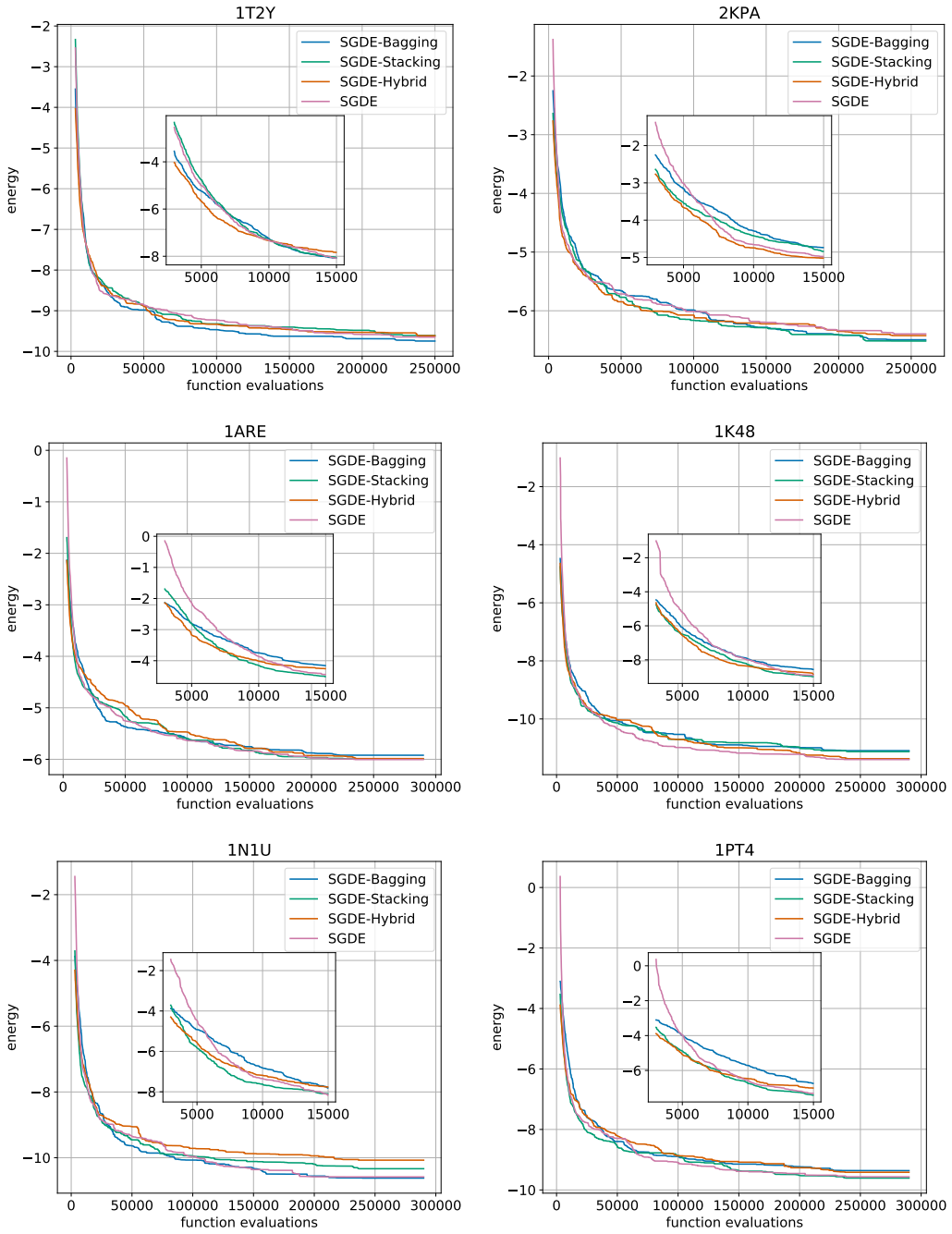


Figure 5.10: This figure illustrates the convergence results of SGDE obtained for 1T2Y, 2KPA, 1ARE, 1K48, 1N1U, and 1PT4 protein sequences with and without warm-start population initialization. We can see that the transferred information using pre-trained ensemble models enhanced the convergence performance of SGDE.

In Figure 5.10, we demonstrate the usefulness of NAS for evolutionary algorithms. Accordingly, the best found architectures for solving the protein sequences $1-10$ are used to accelerate the convergence rate of SGDE on unseen protein sequences $11-16$. To do so, the ensemble model of the best architectures is first adopted to the dimensionality of the

new problem; as we described before in the previous subsection. Then, we re-trained the ensemble model to generate the initial solutions for the SGDE population. The limited budget with a cutoff of 500, 500, and 5000 function evaluations are used for Bagging, Stacking and Hybrid ensemble models, respectively. In SGDE, the experiments are conducted with limited budget $10,000 \times D$ function evaluations over 30 runs. From Figure 5.10, one can see how this transfer learning strategy can replace in part the computationally expensive evaluations by generalizing well from just learning on smaller protein sequences.

5.5.6 When Neural Architecture Search Does Not Help

The authors would like to discuss two major takeaways. First, the application of the existing NAS methods without considering the early stopping strategy decreases the accuracy of the reported results. Relatedly, it is difficult to quantify the performance gains without fine-tuning against leading bath size hyperparameter. In a sense, it is actually an abstract away of optimizing the behavior of the Adam. Second, unlike related approaches, NAS needs more computational time when the computational resources are limited.

5.5.7 Implementation Notes

The obtained results for CEC 2017 and PSP show that NAS is comparable in performance to the traditional evolutionary algorithms which call into question the necessity of incorporating the convolutional operations in hyper-heuristic methods. The comprehensive experimental evaluation of transfer learning and ensemble learning allows us to pinpoint the performance gains associated with the NAS evaluation scheme. We conjecture that even better results could be attained if NAS optimizes multiple PSP sequences to learn the joint distribution of all the instances, and then transfers the learned weights to a new protein sequence.

5.6 Practical End Use of Algorithm

We conclude that neural architecture search can provide an advantage over standard evolutionary algorithms when transferring is enabled. Accordingly, the trained networks can be employed to deliver useful information on unseen optimization problem instances without further training. This makes them suitable alternative for solving low dimensional instances of a problem and using the learned distribution for high dimensional and computational expensive instances. Furthermore, we think that the proposed search strategy can be adopted for multi-objective and binary search spaces.

5.7 Chapter Summary

In this chapter, we revisit the common application of the neural architecture search and reformulates it for optimization tasks. The introduced hyper-heuristic perspective facilitates the process of generating efficient solvers by leveraging a mixture of convolution

components from convolutional neural network. We conducted some experiments regarding to two aspects: (1) using neural architecture search for tackling standard CEC 2017 functions and PSP instances, (2) exploring search acceleration possibilities by means of transfer learning and ensemble learning techniques. Empirical results suggest the superiority of neural architecture search for both problems regarding solution accuracy metrics.

Part II

Metaheuristics for Machine Learning

Chapter 6

Introduction

> Science is not only a disciple of reason but also one of romance and passion.
>
> Stephen Hawking

There is no doubt that CNNs have gained immense popularity for processing visual patterns. The adoption of these techniques to unseen data, however, is severely hampered by choosing a set of optimal hyperparameters associated with them [65, 118]. The learning rate in stochastic gradient descent or number of layers in CNNs are examples of these parameters.

Automated hyperparameter tuning (AHT) addresses the resulting pitfalls by applying optimization search techniques. In recent years, different search strategies [12, 15, 49, 56, 67, 70, 106, 107, 168] have emerged to enhance the performance of CNNs. Although they may come in contact or overlap in some aspects, they can be mainly categorized into random and informed methods. The random search methods are embarrassingly parallel and could be used for low dimensional problems. They do not take into account the useful information about the fitness landscape of the problem at hand which may lead to slow convergence rate. In this context, the advantages of informed search algorithms become clear which build and train computationally cheap-to-evaluate learning models in order to perform an effective search during the optimization process. By leveraging learning models, the computational cost can be greatly reduced since the time overhead of building and training such learning models is insignificant compared to evaluating the CNN models. These methods are not easily parallelizable which is a major challenge to their successful application. Furthermore, they are well suited for single-objective cases. Until very recently, they are mainly designed for optimizing the accuracy of the predictions which make their applications difficult in low-power mobile and embedded areas [1, 3, 164] where there are two conflicting objectives: model complexity and model accuracy. This scenario can also be seen in online visual tracking [49] where some approaches focus on accuracy while several others aim at faster computational speeds. There are some works [40] that adopts a scalarized objective function, but it has been shown that Pareto-based optimization performs better than scalarized methods by providing more generalization ability, and using a scalarized objective function cannot remedy this issue [77].

In this part, we tried to solve the hyperparameter optimization problem using the standard metaheuristics with the hope of exploring more promising leads for CNN models on both single-objective and multi-objective scenarios like FPGAs [1]. Automated tuning process of CNNs is synonymous with computationally expensive simulations, which poses a challenge to the algorithm development and optimization teams. To the best of our knowl-

edge, there is no work that introduce the reader to the efficiency of different state-of-the-art metaheuristics for the single-objective and multi-objective cases.

Chapter 7

Application V: Automated Machine Learning

> Science and everyday life cannot and should not be separated.
>
> Rosalind Franklin

In this chapter, hyperparameter selection and neural architecture search of convolutional neural networks (CNNs) is viewed as single-objective and multi-objective optimization problems. We propose to compare the performance of the conventional metaheuristics with state-of-the-art automated machine learning methods in the literature. The algorithms are testified using HPOBench for joint hyperparameter and architecture optimization of feed forward neural networks on regression problem; and also NASBench101 for the architecture optimization of CNNs. Experimental results demonstrate that metaheuristics can closely optimize the characteristics of the considered problems compared to the recent propositions in the literature; even within a limited computational budget.

7.1 Motivation

The hyperparameter tuning domain is dominated by algorithms that are believed to be more time consuming compared to the model-free methods and accordingly we used metaheuristics to cast some light on how such methods may reduce computation time. The main reason is inherently-sequential features of the model-based algorithms which prevent them from being efficiently parallelized. More worrying is the prospect that this may get worse when tuning methods are exposed to compute several objectives. The parallel random search methods like Hyperband [108] are not able to exploit the search space as efficient as possible. However, incorporating the common-sense knowledge from previous iterations can substantially speed up the search process. There are several works that tried to introduce the Bayesian optimization into the Hyperband [15, 56, 168], but none of them is compared with metaheuristics. In this regard, we also embrace NAS which achieved great success in different computer vision tasks such as object detection and image recognition. The NAS methods are developed in order to automatically search for architecture and hyperparameters of a machine learning model. They already obtain superior results over manually designed architectures on some tasks such as image classification.

7.2 Problem Definition

In the context of automated machine learning, the main focus is on building a high quality pipeline. This is due to the fact that there is no universal approach and new machine learning pipelines have to be constructed for each new data set [186]. Here, a pipeline is a linear sequence of CNN's components that transforms an input vector $\mathbf{x} \in \mathbb{X}$ into a target value $\mathbf{y} \in \mathbb{Y}$. This task can be based on the structure of the pipeline (i.e., neural architecture search) and the choice of the learning algorithms and their hyperparameters (i.e., hyperparameter search).

Definition 1 (Pipeline Creation Problem): *Given a set of CNN layers $\mathcal{A}$ and their associated hyperparameters Λ, a training set $\mathcal{D}_{train}$ and a validation set $\mathcal{D}_{valid}$ such that $\mathcal{D}_{train} \cap \mathcal{D}_{valid} = \varnothing$, the pipeline creation problem (PCP) can be defined as a joint algorithm and hyperparameter selection minimization problem using a loss function $\mathcal{L}$ [186]:*

$$\theta^* = \arg \max_{\theta \in \Theta} u(\theta), \text{where } u(\theta) = f(\theta | I, P_I, P_\zeta, t) \tag{7.1}$$

Here, g is a directed acyclic graph (DAG) that denotes the structure of the pipeline $\mathcal{P}_{g,A,\lambda}$. In g, the nodes (consisting of the selected algorithm A and its associated hyperparameters λ) represent an arbitrary machine learning process and edges represent the flow of an input. The performance of the configuration (g, A, λ) should be evaluated using the validation set $\mathcal{D}_{valid}$.

In hyperparameter optimization, the pipeline structure g is supposed to have a fixed shape which eliminates the complexity of creating a DAG graph. In neural architecture search, however, the hyperparameters are fixed and optimization techniques are used to automatically build a superior model by only finding the best possible architecture.

7.2.1 Methodology

It would be unfair to compare a model with the best hyperparameters/architectures against another one which has not been optimized. Automated machine learning has been emerged to avoid this pitfall. In this subsection, we will walk through the state-of-the-art methods in the literature and also not-yet applied algorithms. In the following, both the single-objective and multi-objective approaches are introduced.

7.2.2 Single-objective Competitive Algorithms

ABC: Artificial bee colony (ABC) [83] mimics the foraging behavior of honey bees for solving optimization problems. In ABC, a colony of artificial forager bees (agents) search for rich artificial food sources (good solutions for a given problem). The algorithm randomly discovers a population of initial solution vectors and then iteratively improve them by employing the strategies: moving towards better solutions by means of a neighbour search mechanism while abandoning poor solutions. The algorithm is shown to perform well on a wide range of complicated real-world problems has the advantage of having less parameters. However, its application to automated machine learning domain has been neglected which

motivated us to consider this algorithm in this chapter.

BOHAMIANN: Bayesian optimization with Hamiltonian Monte Carlo artificial neural networks (BOHAMIANN) [155] is another method which proposed to use neural networks as a powerful and scalable parametric model, while staying as close to a truly Bayesian treatment as possible. Crucially, it aims to keep the well-calibrated uncertainty estimates of GPs since Bayesian optimization relies on them to accurately determine promising hyperparameters. To this end, BOHAMIANN derives a more robust variant of the recent stochastic gradient Hamiltonian Monte Carlo method [32].

