# Feature-Based Algorithm Selection for Constrained Continuous Optimisation

Shayan Poursoltan
Optimisation and Logistics
School of Computer Science
The University of Adelaide
Adelaide, SA 5005, Australia

Frank Neumann
Optimisation and Logistics
School of Computer Science
The University of Adelaide
Adelaide, SA 5005, Australia

***Abstract*—With this paper, we contribute to the growing research area of feature-based analysis of bio-inspired computing. In this research area, problem instances are classified according to different features of the underlying problem in terms of their difficulty of being solved by a particular algorithm. We investigate the impact of different sets of evolved instances for building prediction models in the area of algorithm selection. Building on the work of Poursoltan and Neumann [1], [2], we consider how evolved instances can be used to predict the best performing algorithm for constrained continuous optimisation from a set of bio-inspired computing methods, namely high performing variants of differential evolution, particle swarm optimization, and evolution strategies. Our experimental results show that instances evolved with a multi-objective approach in combination with random instances of the underlying problem allow to build a model that accurately predicts the best performing algorithm for a wide range of problem instances.**

**Keywords:** Constraints, Continuous Optimisation, Difficulty Prediction, Linear Constraints, Features

## I. Introduction

Throughout the history of heuristic optimisation, various methods have been proposed to solve constrained optimisation problems (COPs), especially non-linear ones. The main idea behind these algorithms is to tackle the constraints. Important approaches in this area are differential evolution (DE), particle swarm optimisation (PSO) and evolutionary strategies (ES). To handle the constraints, there have been many techniques applied to these algorithms such as penalty functions, special operators (separating the constraint and objective function treatment) and decoder based methods. We refer the reader to [3] for a survey of constraint handling techniques in evolutionary computation. Given a range of different algorithms for constrained continuous optimisation, we consider the algorithm selection problem (ASP) [4] which consists of selecting the best performing algorithm from a suite of algorithms for a given problem instance. In most circumstances, it is difficult to answer the following question: ”Can we estimate the likelihood that algorithm $A$ will be successful on a given constrained optimisation problem instance $I$?”. Recent works in the field show that it is possible to select the algorithm most likely to be best suited for a given problem instance [5], [6], [7]. Based on these studies, it is possible to find the links between problem characteristics and algorithm performance. The key to these investigations are problem features which can be used to predict the most suitable algorithm from a set of algorithms.

It is widely assumed that constraints play a vital role in COP's difficulty. Therefore, in this study we use the meta-learning framework outlined in [4], [8] to build a prediction model for a given COP. Our model predicts the best algorithm type (DE, ES and PSO) for a given COP based on their constraint features. The model inputs include the features of constraints of a given problem instance. It is shown in [1], [9] that by using an evolving approach, it is possible to generate problem instances covering a wide range of problem/algorithm difficulty. Such instances can be used to extract and analyse the features that make a problem hard or easy to solve for a given algorithm. For a detailed discussion on these constraints (linear, quadratic and their combination) we refer the reader to [1].

To build a reliable prediction model, we need to train it with variety of problem instances that are hard or easy for algorithm(s). Based on the investigations in [2], a multi-objective evolutionary algorithm can be used to generate constrained problem instances that are hard/easy for one algorithm but still easy/hard for the others. The authors show which features of the constraints make the problems hard for certain algorithm but still easy for the others. Hence, we use the same approach to generate problem instances to use in our model training phase. This can improve the accuracy of prediction model since the training instances are used to show the strengths and weaknesses of various algorithm types over constraint features. To illustrate the model's efficiency on constraints, we examine our model with generated testing problem instances such as hard/easy for one but easy/hard for the others and more general random instances. To show the model prediction ability over constraints (linear, quadratic and their combination), we experiment with problems having different objective functions.