BOHB:The bandit-based configuration evaluation approaches based on random search (like HB) lack guidance and do not converge to the best configurations as quickly. So, BOHB [57] proposed to combine the benefits of both Bayesian optimization and bandit-based methods, in order to achieve the best of both worlds: strong anytime performance and fast convergence to optimal configurations. BOHB introduces a new practical state-of-the-art hyperparameter optimization method, which consistently outperforms both Bayesian optimization and HB on a wide range of problem types, including high-dimensional toy functions, support vector machines, feed-forward neural networks,Bayesian neural networks, deep reinforcement learning, and convolutional neural networks.

CMAES: Covariance matrix adaptation evolution strategy (CMAES) [63] is a derivative-free evolutionary algorithm for non-linear and non-convex optimization problems. It generates new candidate solutions according to a multivariate normal distribution. Recombination amounts to selecting a new mean value for the distribution. Mutation amounts to adding a random vector, a perturbation with zero mean. Pairwise dependencies between the variables in the distribution are represented by a covariance matrix. The covariance matrix adaptation is a method to update the covariance matrix of this distribution. The algorithm is shown to perform well with very limited computational budget which makes it a highly competitive method for automated machine learning domain.

DE: The differential evolution (DE) came up with the idea of using vector differences for perturbing the vector population. DE and its extensions are among highly successful meta-heuristics algorithms. A full description of the algorithm can be found in Subsection 2.4.2.

DF: Dragonfly (DF) [81] is an open source Python library for scalable and robust hyperparameter optimization. It incorporates multiple recently developed methods that allow hyperparameter optimization to be applied in challenging real-world settings; these include better methods for handling higher dimensional domains, methods for handling multi-fidelity evaluations when cheap approximations of an expensive function are available, methods for optimizing over structured combinatorial spaces, such as the space of neural network architectures, and methods for handling parallel evaluations. Additionally, the DF develops new methodological improvements in Bayesian optimization for selecting the Bayesian model, selecting the acquisition function, and optimizing over complex domains with different variable types and additional constraints. DF is compared against a suite of other packages and algorithms for global optimization and demonstrate that when the above methods are integrated, they enable significant performance improvements.

DNGO: DNGO [153] explores the use of neural networks as an alternative to Gaussian processes (GPs) to model distributions over functions. The original study shows that per-

forming adaptive basis function regression with a neural network as the parametric form performs competitively with state-of-the-art GP-based approaches, but scales linearly with the number of data rather than cubically. This allows us to achieve a previously intractable degree of parallelism, which we apply to large scale hyperparameter optimization, rapidly finding competitive models on benchmark object recognition tasks using convolutional networks, and image caption generation using neural language models.

HB: Hyperband (HB) [108] is a novel configuration evaluation approach which formulates hyperparameter optimization as a pure-exploration adaptive resource allocation problem addressing how to allocate resources among randomly sampled hyperparameter configurations. The HB's procedure relies on a principled early-stopping strategy to allocate resources, allowing it to evaluate orders-of-magnitude more configurations than black-box procedures like surrogate optimization methods. It is a general-purpose technique that makes minimal assumptions unlike model based configuration evaluation approaches. The theoretical analysis demonstrates the ability of HB adapt to unknown convergence rates and to the behavior of validation losses as a function of the hyperparameters. The theoretical contribution of HB is the introduction of the pure-exploration, infinite-armed bandit problem in the non-stochastic setting, for which HB is one solution. It has been shown $5\times$ to $30\times$ faster than popular Bayesian optimization algorithms on a variety of deep-learning and kernel-based learning problems [108].

HO: HyperOpt (HO) [13] is a hyperparameter optimization library for serial and parallel optimization over awkward search spaces, which may include real-valued, discrete, and conditional dimensions. In HO, the authors propose a meta-modeling approach to support automated hyperparameter optimization, with the goal of providing practical tools that replace hand-tuning with a reproducible and unbiased optimization process. The HO approach is to expose the underlying expression graph of how a performance metric (e.g. classification accuracy on validation examples) is computed from hyperparameters that govern not only how individual processing steps are applied, but even which processing steps are included. A hyperparameter optimization algorithm transforms this graph into a program for optimizing that performance metric. HO yields state of the art results on three disparate computer vision problems and so is considered in this thesis: a face-matching verification task, a face identification task and an object recognition task , using a single broad class of feed-forward vision architectures.

MAC: In this chapter, we adopt a single-objective version of MAC 3.4 as the Bayesian search strategy for automated machine learning. Accordingly, we generate a set of random architectures in the early iterations in parallel, while we adopt a feature selection strategy to generate more promising candidates in the later stages of development. This is a key property in MAC that helps us to make a balance between the solution quality and the computational time. After half of the iterations, a response surface model is created to provide a fast approximation of the expensive evaluations for the later stages of evolution. Moreover, the randomly generated perturbations are also ranked in descending order according to their contribution to the validation accuracy: the top 50% with *promising* label and the others with *non-promising* label. Given this training set, MAC applies a feature selection strategy to obtain weight for generating the new configurations; as elaborated

in 3.4.

IMAC: IMAC is the more enhanced version of the above mentioned modified MAC algorithm in which the parameters of Radial basis function are tuned using a DE metaheuristic algorithm. IMAC is an example of inter-playing the machine learning and metaheuristic algorithms. We proposed an optimization algorithm by virtue of machine learning model for automated machine learning. Meanwhile, the adopted machine learning inside the optimization process itself is tuned using a metaheuristics algorithm.

PSO: In particle swarm optimization (PSO) [86], a population of particles starts to move in search space by following the current optimum particles and changing the positions in order to find out the optima. In every iteration, each particle is updated by following the best solution of current particle achieved so far and the best of the population. When a particle takes part of the population as its topological neighbours, the best value is a local best. The particles tend to move to good areas in the search space by the information spreading to the swarm. PSO is known to be one the most successful metaheuristics for real-world optimization problems and is a challenging competitive for random search, surrogate-base, and metaheuristic algorithms.

RE: Regularized evolution (RE) [138] genetic algorithm presents a novel variant of tournament selection by which genotypes die according to their age, favoring the young. This improved upon standard tournament selection while still allowing for efficiency at scale through asynchronous population updating. In RE, an age is assigned to the individuals (not the genes) and is only used to track which is the oldest individual in the population. This permits removing such oldest individual at each cycle (keeping a constant population size). Meanwhile, this approach keeps the algorithm as simple as possible. In particular, RE remains similar to nature (where the young are less likely to die than the very old) and it requires no additional meta-parameters. It has been shown that RE has somewhat faster search speed and stood out in the regime of scarcer resources/early stopping; compared to RS, RL.

RL: The reinforcement learning (RL) tuning method [104] approaches the problem using a machine learning perspective and represent any particular hyperparameter optimization procedure as a reinforcement learning problem. Under this assumption, any particular hyperparameter simply corresponds to a policy. The algorithm rewards hyperparameters that converge quickly and penalize those that do not. Learning good hyperparameters then reduces to find an optimal policy, which can be solved using any reinforcement learning method. The algorithm uses an off-the-shelf reinforcement learning algorithm known as guided policy search [99], which has demonstrated success in a variety of robotic control settings.

RS: Random search (RS) is the most simple yet effective [105] baseline in this study. We used our implementation according to which the generated candidates are drawn from a uniform probability distribution and are independent of the samples that come before it. This property makes it well suited to highly parallel systems. Moreover, random methods are flexible in that they can be applied to both the continuous and discrete search space; in contrast to Bayesian approaches based on Gaussian processes [90] and gradient-based approaches [113].

SRS: Stochastic Response Surface(SRS) [139] is a surrogate-based algorithm that utilizes a response surface model to approximate the expensive function and identifies a promising point for function evaluation from a set of randomly generated points, called candidate points. Assuming some mild technical conditions, SRS converges to the global minimum in a probabilistic sense. The best candidate point in SRS is chosen according to two criteria: the estimated function value obtained from the response surface model, and the minimum distance from previously evaluated points. It is worth mentioning that MAC and IMAC propositions are inspired by SRS.

SMAC: SMAC (sequential model-based algorithm configuration) [69] is another versatile tool for hyperparameter optimizing. SMAC has helped us speed up both local search and tree search algorithms by orders of magnitude on certain instance distributions. Recently, it has also been found it to be very effective for the hyperparameter optimization of machine learning algorithms, scaling better to high dimensions and discrete input dimensions. The main contribution of SMAC is to make sequential model-based optimization applicable to general algorithm configuration problems with many categorical parameters. Specifically, it generalizes four components of the sequential model-based optimization framework andbased on themdefine two novel instantiations capable of general algorithm configuration: the simple model-free Random Online Adaptive Racing (ROAR) procedure and the more sophisticated Sequential Model-based Algorithm Configuration (SMAC) method. SMAC can be understood as an extension of ROAR that selects configurations based on a surrogate model rather than uniformly at random. We refer the reader to the original study [69] for more details.

TPE: The Tree-structured Parzen Estimator (TPE) [14] is a sequential model-based optimization algorithm which has been successfully applied to computationally expensive optimization tasks. TPE sequentially builds and trains a surrogate model to approximate the performance of hyperparameters based on historical measurements, and then subsequently choose new hyperparameters to test based on the surrogate model. Mathematically speaking, it models $P(x|y)$ and $P(y)$ where x represents hyperparameters and y the associated quality score. Here, $P(x|y)$ is modeled by transforming the generative process of hyperparameters, replacing the distributions of the configuration prior with non-parametric densities. Using different observations $x^{(1)}, \cdots, x^{(k)}$ in the non-parametric densities, these substitutions represent a learning algorithm that can produce a variety of densities over the configuration space X. The TPE defines $p(x|y)$ using two such densities:

$$\begin{cases} \iota(x) & if y < y^\star \\ g(x) & if y \geq y^\star \end{cases} \tag{7.2}$$

where $\iota(x)$ is the density formed by using the observations $x^(i)$ such that corresponding loss function $f(x^{(i)})$ was less than $y^\star$ and $g(x)$ is the density formed by using the remaining observations. Whereas the GP-based approach favored quite an aggressive $y^\star$ (typically less than the best observed loss), the TPE algorithm depends on a $y^\star$ that is larger than the best observed $f(x)$ so that some points can be used to form $\iota(x)$. The TPE algorithm chooses $y^\star$ to be some quantile γ of the observed y values, so that $p(y < y^\star) = \gamma$, but no

specific model for $p(y)$ is necessary. By maintaining sorted lists of observed variables in H, the runtime of each iteration of the TPE algorithm can scale linearly in $|H|$ and linearly in the number of variables (dimensions) being optimized. We refer the reader to [14] for more details.

7.2.3 Multi-objective Competitive Algorithms

DBEA: DBEA [7] is a decomposition based evolutionary algorithm, wherein, uniformly distributed reference points are generated via systematic sampling, balance between convergence and diversity is maintained using two independent distance measures and a simple preemptive distance comparison scheme is used for association. In order to deal with constraints, an adaptive epsilon formulation is used. The reference directions are generated using systematic sampling, where in the points are systematically generated on a hyperplane with unit intercepts in each objective axis. The association of solutions to reference directions are based on two independent distance measures. The distance along the reference direction controls convergence, whereas the perpendicular distance from the solution to the reference direction controls the diversity. The proposed algorithm utilizes a simple prioritized distance comparison scheme to maintain this balance and control association. In order to improve the efficiency of the algorithm, a steady state form is adopted.

DENSEA: Duplicate elimination non-dominated sorting evolutionary algorithm, called as DENSEA, emphasizes the creation and maintenance of population diversity and is conceived as an improvement medium for the previously explained difficulties of having a non-dominated solution reduced set in functional space with respect to the population size [123]. DENSEA is based on the non-domination sorting criterion selection and has incorporated some elitism, but it is characterized by offering population diversity maintenance based on various characteristics: 1) Deletion of duplicate solutions; 2) Replacement of these duplicate solutions; 3) Replacement selection of population of next generation. The population is ordered by the non-domination criterion, with the distribution operator along the front (secondary criterion when solutions belong to the same front) explained later. After this sorting, each individual has a linear selection probability by Roulette Wheel Selection, which determines the individuals that are selected for crossover and mutation. These constitute the offspring population. This population is also ordered by the non-domination criterion after evaluation. DENSEA has specifically a deletion operator for duplicate solutions: the algorithm deletes the accumulated duplicate solutions due to the reduced non-dominated solutions quantity in the functional space. The replacement of each deleted solution is performed by inserting the individual that has the same ordering in the second half of the population until the completion of 50% of the population size ($N/2$). In that way, the inclusion of diverse solutions replacing duplicates is fostered, helping to maintain the population diversity. This filtering process is implemented both in the parent and offspring population.

ϵ-MOEA: In ϵ-MOEA [46], a steady-state multi-objective evolutionary algorithms (MOEA) based on the ϵ-dominance concept [14] and efficient parent and archive update strategies has been proposed. The goal is to introduce a compromised algorithm for achieving a well-distributed set of solutions quickly. It has been observed that the proposed

steady-state MOEA is a good compromise in terms of convergence near to the Pareto-optimal front, diversity of solutions, and computational time.The ϵ-MOEA is a step closer towards making MOEAs pragmatic, particularly allowing a decision-maker to control the achievable accuracy in the obtained Pareto-optimal solutions.

ϵ-NSGAII: The ϵ-NSGAII [93] incorporates prior competent evolutionary algorithm design concepts and epsilon-dominance archiving to improve the original NSGAII's efficiency, reliability, and ease-of-use. This algorithm eliminates much of the traditional trial-and-error parameterization associated with evolutionary multi-objective optimization through epsilon-dominance archiving, dynamic population sizing, and automatic termination. The effectiveness and reliability of the algorithm is successfully compared against the original NSGAII.

FastPGA: Fast Pareto genetic algorithm (FastPGA) [55] uses a new fitness assignment and ranking strategy for the simultaneous optimization of multiple objectives where each solution evaluation is computationally and/or financially expensive. This is often the case when there are time or resource constraints involved in finding a solution. A population regulation operator is introduced to dynamically adapt the population size as needed up to a user-specified maximum population size. The purpose of this is to propose a multi-objective optimization methodology that finds evenly-distributed Pareto optimal solutions in a computationally-efficient manner.