The remainder of this paper is organised as follows: In Section II, we formally introduce constrained continuous optimisation problems and describe the the evolutionary algorithms that are used in our investigations. Moreover, background material related to the multi-objective evolver, the algorithm selection problem and meta-learning prediction models is provided Section III describes and compares all models trained with different subsets of instances from a multi-objective evolver. By choosing the best training data preference in Section III, the experimental analysis on various benchmark problems is described in Section IV. We then conclude with some remarks in Section V.

## II. Preliminaries

### A. Constrained Continuous Optimisation Problems

A constrained optimisation problem (COP) in the continuous space where the objective function should be minimized can be formulated as follows:

$$\begin{aligned} \text{minimise} \quad & f(x), \quad x = (x_1, \dots, x_n) \in \mathbb{R}^n \\ \text{subject to} \quad & g_i(x) \leq 0 \quad \forall i \in \{1, \dots, q\} \\ & h_j(x) = 0 \quad \forall j \in \{q+1, \dots, p\} \end{aligned} \tag{1}$$

In this formulation, $f$, $g_i$ and $h_j$ are real-valued functions on the search space $S = \mathbb{R}^n$, $q$ is the number of inequalities and $p - q$ is the number of equalities. These equality and inequality constraints could be linear or nonlinear. The feasible region $F \subseteq S$ of the search space $S$ is defined by

$$l_i \leq x_i \leq u_i, \qquad 1 \leq i \leq n$$

where both $l_i$ and $u_i$ are lower and upper bounds for the $i$th variable, $1 \leq i \leq n$. To simplify working with COPs, the equalities are replaced by the following inequalities:

$$|h_j(x)| \leq \varepsilon \quad \text{for} \;\; j = q+1 \;\; \text{to} \;\; p \tag{2}$$

where $\varepsilon$ is a small positive value. In all experiments, we use $\varepsilon = 1E^{-4}$ which is the same setting as for the CEC 2010 competition [10].

### B. Algorithms

In this section, we discuss the basic ideas about algorithms for constrained optimisation problems such as differential evolution, evolutionary strategies and particle swarm optimisation.

The $\varepsilon$-constrained differential evolution with an archive and gradient-based mutation ($\varepsilon$DEag) is the winner of 2010 CEC competition for continuous constrained optimisation problems [10]. This algorithm uses the $\varepsilon$-constrained method to transform algorithms for unconstrained problems to constrained ones. Potential solutions are ordered by $\varepsilon$-level comparison. This means, the lexicographic order is used in which constraint violation has priority over the function value. A detailed description of this algorithm can be found in [11].

In the case of evolutionary strategies, $(1+1)$ CMA-ES for constrained optimisation [12] is included in our experiment. This algorithm is a variant of $(1+1)$-ES which adapts the covariance matrix of its offspring distribution in addition to its global step size. The $(1+1)$ CMA-ES for constrained optimisation obtains approximations to the normal vectors directions in the vicinity of the current solution locations by applying low-pass filtering to steps that violate the constraints and reduces the variance of the offspring distribution in these directions. Adopting this method makes $(1+1)$ CMA-ES one of the most efficient algorithms for constrained optimisation problems. We refer the reader to [12] for detailed description.

The algorithm that is used in our investigations from the area of particle swarm optimisation is hybrid multi-swarm particle swarm optimisation (HMPSO). This algorithm divides the current swarms into sub-swarms and searches solutions between them in parallel. Particles in each sub-swarm locate their fittest local particle which attracts the particles to fitter positions. Having multiple sub-swarms near different optima increases the diversity of the algorithm. A detailed description of HMPSO can be found in [13].

### C. Multi-objective Investigations

In order to extract information about the strengths and weaknesses of certain algorithms on constrained optimisation problems, we consider problem instances with different kinds of difficulties for the considered algorithms. To do this, we evolve instances that are hard/easy for one algorithm and easy/hard for the others. Analysing the features of these instances helps us to extract knowledge regarding the strengths and weaknesses of the considered algorithms and gives reasons of why an algorithm performs well on one problem instance while the others have difficulties. Insights from such analyses can be used to develop more efficient prediction model for automated algorithm selection.