IBEA: Indicator-based evolutionary algorithm (IBEA) [184] discusses how preference information of the decision maker can in general be integrated into multi-objective search. The main idea is to first define the optimization goal in terms of a binary performance measure (indicator) and then to directly use this measure in the selection process. In contrast to existing algorithms, IBEA can be adapted to the preferences of the user and moreover does not require any additional diversity preservation mechanism such as fitness sharing to be used. It has been shown that IBEA can substantially improve on the results with respect to different performance measures.

MOCell: Multi-objective cellular genetic algorithm (MOCell) [120] is a cellular genetic algorithm for solving multi-objective continuous optimization problems. MOCell is characterized by using an external archive to store non-dominated solutions and a feedback mechanism in which solutions from this archive randomly replace existing individuals in the population after each iteration. MOCell is a superior algorithm concerning the diversity of the solutions along the Pareto front [120].

MOEAD: MOEAD [103] is a recent algorithm which simultaneously optimizes a number of single objective optimization subproblems. The objective in each of these problems is an aggregation of all the objectives. Neighborhood relations among these subproblems are defined based on the distances between their aggregation coefficient vectors. Each subproblem (i.e., scalar aggregation function) is optimized by using information mainly from its neighboring subproblems.

NSGAII: NSGA-II(Non-dominated Sorting Genetic Algorithm II) [45] suggests a non-dominated sorting-based evolutionary algorithm with a fast non-dominated sorting approach with O(MN/sup 2/) computational complexity. Also, a selection operator is presented that creates a mating pool by combining the parent and offspring populations and

selecting the best N solutions (with respect to fitness and spread). It has been shown that NSGA-II is able to find a much better spread of solutions and better convergence near the true Pareto-optimal front.

NSGAIII: NSGAIII [44] is a recent effort for developing a potential algorithm for solving many-objective optimization problems. It suggests a reference-point-based many-objective evolutionary algorithm following NSGA-II framework that emphasizes population members that are non-dominated, yet close to a set of supplied reference points.

PAES: Pareto archived evolution strategy (PAES) [92] introduces a simple non-trivial evolution scheme capable of generating diverse solutions in the Pareto optimal set. The algorithm is identified as being a $(1 + 1)$ evolution strategy, using local search from a population of one but using a reference archive of previously found solutions in order to identify the approximate dominance ranking of the cur-rent and candidate solution vectors.

PESA2: PESA2 [39] describes a new selection technique for evolutionary multi-objective optimization algorithms in which the unit of selection is a hyperbox in objective space. In this technique, instead of assigning a selective fitness to an individual, selective fitness is assigned to the hyperboxes in objective space which are currently occupied by at least one individual in the current approximation to the Pareto frontier. A hyperbox is thereby selected, and the resulting selected individual is randomly chosen from this hyperbox. This method of selection is shown [39] to be more sensitive to ensuring a good spread of development along the Pareto frontier than individual-based selection.

RVEA: RVEA [34] proposes a reference vector-guided evolutionary algorithm for many-objective optimization. The reference vectors can be used not only to decompose the original multi-objective optimization problem into a number of single-objective subproblems, but also to elucidate user preferences to target a preferred subset of the whole Pareto front. In the proposed algorithm, a scalarization approach, termed angle-penalized distance, is adopted to balance convergence and diversity of the solutions in the high-dimensional objective space. An adaptation strategy is also proposed to dynamically adjust the distribution of the reference vectors according to the scales of the objective functions.

SMSEMOA: SMSEMOA [54] is a promising algorithm for Pareto optimization, especially if a small, limited number of solutions is desired and areas with balanced trade-offs shall be emphasized. The selection and variation procedures do not interfere with an extra archive and the number of strategy parameters is very low(population size and reference point). Instead of specifying a reference point the SMS-EMOA can also work with an infinite reference point. SMSEMOA is of special elegance, since its implementation is quite simple and the update of the population can be computed efficiently.

SPEA2: The Strength Pareto Evolutionary Algorithm (SPEA) is a relatively recent technique for finding or approximating the Pareto optimal set for multi-objective optimization problems. SPEA2 [185] is an improved version which incorporates in contrast to its predecessor a fine-grained fitness assignment strategy, a density estimation technique, and an enhanced archive truncation method. The main differences of SPEA2 in comparison to SPEA are: 1) An improved fitness assignment scheme is used, which takes for each individual into account how many individuals it dominates and it is dominated by, 2) A nearest neighbor density estimation technique is incorporated which allows a more precise guidance

of the search process, 3) A new archive truncation methods guarantees the preservation of boundary solutions.

VEGA: A Vector Evaluated Genetic Algorithm (VEGA) [147] uses a fitness function that returns a vector. The VEGA splits the population into sub-populations, and each sub-population optimizes toward a different part of the vector (or different vector). Then there is an additional comparison to produce the best result. What VEGA suggests is a simple customization in the selection step. Since you now have multiple objective functions, you could loop over them and for each objective function, you select the fittest individuals. This way, you end up with the fittest individual in the current generation regarding the various objectives in your problem. By allowing those to reproduce, you are spreading their characteristics in your population.

7.3 Experiments

7.3.1 Benchmark Sets

We facilitate a better empirical evaluation of optimization methods for automated machine learning by using benchmarks that are cheap to evaluate, but still represent realistic use cases. These benchmarks are easy and computational efficient ways to conduct reproducible experiments. The hyperparamter optimization part includes a large grid of configurations of a feed forward neural network on four different regression datasets including architectural hyperparameters and hyperparameters. For neural architecture search part, we aim to do the same by introducing NAS-Bench-101, the first public architecture dataset for architecture search research. The dataset is carefully constructed by considering a compact, yet expressive, search space, exploiting graph isomorphisms to identify 423k unique convolutional architectures. These architectures are trained and evaluated multiple times on CIFAR-10 and compiled the results into a large dataset of over 5 million trained models. Based on these two dataset, we conducted an in-depth experiment to gain a better understanding of the properties of different optimization methods from the literature in terms of performance and robustness.

HPOBench: HPOBench [91] contains a large grid four popular UCI [8] regression datasets: protein structure [136], slice localization [60], naval propulsion [38] and parkinsons telemonitoring [165]. For each dataset, 60% are used for training, 20% for validation and 20% for testing [91]. A two layer feed forward neural network followed by a linear output layer on top has been used. The obtained neural architecture are trained with Adam [88] for 100 epochs. The configuration space includes number of units and activation functions for both layers, dropout rates per layer, batch size, initial learning rate and learning rate schedule. More details are given in Table 7.1.

NAS-Bench-101: The NAS-Bench-101 dataset [176] maps a generated neural architecture to its training and evaluation metrics on the CIFAR-10 classification set. In NAS-Bench-101, the search space of neural architectures is defined by considering the *cells*(please see 5.4.1). Accordingly, the variation in the neural architectures comes directly from variation in the *cells*. In the final architecture, each cell in stacked for 3 times, followed by

Table 7.1: The configuration space of the HPOBench benchmarks

Hyperparameters	Values
Initial LR	{0.0005, 0.001, 0.005, 0.01, 0.05, 0.1}
Batch Size	{8, 16, 32, 64}
LR Schedule	{cosine, fix}
Activation/Layer 1	{relu, tanh}
Activation/Layer 2	{relu, tanh}
Layer 1 Size	{8, 16, 32, 64, 128, 256, 512}
Layer 2 Size	{8, 16, 32, 64, 128, 256, 512}
Dropout/Layer 1	{0.0, 0.3, 0.6}
Dropout/Layer 2	{0.0, 0.3, 0.6}

a down sampling layer. This pattern is also repeated 3 times, followed by global average pooling and a final dense softmax layer [176]. The space of cell architectures consists of all possible directed acyclic graphs on V nodes, where each possible node has one of L labels, representing the corresponding operation. In order to limit the size of the space, NAS-Bench-101 imposes some constraints as follows: 1) $L = 3$ and only 3×3 convolution, 1×1 convolution, 3×3 max-pool; 2) $V \leq 7$ and the maximum number of edges is limited to 9. In all the experiment the following encoding schema is used to represent the solution vector: a 7-vertex directed acyclic graph, represented by a 7×7 upper-triangular binary matrix, and a list of 5 labels, one for each of the 5 intermediate vertices [176].

7.3.2 Experimental Settings

For each algorithm, the initial population is randomly generated. The metaheuristics are criticized in the machine learning domain due to the fact that tuning and adjusting the control parameters of these algorithms is itself an optimization problem. We set the parameters for the metaheuristics to their default values in order to address the aforementioned issue and to have a fair comparison against the parameter free methods. We followed the same procedure for some Bayesian optimization methods. For the single-objective metaheuristics, their default parameters are clearly reported in their original studies. For the multi-objective algorithms, we followed the default parameters which are defined in MOEA [62], PISA [18] and jMetal [50] frameworks. The two objectives are the accuracy, and the complexity of the model during the training process. We used hypervolume (HV) in order to compare the proximity and diversity of the obtained results. The HV indicator should be maximized during the configuration process.

7.3.3 Results and Discussion

Single-objective: The simple, but still step toward comparing different hyperparameter optimization algorithms is considering single-objective algorithms. To do so, we report the test regret for naval propulsion, protein structure, slice localization, and parkinsons telemonitoring datasets in Tables 7.2 − 7.5, respectively. The test regret for $\hat{A}_i$ is computed as $r(\hat{A}_i) = f(\hat{A}_i) - f(\hat{A}^*)$, where A^* is the model with the best mean test accuracy in the entire dataset. For all the algorithms, stopping criteria is 200 evaluations and the experiments are repeated 500 times. The results are sorted according to the mean criterion that makes it easier to read and understand. In addition, these results are also visualized in Figures 7.1 and 7.4.

From Tables 7.2 and 7.3, TPE is more successful than other algorithms in terms of solution quality and standard deviation. Conversely, MAC shows better performance according to the results in Tables 7.4 and 7.5. Meanwhile, the best reported result for RL, BOHB, PSO and DE indicate that better performance is probably available, but the configuration space should be explored more efficiently.

Table 7.2: The reported test regret for single-objective naval propulsion hyperparameter optimization dataset. The results show the performance of 500 independent runs over 200 function evaluations. The results are sorted according to the mean criterion that makes it easier to read and understand.

Algorithm	Best	Worst	Mean	Std.
TPE	0.000e+00	8.221e-05	**1.39e-05**	1.512e-05
MAC	0.000e+00	1.014e-04	1.701e-05	1.857e-05
RE	0.000e+00	1.226e-04	2.067e-05	1.925e-05
BOHAMIANN	9.343e-06	5.990e-05	3.081e-05	1.413e-05
DNGO	2.592e-05	9.142e-05	5.259e-05	2.030e-05
HO	4.715e-05	7.965e-05	5.579e-05	6.380e-06
DF	4.833e-05	9.142e-05	6.300e-05	1.531e-05
DE	1.287e-05	1.394e-04	6.434e-05	1.902e-05
PSO	7.169e-06	2.891e-04	6.710e-05	2.347e-05
ABC	7.169e-06	2.021e-04	7.033e-05	2.317e-05
SMAC	6.694e-05	1.174e-04	8.948e-05	1.754e-05
BOHB	0.000e+00	2.507e-01	5.838e-04	1.120e-02
RS	0.000e+00	3.355e-01	1.761e-03	2.038e-02
HB	0.000e+00	5.013e-01	2.425e-03	2.986e-02
RL	0.000e+00	7.519e-01	2.578e-03	3.709e-02
CMAES	3.874e-06	7.519e-01	9.471e-03	6.184e-02

Table 7.3: The reported test regret for single-objective protein structure hyperparameter optimization dataset. The results show the performance of 500 independent runs over 200 function evaluations. The results are sorted according to the mean criterion that makes it easier to read and understand.

Algorithm	Best	Worst	Mean	Std.
TPE	0.000e+00	3.604e-02	**4.87e-03**	5.931e-03
RE	0.000e+00	4.258e-02	8.138e-03	6.768e-03
MAC	0.000e+00	5.335e-02	1.246e-02	1.185e-02
BOHB	0.000e+00	5.575e-02	1.651e-02	9.680e-03
RL	4.658e-04	6.264e-02	2.062e-02	1.034e-02
RS	0.000e+00	6.211e-02	2.207e-02	1.076e-02
HB	4.658e-04	7.979e-02	2.680e-02	1.347e-02
BOHAMIANN	7.650e-03	6.292e-02	2.723e-02	1.628e-02
DNGO	5.002e-03	5.884e-02	3.562e-02	1.547e-02
PSO	9.836e-03	8.938e-02	6.467e-02	1.409e-02
HO	6.300e-02	8.316e-02	6.744e-02	3.941e-03
DF	6.517e-02	7.469e-02	7.064e-02	4.475e-03
DE	4.461e-02	1.000e-01	7.365e-02	7.962e-03
ABC	4.837e-02	3.408e-01	7.612e-02	1.503e-02
CMAES	1.272e-02	3.438e-01	9.435e-02	4.263e-02
SMAC	2.848e-01	3.087e-01	3.021e-01	8.080e-03

Table 7.4: The reported test regret for single-objective slice localization hyperparameter optimization dataset. The results show the mean performance of 500 independent runs over 200 function evaluations. The results are sorted according to the mean criterion that makes it easier to read and understand.

Algorithm	Best	Worst	Mean	Std.
MAC	0.000e+00	1.180e-04	**2.98e-05**	2.913e-05
TPE	0.000e+00	3.222e-04	4.744e-05	4.027e-05
RE	0.000e+00	1.718e-03	6.212e-05	8.443e-05
HO	6.893e-05	1.141e-04	7.523e-05	1.151e-05
BOHAMIANN	4.183e-05	1.593e-04	7.754e-05	3.382e-05
DNGO	7.403e-06	1.593e-04	7.791e-05	4.174e-05
BOHB	0.000e+00	2.071e-03	1.111e-04	1.137e-04
DF	9.718e-05	2.048e-04	1.387e-04	4.725e-05
DE	4.183e-05	3.948e-04	1.404e-04	6.043e-05
RL	0.000e+00	1.945e-03	1.522e-04	1.163e-04
ABC	5.002e-05	6.050e-04	1.738e-04	8.050e-05
PSO	4.508e-05	6.979e-04	1.748e-04	8.643e-05
RS	0.000e+00	2.138e-03	1.769e-04	1.541e-04
HB	0.000e+00	2.582e-03	2.244e-04	1.860e-04
SMAC	2.594e-04	3.230e-04	2.928e-04	1.964e-05
CMAES	2.044e-05	2.911e-02	2.552e-03	3.257e-03

Table 7.5: The reported test regret for single-objective parkinsons telemonitoring hyperparameter optimization dataset. The results show the mean performance of 500 independent runs over 200 function evaluations. The results are sorted according to the mean criterion that makes it easier to read and understand.

Algorithm	Best	Worst	Mean	Std.
MAC	0.000e+00	1.139e-02	**3.89e-03**	2.265e-03
HO	2.015e-03	1.139e-02	4.332e-03	1.999e-03
DNGO	0.000e+00	1.016e-02	4.480e-03	2.628e-03
BOHAMIANN	2.015e-03	8.969e-03	5.108e-03	2.122e-03
RE	0.000e+00	1.077e-01	5.594e-03	7.125e-03
DE	0.000e+00	1.888e-02	5.730e-03	2.986e-03
DF	2.400e-03	9.745e-03	6.381e-03	3.020e-03
PSO	0.000e+00	2.196e-02	6.870e-03	3.270e-03
TPE	0.000e+00	1.289e-01	6.952e-03	1.420e-02
ABC	9.081e-04	2.464e-02	6.966e-03	3.441e-03
RL	0.000e+00	1.077e-01	7.870e-03	6.204e-03
RS	0.000e+00	3.788e-02	8.755e-03	4.323e-03
BOHB	9.081e-04	1.077e-01	9.504e-03	7.139e-03
SMAC	7.948e-03	1.047e-02	9.851e-03	4.920e-04
HB	9.081e-04	1.289e-01	1.208e-02	9.921e-03
CMAES	1.468e-03	5.036e-01	3.646e-02	4.803e-02

The obtained results show that the hyperparameter optimization task is harder for some datasets. We believe that this is related to the uncertainly around the trained models which is a key point for TPE. In naval propulsion and protein structure datasets, the number of features is small and the model is able to predict the output with a low uncertainly of training process. This helps TPE to be a superior algorithm compared to MAC. On the other hand, MAC performs better on the two other datasets which have a higher number

of feature because it does not take into account the uncertainly of the model. To show this, we present Figure 7.2 which tell us how sensitive an optimizer is to the randomness during the search and the training process.

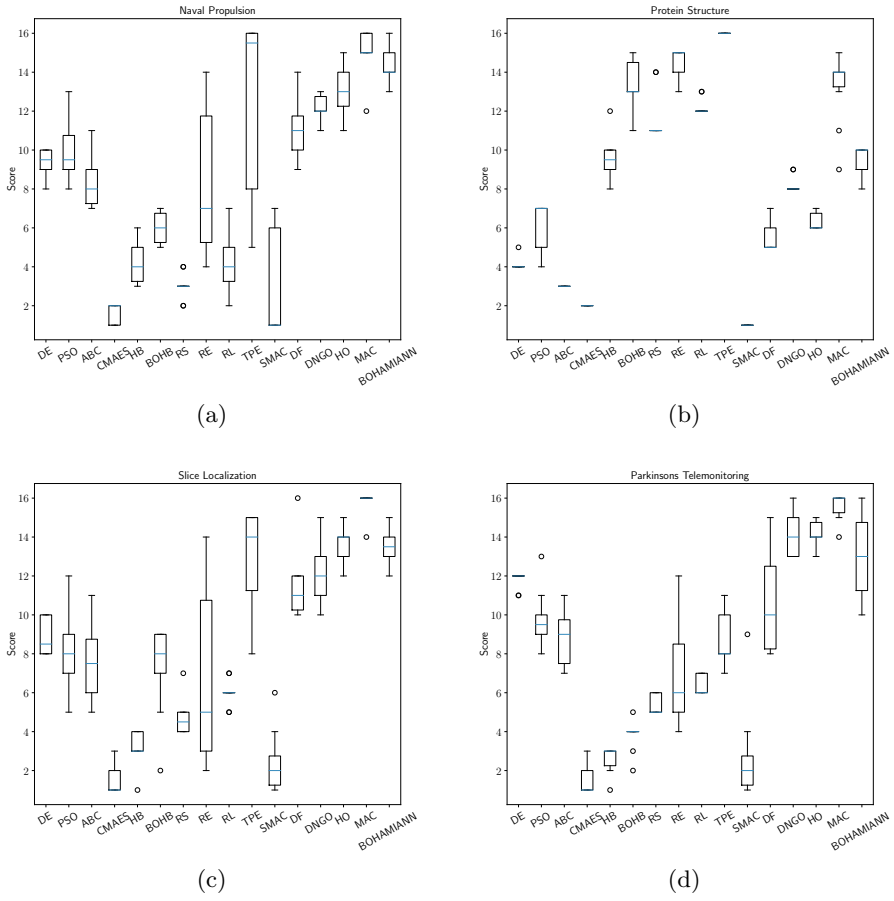


Figure 7.1: The ranking results for the single-objective hyperparameter optimization algorithms. The plot uses the mean performance of 500 independent runs over 200 function evaluations.

In Figure 7.3, the convergence rate of the best three algorithms are reported. This is of practical importance when working with computationally expensive evaluations, as then we can see which of the iterative methods needs fewer iterations. From this figure, we can see that MAC algorithms performs better in early step of the evolution. It seems that MAC can perform well when we have a very small computational budget. For the protein structure dataset, TPE and RE performs better than MAC and have more rapid convergence rate.

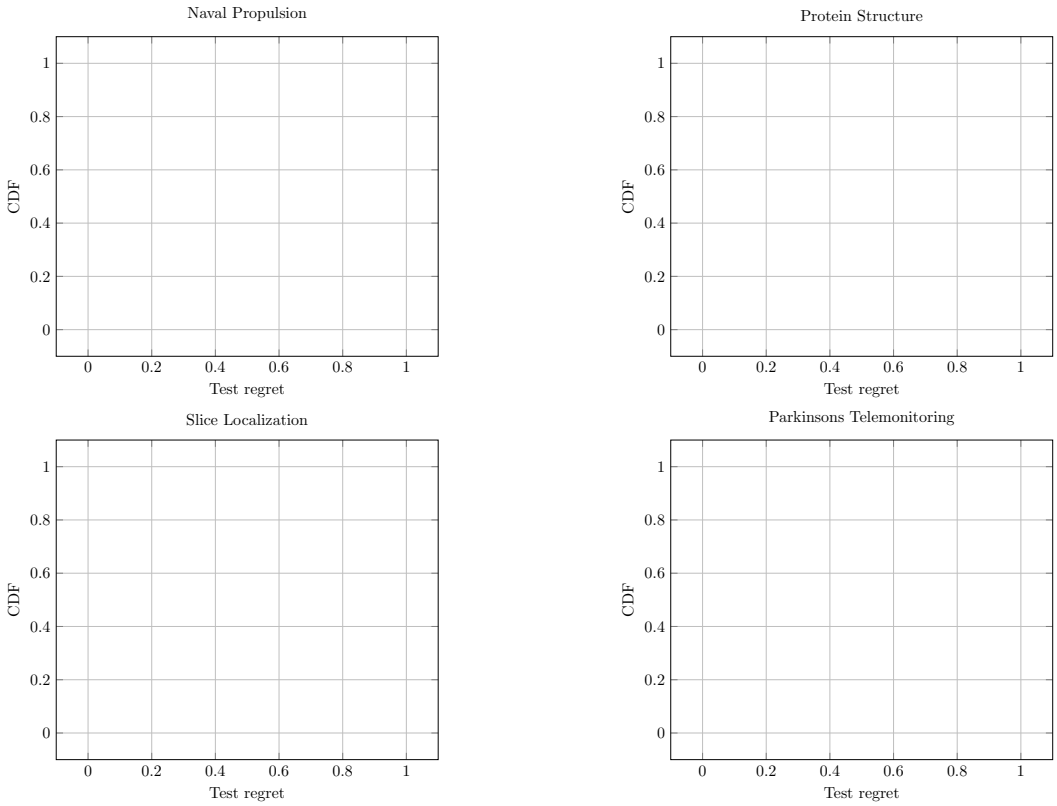


Figure 7.2: The obtained cumulative distribution function of the best algorithms for single-objective hyperparameter optimization. The plot is based on the performance of 500 independent runs over 200 function evaluations.

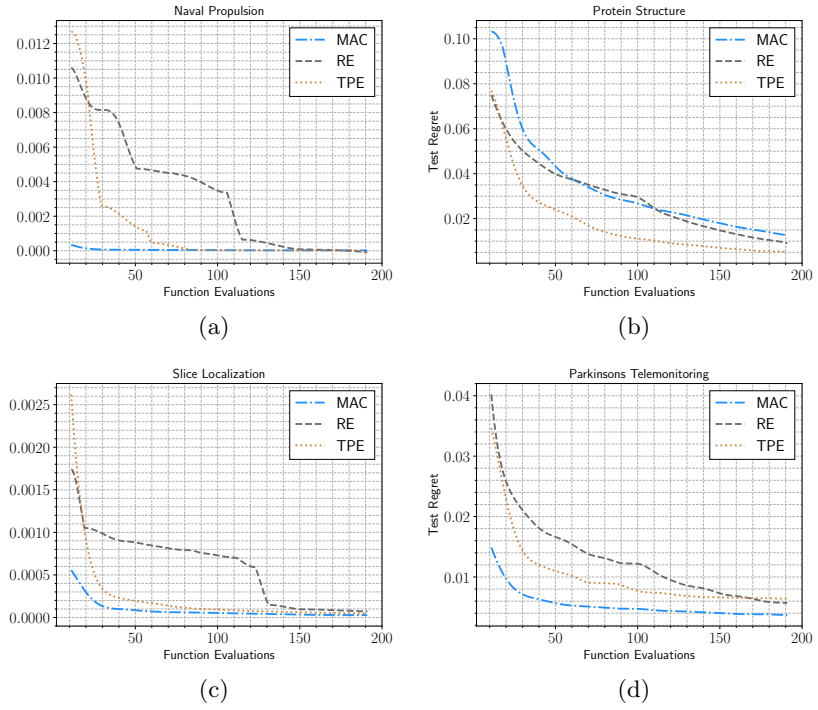


Figure 7.3: The converges rate for the best single-objective hyperparameter optimization algorithms. The plot is based on the mean performance of 500 independent runs over 200 function evaluations on CIFAR-10 dataset.

Similarly, we reported the results for single-objective neural architecture search in Table 7.6 and Figure 7.4. In this case, we have a large search space and so the number of function evaluations are increased to 1000. We should note that the considered algorithms here are the superior models for hyperparameter optimization, while some Bayesian optimization algorithms with very high computational complexity are ignored. These methods rely on querying a distribution over the surrogate model. We know that an accurate model is critical to the effectiveness of the approach and they are mainly using Gaussian processes. The problem arises due to the fact that Gaussian processes scale cubically with the number of observations and this is a challenge to our problem whose optimization requires many evaluations. Meanwhile, metaheuristics like DE and PSO can be easily parallelized and so will need less computation hours.

Table 7.6: The reported test regret for single-objective neural architecture search. The results show the performance of 500 independent runs over 1000 function evaluations on CIFAR-10 dataset. The results are sorted according to the mean criterion that makes it easier to read and understand.

Algorithm	Best	Worst	Mean	Std.
PSO	8.347e-04	7.378e-03	**2.53e-03**	1.866e-03
IMAC	8.347e-04	7.378e-03	3.507e-03	2.553e-03
CMAES	8.347e-04	8.647e-03	3.978e-03	2.508e-03
MAC	8.347e-04	7.378e-03	4.480e-03	2.249e-03
SLS	8.347e-04	7.378e-03	4.633e-03	2.334e-03
RE	8.347e-04	1.042e-02	4.974e-03	2.944e-03
DE	8.347e-04	9.282e-03	5.363e-03	2.008e-03
ABC	1.169e-03	1.018e-02	6.561e-03	1.727e-03
BOHB	8.347e-04	1.252e-02	6.805e-03	2.297e-03
RL	9.682e-04	1.202e-02	6.945e-03	1.928e-03
TPE	8.347e-04	1.199e-02	7.175e-03	2.305e-03
HB	9.682e-04	1.272e-02	7.256e-03	1.918e-03

From the results in Table 7.6, one can say that PSO and IMAC have obtained the best overall performance. Interestingly, we can see that TPE and RL are among the worst performed algorithms. In TPE, We argue that this contradictory performance is due to the fact the algorithms is very sensitive to the increase of the problem dimension. In the case of RL, unlike supervised learning where feedback provided to the agent is correct set of actions for performing a task, it uses rewards and punishment as signals for positive and negative behavior. To do so, we require more data to train the RL in such high dimensional search space (i.e., more function evaluations). Also for multi-fidelity methods like BOHB and HB, there is no hope to beat metaheuristics by substantial amount in this highly structured search space. We can see that MAC suffers from the same problem; *curse of dimensionality* where the number of function evaluations needed to support the result often grows exponentially with the dimensionality. This motivated us to propose the IMAC which use a DE algorithm to generate the candidate points for its response surface model. From the results, we can see that IMAC have more enhanced results compared to MAC. Interestingly, we can see that in very early stage of evolution IMAC obtained a more rapid convergence speed in comparison with PSO and CMAES; as illustrated in Figure 7.6. The

same conclusion can be drawn regarding the cumulative distribution function in Figure 7.5.

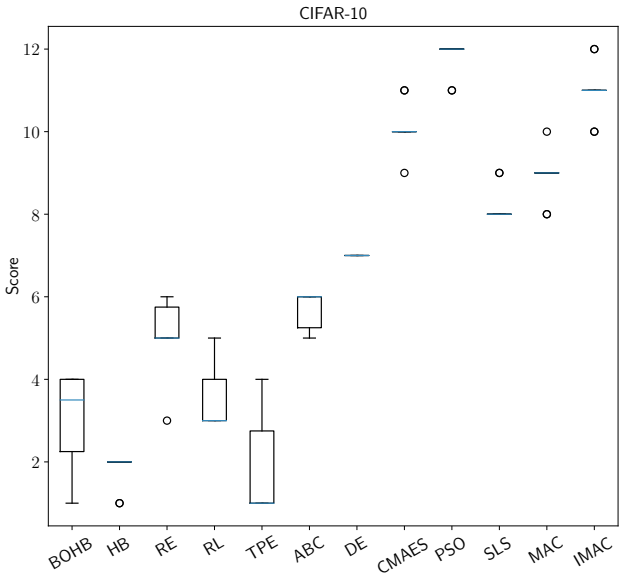


Figure 7.4: The ranking results for the single-objective neural architecture search algorithms based on the mean criterion. The plot uses the performance of 500 independent runs over 1000 function evaluations on CIFAR-10 dataset.