We use a multi-objective DE algorithm (evolver) [14] to evolve constraints that make problem instances hard for one algorithm and easy for the others following the approach outlined in [1], [2]. The authors of these articles show the constraint (linear and/or quadratic and their combination) features that are significantly contributing to problem difficulty for certain algorithms. We refer the reader to [1], [2] for a detailed description.

### D. Algorithm Selection Problem

It is difficult to understand which algorithms or types of algorithms are more efficient to solve a given COP. The problem of determining the best algorithm to solve a given problem instance is referred to as the "Algorithm Selection Problem" (see Rice [4]). In his work, Rice proposed a model with four main characteristics: a set of problem instances $F$, a set of algorithms $A$, measures for the cost of performing algorithms on particular problem ($Y$), and a set of characteristics of problem instances ($C$). The illustration for Rice general algorithm selection framework is shown in Figure 1 which predicts the performance $y(a(f))$ of a given algorithm $a$ on a problem $f$ by extracted features $c$. The idea is to extract features from a given problem instance and select the most appropriate algorithm or predict the performance of the algorithm based on these features. This framework has been extended by [8], [15], [16] in a variety of computational problem domains using meta-learning framework. So, if the values are features of problems with algorithm performance measure are known beforehand, then it is possible to use a learning strategy to predict the algorithm performance based on the problem features.

### E. Prediction Model

Our prediction model is implemented based on the work of Munoz et al. [5]. This approach has originally been proposed for unconstrained continuous optimisation problems

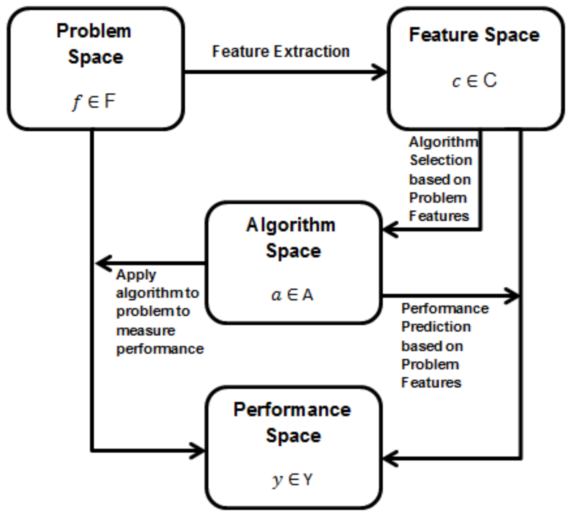
Problem Space
$f \in F$
Feature Extraction
Feature Space
$c \in C$
Algorithm Selection based on Problem Features
Algorithm Space
$a \in A$
Apply algorithm to problem to measure performance
Performance Prediction based on Problem Features
Performance Space
$y \in Y$

Fig. 1: The framework of general problem of algorithm selection and performance prediction using problem features based on [4].

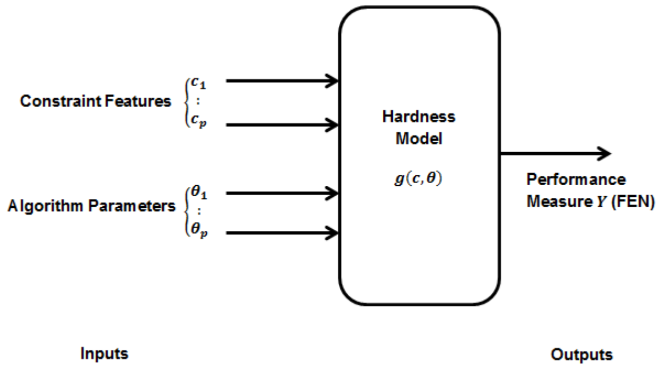

Fig. 2: Meta-learning prediction model for a constrained continuous optimisation problem

using landscapes features. In our study, inputs are problem constraint features and algorithm parameters. The output is predicted required function evaluation number (FEN) to solve the problem. This model can be used to predict the algorithm behaviour on a given problem instance. To achieve this we use popular basic technique for model building used in [5]. A high-level overview of the used prediction model is shown in Figure 2.