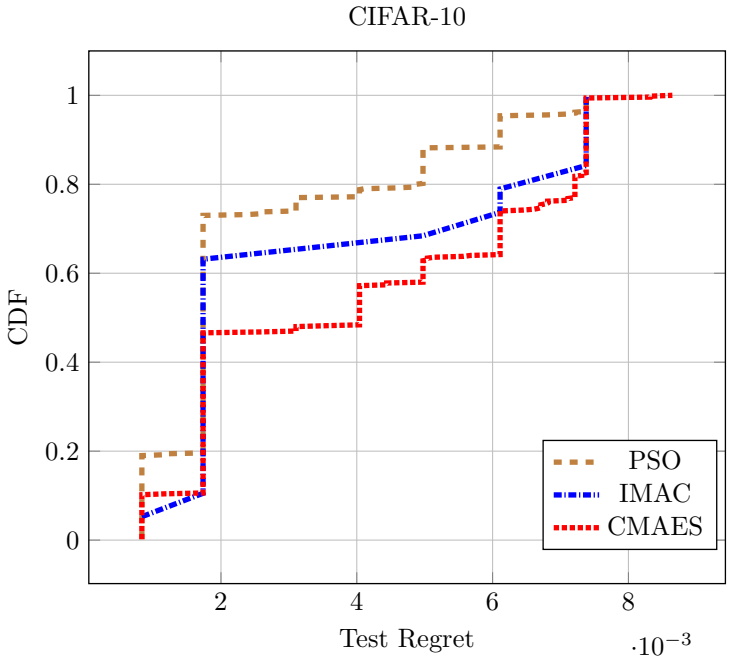


Figure 7.5: The obtained cumulative distribution function of the best algorithms for single-objective neural architecture search. The plot is based on the performance of 500 independent runs over 1000 function evaluations on CIFAR-10 dataset.

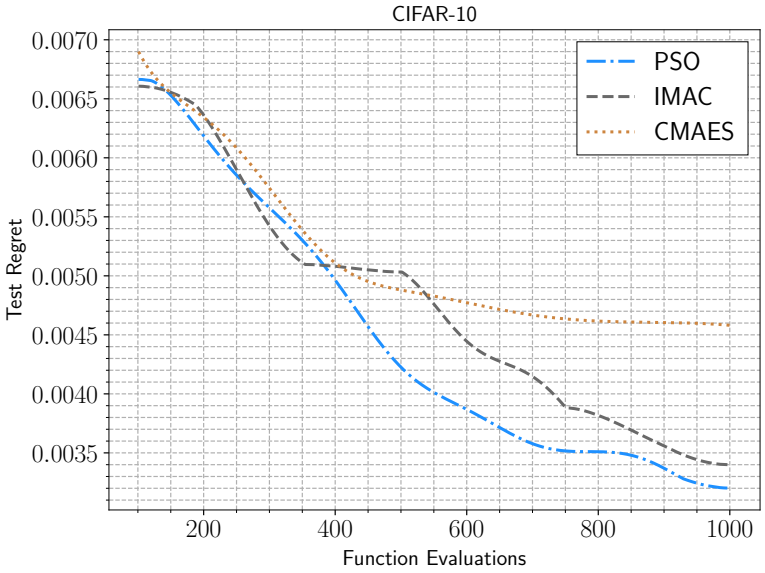


Figure 7.6: The converges rate for the best single-objective neural architecture search algorithms. The plot is based on the mean performance of 500 independent runs over 1000 function evaluations on CIFAR-10 dataset.

Multi-objective: We presented the results for multi-objective hyperparameter optimization and neural architecture search. The best, worst, mean, and standard deviation of the hypervolume are reported in Tables 7.7-7.11. Moreover, the corresponding ranking, CDF and convergence plots are depicted in Figures 7.7-7.12. All the considered algorithms are metaheuristics and so the number of function evaluations is considered to be 1000. The experiments are repeated for 500 different runs. To compute the hypervolume, all the algorithms are executed for 15 runs and the reference point is selected to be a slightly worse point than the nadir point. We used this reference point specification method for fair performance comparison of the multi-objective algorithms.

The results in Tables 7.7-7.10 indicates that SPEA2 performs better on Naval Propulsion and Slice Localization, NSGAII on Protein Structure, and SMSEMOA on Parkinsons Telemonitoring datasets. Form Figure 7.9, however, one can say that SPEA2 obtains the best convergence rate compared to the other competitive algorithms. For neural architecture search, PESA2 performs better on this high dimensional task according to the results in Tables 7.11 and Figure 7.12. This is in contradiction with the existing proposed methods in the literature which are are mainly use NSGAII as their basis algorithm. We would like to strongly encourage the researchers to adopt and apply other superior multi-objective algorithms in their research studies. These results reveal the fact that optimizing the machine learning pipe lines according to several objectives needs more comprehensive experiments and applying only the NSGAII algorithm might lead to inferior results.

Table 7.7: The reported hypervolume results for multi-objective hyperparameter optimization on Naval Propulsion dataset. The results show the performance of 500 independent runs over 1000 function evaluations. The results are sorted according to the mean criterion that makes it easier to read and understand.

Algorithm	Best	Worst	Mean	Std.
SPEA2	9.863e-01	9.391e-01	**9.74e-01**	5.453e-03
SMSEMOA	9.866e-01	9.182e-01	9.735e-01	8.668e-03
NSGAII	9.867e-01	9.504e-01	9.733e-01	5.718e-03
eNSGAII	9.858e-01	9.512e-01	9.732e-01	5.496e-03
FastPGA	9.856e-01	8.802e-01	9.682e-01	1.232e-02
NSGAIII	9.857e-01	9.262e-01	9.668e-01	8.588e-03
PESA2	9.855e-01	5.033e-01	9.623e-01	3.692e-02
eMOEA	9.829e-01	7.720e-01	9.610e-01	2.257e-02
IBEA	9.821e-01	8.917e-01	9.565e-01	1.563e-02
MOCell	9.852e-01	2.692e-01	9.562e-01	4.639e-02
DBEA	9.763e-01	7.745e-01	9.430e-01	2.079e-02
VEGA	9.751e-01	7.448e-01	9.422e-01	2.234e-02
MOEAD	9.617e-01	0.000e+00	7.984e-01	1.314e-01
RVEA	9.290e-01	1.273e-01	7.274e-01	1.349e-01
DENSEA	9.054e-01	0.000e+00	3.144e-01	2.757e-01
PAES	9.547e-01	0.000e+00	1.892e-01	2.924e-01

Table 7.8: The reported hypervolume results for multi-objective hyperparameter optimization on Protein Structure dataset. The results show the performance of 500 independent runs over 1000 function evaluations. The results are sorted according to the mean criterion that makes it easier to read and understand.

Algorithm	Best	Worst	Mean	Std.
NSGAII	9.729e-01	8.562e-01	**9.47e-01**	1.974e-02
eNSGAII	9.731e-01	8.449e-01	9.467e-01	2.009e-02
SPEA2	9.739e-01	8.542e-01	9.447e-01	2.291e-02
SMSEMOA	9.731e-01	7.470e-01	9.324e-01	3.137e-02
FastPGA	9.739e-01	7.606e-01	9.287e-01	3.602e-02
NSGAIII	9.702e-01	7.725e-01	9.197e-01	3.136e-02
IBEA	9.727e-01	7.354e-01	9.191e-01	3.783e-02
PESA2	9.741e-01	6.565e-01	9.163e-01	5.264e-02
eMOEA	9.731e-01	7.354e-01	9.127e-01	4.872e-02
DBEA	9.647e-01	7.342e-01	8.883e-01	4.163e-02
MOCell	9.727e-01	5.970e-01	8.826e-01	7.322e-02
VEGA	9.629e-01	7.370e-01	8.656e-01	4.565e-02
MOEAD	9.000e-01	3.854e-01	6.953e-01	9.112e-02
RVEA	9.050e-01	3.938e-01	6.630e-01	9.214e-02
DENSEA	8.795e-01	3.473e-03	4.886e-01	1.659e-01
PAES	9.440e-01	0.000e+00	3.670e-01	2.479e-01

Table 7.9: The reported hypervolume results for multi-objective hyperparameter optimization on Slice Localization dataset. The results show the performance of 500 independent runs over 1000 function evaluations. The results are sorted according to the mean criterion that makes it easier to read and understand.

Algorithm	Best	Worst	Mean	Std.
SPEA2	9.553e-01	9.111e-01	**9.45e-01**	6.861e-03
SMSEMOA	9.557e-01	7.728e-01	9.433e-01	1.275e-02
eNSGAII	9.537e-01	9.102e-01	9.414e-01	8.766e-03
NSGAII	9.542e-01	9.031e-01	9.404e-01	9.043e-03
FastPGA	9.549e-01	8.485e-01	9.381e-01	1.637e-02
PESA2	9.558e-01	7.482e-01	9.365e-01	2.131e-02
MOCell	9.559e-01	5.862e-01	9.296e-01	3.064e-02
NSGAIII	9.521e-01	8.693e-01	9.289e-01	1.304e-02
eMOEA	9.532e-01	8.432e-01	9.282e-01	1.922e-02
IBEA	9.513e-01	7.927e-01	9.132e-01	2.484e-02
DBEA	9.413e-01	8.169e-01	8.992e-01	2.138e-02
VEGA	9.422e-01	6.924e-01	8.881e-01	2.829e-02
MOEAD	8.804e-01	2.489e-01	6.814e-01	1.091e-01
RVEA	8.436e-01	1.858e-01	6.074e-01	1.188e-01
PAES	9.305e-01	0.000e+00	2.962e-01	2.873e-01
DENSEA	8.076e-01	0.000e+00	2.511e-01	2.512e-01

Table 7.10: The reported hypervolume results for multi-objective hyperparameter optimization on Parkinsons Telemonitoring dataset. The results show the performance of 500 independent runs over 1000 function evaluations. The results are sorted according to the mean criterion that makes it easier to read and understand.

Algorithm	Best	Worst	Mean	Std.
SMSEMOA	6.992e-01	2.004e-01	**5.16e-01**	6.552e-02
SPEA2	6.989e-01	2.040e-01	5.143e-01	6.349e-02
NSGAII	6.999e-01	1.633e-01	5.115e-01	7.627e-02
eNSGAII	6.989e-01	1.653e-01	5.087e-01	7.060e-02
IBEA	6.999e-01	2.212e-01	5.012e-01	7.307e-02
PESA2	6.995e-01	1.483e-03	4.837e-01	1.052e-01
FastPGA	6.985e-01	6.498e-02	4.801e-01	8.938e-02
eMOEA	6.995e-01	5.241e-02	4.683e-01	9.817e-02
NSGAIII	6.937e-01	1.438e-01	4.521e-01	7.784e-02
MOCell	6.995e-01	0.000e+00	4.480e-01	1.256e-01
VEGA	6.729e-01	0.000e+00	3.678e-01	1.043e-01
RVEA	6.662e-01	0.000e+00	3.164e-01	1.487e-01
DBEA	6.858e-01	0.000e+00	3.013e-01	1.435e-01
MOEAD	6.751e-01	0.000e+00	1.922e-01	1.801e-01
PAES	5.574e-01	0.000e+00	2.626e-02	9.090e-02
DENSEA	4.563e-01	0.000e+00	1.057e-02	5.063e-02

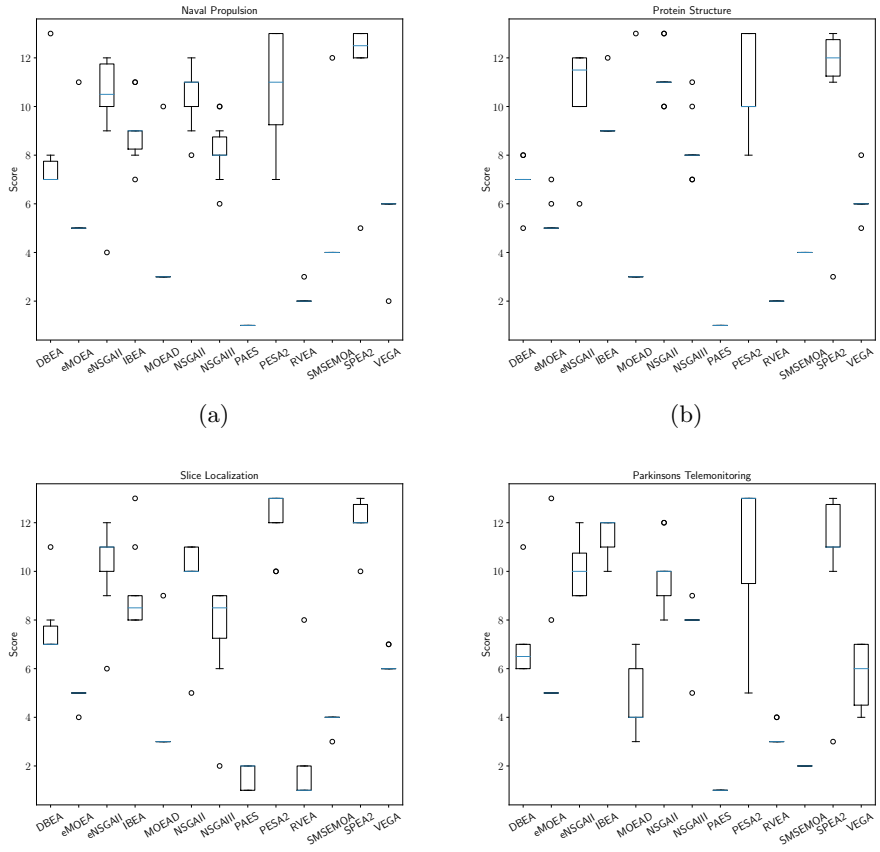


Figure 7.7: The ranking results for the multi-objective hyperparameter optimization. The plot uses the mean hypervolume performance of 500 independent runs over 1000 function evaluations.

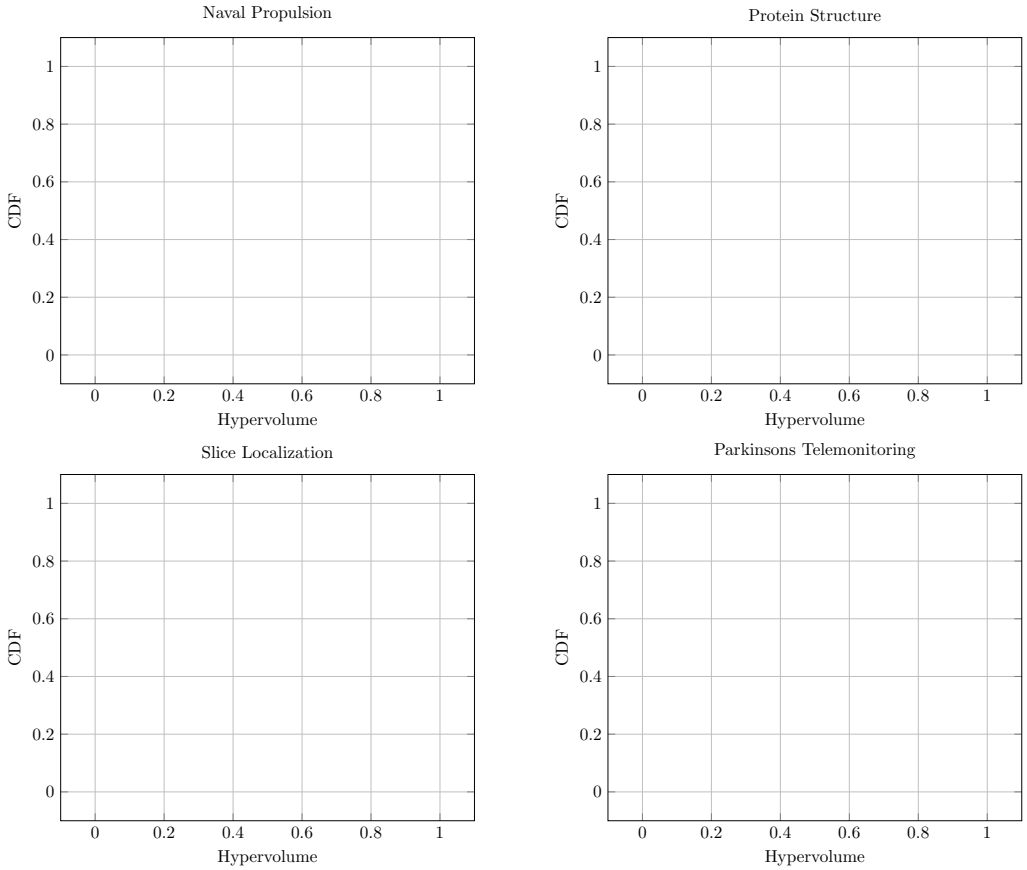


Figure 7.8: The obtained cumulative distribution function of the best algorithms for multi-objective hyperparameter optimization. The plot is based on the hypervolume of 500 independent runs over 1000 function evaluations.

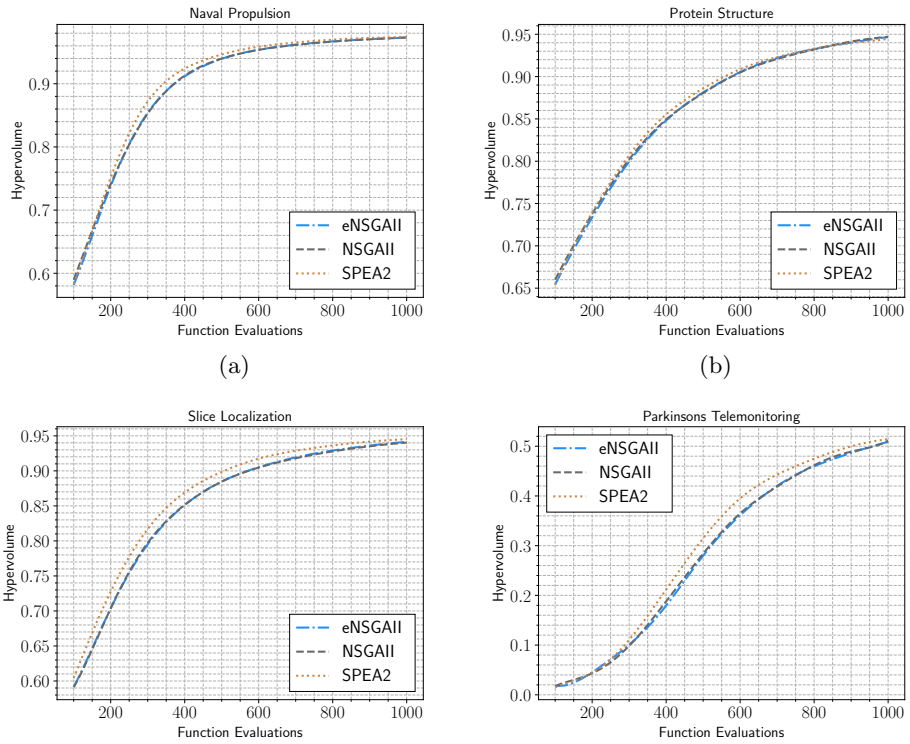


Figure 7.9: The converges rate for the best multi-objective hyperparameter optimization algorithms. The plot is based on the mean hypervolume performance of 500 independent runs over 1000 function evaluations.

Table 7.11: The reported hypervolume results for multi-objective neural architecture search. The results show the performance of 500 independent runs over 1000 function evaluations on CIFAR-10 dataset. The results are sorted according to the mean criterion that makes it easier to read and understand.

Algorithm	Best	Worst	Mean	Std.
PESA2	8.319e-01	7.593e-01	**8.07e-01**	1.282e-02
eMOEA	8.291e-01	7.634e-01	8.057e-01	1.217e-02
NSGAII	8.277e-01	7.575e-01	8.054e-01	1.067e-02
eNSGAII	8.236e-01	7.644e-01	8.041e-01	1.123e-02
IBEA	8.237e-01	7.693e-01	8.026e-01	8.931e-03
RVEA	8.305e-01	7.612e-01	8.012e-01	9.352e-03
SPEA2	8.327e-01	7.549e-01	8.011e-01	1.213e-02
FastPGA	8.236e-01	7.542e-01	7.977e-01	1.379e-02
DBEA	8.190e-01	7.524e-01	7.951e-01	1.280e-02
SMSEMOA	8.207e-01	7.519e-01	7.934e-01	1.437e-02
MOCell	8.297e-01	0.000e+00	7.883e-01	6.416e-02
NSGAIII	8.174e-01	7.439e-01	7.872e-01	1.481e-02
VEGA	8.153e-01	7.228e-01	7.802e-01	1.421e-02
MOEAD	8.128e-01	7.024e-01	7.664e-01	1.885e-02
DENSEA	7.384e-01	0.000e+00	3.831e-01	2.002e-01
PAES	7.815e-01	0.000e+00	2.535e-01	3.028e-01

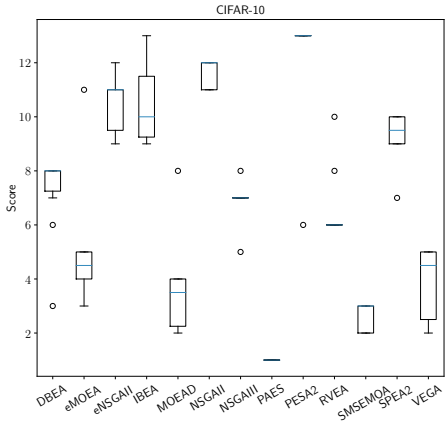


Figure 7.10: The ranking results for the multi-objective neural architecture search algorithms on CIFAR-10 dataset. The plot uses the mean hypervolume performance of 500 independent runs over 1000 function evaluations.

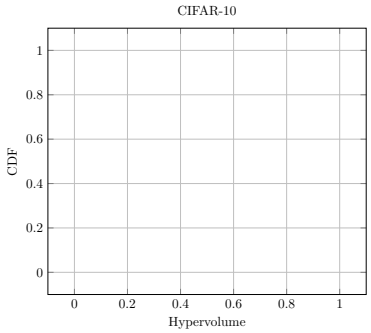


Figure 7.11: The obtained cumulative distribution function of the best algorithms for multi-objective neural architecture search. The plot is based on the hypervolume of 500 independent runs over 1000 function evaluations on CIFAR-10 dataset.

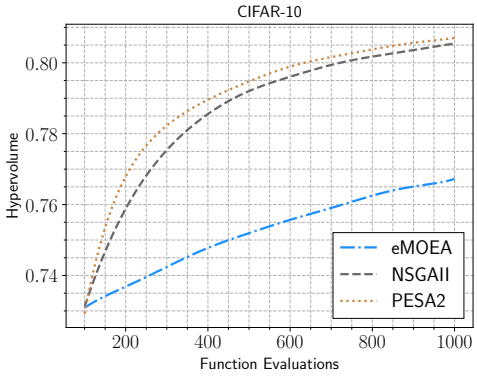


Figure 7.12: The converges rate for the best multi-objective neural architecture search algorithms. The plot is based on the mean hypervolume performance of 500 independent runs over 1000 function evaluations on CIFAR-10 dataset.

7.3.4 Implementation Notes

For this study, we use Socket programming for communication between the MOEA framework in Java and the AutoML benchmarks in Python. One socket (i.e., fitness function) listens on a particular port at an IP so that the TCP layer can identify the application that data is destined to be sent to (i.e., he MOEA framework). Compared to the latency of the shell/file exchange, the general cost of creating a new connection and all other costs such as kernel setup times are insignificant. This gives us the opportunity to use all the designed sate-of-the-art multi-objective approaches inside the MOEA framework. For the single-objective Bayesian algorithms, we used the standard Python implementations of the authors from their GitHub repository. The PSO, ABC, DE, and CS are carefully implemented according to the original study, while for CMAES we used the 'cmaes' Python library. The considered metaheuristics algorithms for both the single-objective and multi-objective cases encode the categorical variables as integer identities.

7.4 Practical End Use of Research

The results in this chapter can be mainly used by the researchers in the machine learning field so as to have an idea about the possible application of metaheuristics for having superior models. As we showed for the MAC, the metaheuristics can be easily applied to tune the parameters of the Bayesian algorithms which can significantly improve the results. This study also could help the researcher who would like to apply multi-objective algorithms by giving extensive results on a broad range of state-of-the-art methods.

7.5 Chapter Summary

In this chapter, we investigate the application of sing-objective and multi-objective cases to enhance the performance of the machine learning techniques. The obtained results are compared against the most recent Bayesian optimization models in the literature. The simulation results show that the metaheuristics have advantages of being used for both the single-objective and multi-objective cases. Moreover, these algorithms can be easily parallelized, while the Bayesian methods are mainly based on sequential models. We also notice that more enhanced results can be obtained by incorporating the metaheuristics for optimizing the parameters of the Bayesian methods.

Part III

Conclusion and Perspectives

This thesis started with the vision to investigate the possible synergies between machine learning and met-heuristics. In Part I, we proposed and trained different machine learning techniques that are capable of enhancing both the solution quality and the convergence rate. It was our clear mission not to focus on reproducing competitive results but to come up with an optimization approach that takes advantage of the machine learning properties. Accordingly, the first perspective draws attention to recent transfer learning and ensemble learning techniques. Thereby, an optimization algorithm can benefit from warm-star population initialization to reduce the required computational resources. In the same direction, it has shown that storing knowledge gained while solving one problem and applying it to a different but related problem can be also effective. We believe that this consideration should be investigated on the other instance-based optimization problems like TSP. Moreover, the application of such techniques for multi-objective problems can be even more interesting. Whether single-objective or multi-objective, effective machine learning techniques are central in the development of fast heuristic algorithms for computationally expensive problems.

In Part II, we place the application of meta-heuristics for enhancing the performance of machine learning in a wider perspective. The results verify that a simple PSO algorithm could outperform all the recent propositions which are specifically designed for improving the machine learning models. We would like to offer to consider meta-heuristics as strong competitive baselines in the machine learning community. Moreover, we believe that the application of multi-objective algorithms should not be limited to NSGA-II algorithm. Finally, this thesis point to a need to refocus on heuristic and meta-heuristics methods for the other machine learning models like RNN.

Bibliography

[1] Kamel Abdelouahab, Maxime Pelcat, Jocelyn Serot, and François Berry. Accelerating cnn inference on fpgas: A survey. *arXiv preprint arXiv:1806.01683*, 2018.

[2] Feras Al-Obeidat, Nabil Belacel, and Bruce Spencer. Combining machine learning and metaheuristics algorithms for classification method proaftn. In *Enhanced Living Environments*, pages 53–79. Springer, 2019.

[3] Cesare Alippi, Simone Disabato, and Manuel Roveri. Moving convolutional neural networks to embedded systems: the alexnet and vgg-16 case. In *Proceedings of the 17th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 212–223. IEEE Press, 2018.

[4] Md Zahangir Alom, Tarek M Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C Van Esesn, Abdul A S Awwal, and Vijayan K Asari. The history began from alexnet: a comprehensive survey on deep learning approaches. *arXiv preprint arXiv:1803.01164*, 2018.

[5] Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 136–145. JMLR. org, 2017.

[6] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989, 2016.

[7] M. Asafuddoula, T. Ray, and R. Sarker. A decomposition-based evolutionary algorithm for many objective optimization. *IEEE Transactions on Evolutionary Computation*, 19(3):445–460, 2015.

[8] Arthur Asuncion and David Newman. Uci machine learning repository, 2007.

[9] NH Awad, MZ Ali, JJ Liang, BY Qu, and PN Suganthan. Problem definitions and evaluation criteria for the cec 2017 special session and competition on single objective real-parameter numerical optimization. *Tech. Rep.*, 2016.

[10] Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Accelerating neural architecture search using performance prediction. *NeurIPS worshop on Meta-Learning*, 2017.

[11] Kristin P Bennett and Emilio Parrado-Hernández. The interplay of optimization and machine learning research. *Journal of Machine Learning Research*, 7(Jul):1265–1281, 2006.

[12] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[13] James Bergstra, Daniel Yamins, and David Daniel Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. 2013.

[14] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.

[15] Hadrien Bertrand, Roberto Ardon, Matthieu Perrot, and Isabelle Bloch. Hyperparameter optimization of deep neural networks: Combining hyperband with bayesian model selection.

[16] Robert B Best, Xiao Zhu, Jihyun Shim, Pedro EM Lopes, Jeetain Mittal, Michael Feig, and Alexander D MacKerell Jr. Optimization of the additive charmm all-atom protein force field targeting improved sampling of the backbone ϕ, ψ and side-chain $\chi 1$ and $\chi 2$ dihedral angles. *Journal of chemical theory and computation*, 8(9):3257–3273, 2012.

[17] Mauro Birattari, Zhi Yuan, Prasanna Balaprakash, and Thomas Stützle. F-race and iterated f-race: An overview. In *Experimental methods for the analysis of optimization algorithms*, pages 311–336. Springer, 2010.

[18] Stefan Bleuler, Marco Laumanns, Lothar Thiele, and Eckart Zitzler. Pisaa platform and programming language independent interface for search algorithms. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 494–508. Springer, 2003.

[19] Aymeric Blot, Holger H Hoos, Laetitia Jourdan, Marie-Éléonore Kessaci-Marmion, and Heike Trautmann. Mo-paramils: a multi-objective automatic algorithm configuration framework. In *International Conference on Learning and Intelligent Optimization*, pages 32–47. Springer, 2016.

[20] Antonio Bolufé-Röhler and Dania Tamayo-Vera. Machine learning based metaheuristic hybrids for s-box optimization. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–14, 2020.

[21] Borko Bošković and Janez Brest. Differential evolution for protein folding optimization based on a three-dimensional ab off-lattice model. *Journal of molecular modeling*, 22(10):252, 2016.

[22] J. Branke, S. Nguyen, C. W. Pickardt, and M. Zhang. Automated design of production scheduling heuristics: A review. *IEEE Transactions on Evolutionary Computation*, 20(1):110–124, Feb 2016.

[23] J. Brest, M. S. Mauec, and B. Bokovi. Single objective real-parameter optimization: Algorithm jso. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 1311–1318, June 2017.

[24] Janez Brest, Sao Greiner, Borko Boskovic, Marjan Mernik, and Viljem Zumer. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE transactions on evolutionary computation*, 10(6):646–657, 2006.

[25] Janez Brest, Mirjam Sepesy Maučec, and Borko Bošković. il-shade: Improved l-shade algorithm for single objective real-parameter optimization. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 1188–1195. IEEE, 2016.

[26] Janez Brest, Mirjam Sepesy Maučec, and Borko Bošković. Single objective real-parameter optimization: Algorithm jso. In *2017 IEEE congress on evolutionary computation (CEC)*, pages 1311–1318. IEEE, 2017.

[27] Edmund K Burke, Matthew R Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R Woodward. A classification of hyper-heuristic approaches: Revisited. In *Handbook of Metaheuristics*, pages 453–477. Springer, 2019.

[28] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[29] Han Cai, Jiacheng Yang, Weinan Zhang, Song Han, and Yong Yu. Path-level network transformation for efficient architecture search. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 678–687, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[30] Laura Calvet, Jésica de Armas, David Masip, and Angel A Juan. Learnheuristics: hybridizing metaheuristics with machine learning for optimization with dynamic inputs. *Open Mathematics*, 15(1):261–280, 2017.

[31] Jie Chen, Bin Xin, Zhihong Peng, Lihua Dou, and Juan Zhang. Optimal contraction theorem for exploration–exploitation tradeoff in search and optimization. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 39(3):680–691, 2009.

[32] Tianqi Chen, Emily Fox, and Carlos Guestrin. Stochastic gradient hamiltonian monte carlo. In *International conference on machine learning*, pages 1683–1691, 2014.

[33] Yukang Chen, Gaofeng Meng, Qian Zhang, Shiming Xiang, Chang Huang, Lisen Mu, and Xinggang Wang. Renas: Reinforced evolutionary neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4787–4796, 2019.

[34] R. Cheng, Y. Jin, M. Olhofer, and B. Sendhoff. A reference vector guided evolutionary algorithm for many-objective optimization. *IEEE Transactions on Evolutionary Computation*, 20(5):773–791, 2016.

[35] Ran Cheng, Miqing Li, Ye Tian, Xiaoshu Xiang, Xingyi Zhang, Shengxiang Yang, Yaochu Jin, and Xin Yao. Benchmark functions for the cec'2018 competition on many-objective optimization. Technical report, 2018.

[36] Kuo-Chen Chou. An unprecedented revolution in medicinal chemistry driven by the progress of biological science. *Current topics in medicinal chemistry*, 17(21):2337–2358, 2017.

[37] CA Coello Coello and M Salazar Lechuga. Mopso: A proposal for multiple objective particle swarm optimization. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, volume 2, pages 1051–1056. IEEE, 2002.

[38] Andrea Coraddu, Luca Oneto, Aessandro Ghio, Stefano Savio, Davide Anguita, and Massimo Figari. Machine learning approaches for improving condition-based maintenance of naval propulsion plants. *Proceedings of the Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment*, 230(1):136–153, 2016.

[39] David W. Corne, Nick R. Jerram, Joshua D. Knowles, and Martin J. Oates. Pesa-ii: Region-based selection in evolutionary multiobjective optimization. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, GECCO01, page 283290, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[40] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[41] Laizhong Cui, Genghui Li, Qiuzhen Lin, Jianyong Chen, and Nan Lu. Adaptive differential evolution algorithm with novel mutation strategies in multiple sub-populations. *Computers & Operations Research*, 67:155 – 173, 2016.

[42] Swagatam Das, Sankha Subhra Mullick, and Ponnuthurai N Suganthan. Recent advances in differential evolution–an updated survey. *Swarm and Evolutionary Computation*, 27:1–30, 2016.

[43] Saulo HP de Oliveira, Eleanor C Law, Jiye Shi, and Charlotte M Deane. Sequential search leads to faster, more efficient fragment-based de novo protein structure prediction. *Bioinformatics*, 34(7):1132–1140, 2018.

[44] K. Deb and H. Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, 18(4):577–601, 2014.

[45] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[46] Kalyanmoy Deb, Manikanth Mohan, and Shikar Mishra. A fast multiobjective evolutionary algorithm for finding wellspread pareto-optimal solutions. In *In KanGAL Report No. 2003002*. lndian Institute Of Technology Kanpur, 2002.

[47] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[48] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[49] Xingping Dong, Jianbing Shen, Wenguan Wang, Yu Liu, Ling Shao, and Fatih Porikli. Hyperparameter optimization for tracking with continuous deep q-learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 518–527, 2018.

[50] Juan J Durillo and Antonio J Nebro. jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10):760–771, 2011.

[51] Saber M Elsayed, Tapabrata Ray, and Ruhul A Sarker. A surrogate-assisted differential evolution algorithm with dynamic parameters selection for solving expensive optimization problems. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, pages 1062–1068. IEEE, 2014.

[52] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. In *International Conference on Learning Representations*, 2019.

[53] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

[54] Michael Emmerich, Nicola Beume, and Boris Naujoks. An emo algorithm using the hypervolume measure as selection criterion. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 62–76. Springer, 2005.

[55] Hamidreza Eskandari, Christopher D Geiger, and Gary B Lamont. Fastpga: A dynamic population sizing approach for solving expensive multiobjective optimization problems. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 141–155. Springer, 2007.

[56] Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. *arXiv preprint arXiv:1807.01774*, 2018.

[57] Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 1437–1446. PMLR, 10–15 Jul 2018.

[58] Pengmian Feng and Zhenyi Wang. Recent advances in computational methods for identifying anticancer peptides. *Current drug targets*, 20(5):481–487, 2019.

[59] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D Sculley. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1487–1495. ACM, 2017.

[60] Franz Graf, Hans-Peter Kriegel, Matthias Schubert, Sebastian Pölsterl, and Alexander Cavallaro. 2d image registration in ct images using radial image descriptors. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 607–614. Springer, 2011.

[61] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, 2018.

[62] David Hadka. Moea framework: a free and open source java framework for multiobjective optimization, 2012.

[63] 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.

[64] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[65] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*, 2017.

[66] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition. *Signal Processing Magazine*, 2012.

[67] Tobias Hinz, Nicolás Navarro-Guerrero, Sven Magg, and Stefan Wermter. Speeding up the hyperparameter optimization of deep convolutional neural networks. *International Journal of Computational Intelligence and Applications*, page 1850008, 2018.

[68] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

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

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

[71] Christian Igel and Bernhard Sendhoff. Synergies between evolutionary and neural computation. In *ESANN*, pages 241–252, 2005.

[72] Ilija Ilievski, Taimoor Akhtar, Jiashi Feng, and Christine Annette Shoemaker. Efficient hyperparameter optimization for deep learning algorithms using deterministic rbf surrogates. In *AAAI*, pages 822–829, 2017.

[73] Dario Izzo, Christopher Iliffe Sprague, and Dharmesh Vijay Tailor. Machine learning and evolutionary techniques in interplanetary trajectory design. In *Modeling and Optimization in Space Engineering*, pages 191–210. Springer, 2019.

[74] Nanda Dulal Jana, Jaya Sil, and Swagatam Das. Selection of appropriate metaheuristic algorithms for protein structure prediction in ab off-lattice model: a perspective from fitness landscape analysis. *Information Sciences*, 391:28–64, 2017.

[75] C. Jin, A. K. Qin, and K. Tang. Local ensemble surrogate assisted crowding differential evolution. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 433–440, May 2015.

[76] Yaochu Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, 1(2):61–70, 2011.

[77] Yaochu Jin and Bernhard Sendhoff. Pareto-based multiobjective machine learning: An overview and case studies. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(3):397–415, 2008.

[78] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[79] Laetitia Jourdan, Clarisse Dhaenens, and El-Ghazali Talbi. Using datamining techniques to help metaheuristics: A short survey. In *International Workshop on Hybrid Metaheuristics*, pages 57–69. Springer, 2006.

[80] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. In *Advances in Neural Information Processing Systems*, pages 2016–2025, 2018.

[81] Kirthevasan Kandasamy, Karun Raju Vysyaraju, Willie Neiswanger, Biswajit Paria, Christopher R. Collins, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Tuning Hyperparameters without Grad Students: Scalable and Robust Bayesian Optimisation with Dragonfly. *arXiv preprint arXiv:1903.06694*, 2019.

[82] Kyuchang Kang, Changseok Bae, Henry Wing Fung Yeung, and Yuk Ying Chung. A hybrid gravitational search algorithm with swarm intelligence and deep convolutional feature for object tracking optimization. *Applied Soft Computing*, 66:319–329, 2018.

[83] Dervis Karaboga and Bahriye Akay. A comparative study of artificial bee colony algorithm. *Applied mathematics and computation*, 214(1):108–132, 2009.

[84] Dervis Karaboga and Bahriye Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. *Journal of global optimization*, 39(3):459–471, 2007.

[85] Lawrence A Kelley, Stefans Mezulis, Christopher M Yates, Mark N Wass, and Michael JE Sternberg. The phyre2 web portal for protein modeling, prediction and analysis. *Nature protocols*, 10(6):845, 2015.

[86] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, 1995.

[87] Michael Peter Kennedy and Leon O Chua. Neural networks for nonlinear programming. *IEEE Transactions on Circuits and Systems*, 35(5):554–562, 1988.

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

[89] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[90] Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 528–536, Fort Lauderdale, FL, USA, 20–22 Apr 2017. PMLR.

[91] Aaron Klein and Frank Hutter. Tabular benchmarks for joint architecture and hyperparameter optimization. *arXiv preprint arXiv:1905.04970*, 2019.

[92] Joshua Knowles and David Corne. The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 1, pages 98–105. IEEE, 1999.

[93] Joshua B Kollat and Patrick M Reed. Comparing state-of-the-art evolutionary multi-objective algorithms for long-term groundwater monitoring design. *Advances in Water Resources*, 29(6):792–807, 2006.

[94] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[95] Andriy Kryshtafovych, Krzysztof Fidelis, and John Moult. Casp10 results compared to those of previous casp experiments. *Proteins: Structure, Function, and Bioinformatics*, 82:164–174, 2014.

[96] Kee Huong Lai, Zarita Zainuddin, and Pauline Ong. A study on the performance comparison of metaheuristic algorithms on the learning of neural networks. In *AIP Conference Proceedings*, volume 1870, page 040039. AIP Publishing LLC, 2017.

[97] Heung-Chang Lee, Do-Guk Kim, and Bohyung Han. Efficient decoupled neural architecture search by structure and operation sampling. *arXiv preprint arXiv:1910.10397*, 2019.

[98] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[99] Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*, pages 1071–1079, 2014.

[100] Bai Li, Ya Li, and Ligang Gong. Protein secondary structure optimization using an improved artificial bee colony algorithm based on ab off-lattice model. *Engineering Applications of Artificial Intelligence*, 27:70–79, 2014.

[101] Bai Li, Ya Li, and Ligang Gong. Protein secondary structure optimization using an improved artificial bee colony algorithm based on ab off-lattice model. *Engineering Applications of Artificial Intelligence*, 27:70 – 79, 2014.

[102] Guilin Li, Xing Zhang, Zitong Wang, Zhenguo Li, and Tong Zhang. Stacnas: Towards stable and consistent optimization for differentiable neural architecture search. *arXiv preprint arXiv:1909.11926*, 2019.

[103] Hui Li and Qingfu Zhang. Multiobjective optimization problems with complicated pareto sets, moea/d and nsga-ii. *IEEE transactions on evolutionary computation*, 13(2):284–302, 2008.

[104] Ke Li and Jitendra Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

[105] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638*, 2019.

[106] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

[107] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

[108] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

[109] Lisha Li, Kevin Jamieson, Afshin Rostamizadeh, Katya Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. Massively parallel hyperparameter tuning, 2018.

[110] Kresten Lindorff-Larsen, Stefano Piana, Kim Palmo, Paul Maragakis, John L Klepeis, Ron O Dror, and David E Shaw. Improved side-chain torsion potentials for the amber ff99sb protein force field. *Proteins: Structure, Function, and Bioinformatics*, 78(8):1950–1958, 2010.

[111] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen Awm Van Der Laak, Bram Van Ginneken, and Clara I Sánchez. A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88, 2017.

[112] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

[113] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

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

[115] Ilya Loshchilov, Marc Schoenauer, and Michele Sebag. Self-adaptive surrogate-assisted covariance matrix adaptation evolution strategy. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, pages 321–328, 2012.

[116] Rammohan Mallipeddi and Minho Lee. An evolving surrogate model-based differential evolution algorithm. *Applied Soft Computing*, 34:770–787, 2015.

[117] Rammohan Mallipeddi, Ponnuthurai N Suganthan, Quan-Ke Pan, and Mehmet Fatih Tasgetiren. Differential evolution algorithm with ensemble of parameters and mutation strategies. *Applied soft computing*, 11(2):1679–1696, 2011.

[118] Gábor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*, 2017.

[119] Juliane Müller. Matsumoto: the matlab surrogate model toolbox for computationally expensive black-box global optimization problems. *arXiv preprint arXiv:1404.4261*, 2014.

[120] Antonio J. Nebro, Juan J. Durillo, Francisco Luna, Bernab Dorronsoro, and Enrique Alba. Mocell: A cellular genetic algorithm for multiobjective optimization. *International Journal of Intelligent Systems*, 24(7):726–746, 2009.

[121] Vladimir Nekrasov, Hao Chen, Chunhua Shen, and Ian Reid. Fast neural architecture search of compact semantic segmentation models via auxiliary cells. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9126–9135, 2019.

[122] Su Nguyen, Mengjie Zhang, Mark Johnston, and Kay Chen Tan. Genetic programming for job shop scheduling. In *Evolutionary and Swarm Intelligence Algorithms*, pages 143–167. Springer, 2019.

[123] Shigeru Obayashi, Kalyanmoy Deb, Carlo Poloni, Tomoyuki Hiroyasu, and Tadahiko Murata. *Evolutionary Multi-Criterion Optimization: 4th International Conference, EMO 2007, Matsushima, Japan, March 5-8, 2007, Proceedings*, volume 4403. Springer, 2007.

[124] Anders Olsson, Göran Sandberg, and Ola Dahlblom. On latin hypercube sampling for structural reliability analysis. *Structural safety*, 25(1):47–68, 2003.

[125] Sergey Ovchinnikov, Hahnbeom Park, David E Kim, Frank DiMaio, and David Baker. Protein structure prediction using rosetta in casp12. *Proteins: Structure, Function, and Bioinformatics*, 86:113–121, 2018.

[126] Jooyoung Park and Irwin W Sandberg. Universal approximation using radial-basis-function networks. *Neural computation*, 3(2):246–257, 1991.

[127] Juan-Manuel Pérez-Rúa, Moez Baccouche, and Stephane Pateux. Efficient progressive neural architecture search. *arXiv preprint arXiv:1808.00391*, 2018.

[128] A. K. Qin, V. L. Huang, and P. N. Suganthan. Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Transactions on Evolutionary Computation*, 13(2):398–417, April 2009.

[129] A Kai Qin and Ponnuthurai N Suganthan. Self-adaptive differential evolution algorithm for numerical optimization. In *2005 IEEE congress on evolutionary computation*, volume 2, pages 1785–1791. IEEE, 2005.

[130] Amin Rahati and Hojjat Rakhshani. A gene expression programming framework for evolutionary design of metaheuristic algorithms. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 1445–1452. IEEE, 2016.

[131] Hojjat Rakhshani, Lhassane Idoumghar, Julien Lepagnot, and Mathieu Brevilliers. From feature selection to continuous optimization. *arXiv preprint arXiv:1909.09444*, 2019.

[132] Hojjat Rakhshani, Lhassane Idoumghar, Julien Lepagnot, and Mathieu Brévilliers. Mac: Many-objective automatic algorithm configuration. In Kalyanmoy Deb, Erik Goodman, Carlos A. Coello Coello, Kathrin Klamroth, Kaisa Miettinen, Sanaz Mostaghim, and Patrick Reed, editors, *Evolutionary Multi-Criterion Optimization*, pages 241–253, Cham, 2019. Springer International Publishing.

[133] Hojjat Rakhshani, Lhassane Idoumghar, Julien Lepagnot, and Mathieu Brévilliers. Mac: Many-objective automatic algorithm configuration. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 241–253. Springer, 2019.

[134] Hojjat Rakhshani, Lhassane Idoumghar, Julien Lepagnot, and Mathieu Brvilliers. Speed up differential evolution for computationally expensive protein structure prediction problems. *Swarm and Evolutionary Computation*, 50:100493, 2019.

[135] Hojjat Rakhshani and Amin Rahati. Snap-drift cuckoo search: A novel cuckoo search optimization algorithm. *Applied Soft Computing*, 52:771–794, 2017.

[136] PS Rana. Physicochemical properties of protein tertiary structure data set. *UCI Machine Learning Repository, pp. https://archive. ics. uci. edu/ml/datasets/Physicochemical+ Properties+ of+ Protein+ Tertiary+ Structure*, 2013.

[137] E Real, A Aggarwal, Y Huang, and QV Le. Aging evolution for image classifier architecture search. In *AAAI Conference on Artificial Intelligence*, 2019.

[138] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4780–4789, 2019.

[139] Rommel G Regis and Christine A Shoemaker. A stochastic radial basis function method for the global optimization of expensive functions. *INFORMS Journal on Computing*, 19(4):497–509, 2007.

[140] Rommel G. Regis and Christine A. Shoemaker. A stochastic radial basis function method for the global optimization of expensive functions. *INFORMS Journal on Computing*, 19(4):497–509, 2007.

[141] Giorgio Roffo and Simone Melzi. Features selection via eigenvector centrality. *Proceedings of New Frontiers in Mining Complex Patterns (NFMCP 2016)(Oct 2016)*, 2016.

[142] Giorgio Roffo, Simone Melzi, Umberto Castellani, and Alessandro Vinciarelli. Infinite latent feature selection: A probabilistic latent graph-based ranking approach. In *Computer Vision and Pattern Recognition*, 2017.

[143] Raymond Ros and Nikolaus Hansen. A simple modification in cma-es achieving linear time and space complexity. In *International Conference on Parallel Problem Solving from Nature*, pages 296–305. Springer, 2008.

[144] N. R. Sabar, M. Ayob, G. Kendall, and R. Qu. Automatic design of a hyper-heuristic framework with gene expression programming for combinatorial optimization problems. *IEEE Transactions on Evolutionary Computation*, 19(3):309–325, June 2015.

[145] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 901–909, 2016.

[146] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pages 2483–2493, 2018.

[147] J David Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of the first international conference on genetic algorithms and their applications, 1985*. Lawrence Erlbaum Associates. Inc., Publishers, 1985.

[148] Torsten Schwede, Jurgen Kopp, Nicolas Guex, and Manuel C Peitsch. Swiss-model: an automated protein homology-modeling server. *Nucleic acids research*, 31(13):3381–3385, 2003.

[149] T Senjyu, AY Saber, T Miyagi, K Shimabukuro, N Urasaki, and T Funabashi. Fast technique for unit commitment by genetic algorithm based on unit clustering. *IEE Proceedings-Generation, Transmission and Distribution*, 152(5):705–713, 2005.

[150] Margarita Reyes Sierra and Carlos A Coello Coello. Improving pso-based multi-objective optimization using crowding, mutation and ϵ-dominance. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 505–519. Springer, 2005.

[151] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[152] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[153] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *International conference on machine learning*, pages 2171–2180, 2015.

[154] Heda Song, Isaac Triguero, and Ender Özcan. A review on the self and dual interactions between machine learning and optimisation. *Progress in Artificial Intelligence*, 8(2):143–165, 2019.

[155] Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter. Bayesian optimization with robust bayesian neural networks. In *Advances in neural information processing systems*, pages 4134–4142, 2016.

[156] Michael Stein. Large sample properties of simulations using latin hypercube sampling. *Technometrics*, 29(2):143–151, 1987.

[157] Frank H Stillinger and Teresa Head-Gordon. Collective aspects of protein folding illustrated by a toy model. *Physical review E*, 52(3):2872, 1995.

[158] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[159] Witold K Surewicz, Henry H Mantsch, and Dennis Chapman. Determination of protein secondary structure by fourier transform infrared spectroscopy: a critical assessment. *Biochemistry*, 32(2):389–394, 1993.

[160] El-Ghazali Talbi. A unified taxonomy of hybrid metaheuristics with mathematical programming, constraint programming and machine learning. In *Hybrid Metaheuristics*, pages 3–76. Springer, 2013.

[161] El-Ghazali Talbi. Machine learning for metaheuristics-state of the art and perspectives. In *2019 11th International Conference on Knowledge and Smart Technology (KST)*, pages XXIII–XXIII. IEEE, 2019.

[162] Ryoji Tanabe and Alex Fukunaga. Success-history based parameter adaptation for differential evolution. In *2013 IEEE congress on evolutionary computation*, pages 71–78. IEEE, 2013.

[163] Ryoji Tanabe and Alex S Fukunaga. Improving the search performance of shade using linear population size reduction. In *2014 IEEE congress on evolutionary computation (CEC)*, pages 1658–1665. IEEE, 2014.

[164] Subarna Tripathi, Gokce Dane, Byeongkeun Kang, Vasudev Bhaskaran, and Truong Nguyen. Lcdet: Low-complexity fully-convolutional neural networks for object detection in embedded systems. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 411–420. IEEE, 2017.

[165] Athanasios Tsanas, Max A Little, Patrick E McSharry, and Lorraine O Ramig. Accurate telemonitoring of parkinson's disease progression by noninvasive speech tests. *IEEE transactions on Biomedical Engineering*, 57(4):884–893, 2009.

[166] Felipe AC Viana, Raphael T Haftka, and Valder Steffen. Multiple surrogates: how cross-validation errors can help us to obtain the best predictor. *Structural and Multidisciplinary Optimization*, 39(4):439–457, 2009.

[167] C. Wang, C. Xu, X. Yao, and D. Tao. Evolutionary generative adversarial networks. *IEEE Transactions on Evolutionary Computation*, 23(6):921–934, Dec 2019.

[168] Jiazhuo Wang, Jason Xu, and Xuejun Wang. Combination of hyperband and bayesian optimization for hyperparameter optimization in deep learning. *arXiv preprint arXiv:1801.01596*, 2018.

[169] Ning Wang, Yang Gao, Hao Chen, Peng Wang, Zhi Tian, and Chunhua Shen. Nas-fcos: Fast neural architecture search for object detection, 2019.

[170] Yong Wang, Zixing Cai, and Qingfu Zhang. Differential evolution with composite trial vector generation strategies and control parameters. *IEEE transactions on evolutionary computation*, 15(1):55–66, 2011.

[171] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[172] Guohua Wu, R Mallipeddi, and PN Suganthan. Problem definitions and evaluation criteria for the cec 2017 competition on constrained real-parameter optimization. *National University of Defense Technology, Changsha, Hunan, PR China and Kyungpook National University, Daegu, South Korea and Nanyang Technological University, Singapore, Technical Report*, 2017.

[173] Guohua Wu, Rammohan Mallipeddi, P.N. Suganthan, Rui Wang, and Huangke Chen. Differential evolution with multi-population based ensemble of mutation strategies. *Information Sciences*, 329:329–345, 2016. Special issue on Discovery Science.

[174] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: stochastic neural architecture search. In *International Conference on Learning Representations*, 2019.

[175] Chris Ying. Enumerating unique computational graphs via an iterative graph invariant. *arXiv preprint arXiv:1902.06192*, 2019.

[176] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. NAS-bench-101: Towards reproducible neural architecture search. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7105–7114, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[177] Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3712–3722, 2018.

[178] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018.

[179] J. Zhang, Z. Zhan, Y. Lin, N. Chen, Y. Gong, J. Zhong, H. S. H. Chung, Y. Li, and Y. Shi. Evolutionary computation meets machine learning: A survey. *IEEE Computational Intelligence Magazine*, 6(4):68–75, Nov 2011.

[180] Jingqiao Zhang and Arthur C Sanderson. Jade: adaptive differential evolution with optional external archive. *IEEE Transactions on evolutionary computation*, 13(5):945–958, 2009.

[181] Jun Zhang, Zhi-hui Zhan, Ying Lin, Ni Chen, Yue-jiao Gong, Jing-hui Zhong, Henry SH Chung, Yun Li, and Yu-hui Shi. Evolutionary computation meets machine learning: A survey. *IEEE Computational Intelligence Magazine*, 6(4):68–75, 2011.

[182] X. Zhang, Y. Gong, Y. Lin, J. Zhang, S. Kwong, and J. Zhang. Dynamic cooperative coevolution for large scale optimization. *IEEE Transactions on Evolutionary Computation*, 23(6):935–948, Dec 2019.

[183] Kaixiong Zhou, Qingquan Song, Xiao Huang, and Xia Hu. Auto-gnn: Neural architecture search of graph neural networks. *arXiv preprint arXiv:1909.03184*, 2019.

[184] Eckart Zitzler and Simon Künzli. Indicator-based selection in multiobjective search. In Xin Yao, Edmund K. Burke, José A. Lozano, Jim Smith, Juan Julián Merelo-Guervós, John A. Bullinaria, Jonathan E. Rowe, Peter Tiňo, Ata Kabán, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VIII*, pages 832–842, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[185] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. Spea2: Improving the strength pareto evolutionary algorithm. Technical report, 2001.

[186] Marc-André Zöller and Marco F. Huber. Survey on automated machine learning. *CoRR*, abs/1904.12054, 2019.

[187] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. 2017.

[188] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.