There have been many attempts to train these prediction models with random generated or benchmark problem instances which could not fully include all problem instances with difficulty variations. To improve this, we cover a wider spectrum of difficulty by evolving two sets of instances with extreme problem difficulties. These extreme difficulty instances are the ones which are hard for one algorithm and easy for the others or easy for one and still hard for the rest.

To build our regression model, we are using the same approach as [5]. This model is a multi-layered feed-forward neural network with 2 hidden layers and 10 neurons in each layer. For training the model, we use a Levenberg-Marquardt back-propagation algorithm [17] package using Matlab R2014b. To train this model, we use evolved instances that are generated from multi-objective evolver in [2]. The prediction model inputs are given COPs constraint features and algorithm parameter values.

## III. Prediction Model based on Evolved Instances

Our goal is to introduce a reliable prediction model using constraint features. The reliability can be improved by choosing proper set of learning data. The accuracy of this prediction model depends on many factors such as the relevance of constraint features, the diversity of instances used to train the model and its training method. Therefore, we train our prediction model on instances obtained from multi-objective evolver described in [2]. These instances are hard for one algorithm but still easy for the other (or easy for one and hard for the other algorithms).

The prediction model uses constraint features to predict the best algorithm type for a given constrained optimisation problem. These constraint features are constraint coefficients relationships such as standard deviation, angle between constraint hyperplanes, feasibility ratio in vicinity of optimum and number of constraints. The details of these features are discussed in [1]. For algorithm parameters, we use the parameter settings for DE, CMA-ES and HMPSO given in [11], [13], [12].

Our goal of building a prediction model is to identify the best algorithm type for a given problem. Therefore, the model output predicts the most suited algorithm and required FEN to solve a given constrained problem. The suggested FENs denote numbers of function evaluation which are needed by different algorithms to solve a given COP.

In the following we train our prediction model with variety of instance subsets generated by the multi-objective evolver. We choose different combinations of subsets of instances to maximise our prediction model accuracy upon a given constrained problem. In the different training phases instance subsets are selected from extreme points, Pareto optimal instances, randomly generated instances and a combination of Pareto optimal instances and random points from multi-objective evolver solution population in [2]. We then compare the prediction accuracy of the different prediction models. For all experiments in Section III, we generate 30 problem instances that are hard/easy for one algorithm type but still easy/hard for the others. We then run our experimented prediction models on each of these problem instances to find the suggested algorithm type and their required FEN.

### A. *Extreme Instances*

We first train our prediction model with an extreme instance subset which covers the extreme points of the Pareto front in the multi-objective evolver population set. These extreme solutions are selected from evolved instances that are easiest/hardest for one and hardest/easiest for the other algorithms at the same time.

TABLE I: Comparison of prediction models extreme (EP-PM), Pareto front (PF-PM), random only (RO-PM) and Pareto+random points (PFR-PM). Instances are hard/easy for the algorithm named in the first column and easy/hard for the others. For example, DE hard (1 c) in the first line are problem instances with 1 constraint that are hard/easy for DE algorithm but easy/hard for the others. The last 20 instances are random instances with different numbers of constraints.

| Problem | Correct algorithm / FEN | EP-PM predicted algorithm / FEN | PF-PM predicted algorithm / FEN | RO-PM predicted algorithm / FEN | PFR-PM predicted algorithm / FEN | EP-PM average deviation of FEN | PF-PM average deviation of FEN | RO-PM average deviation of FEN | PFR-PM average deviation of FEN |
|---|---|---|---|---|---|---|---|---|---|
| DE hard (1 c) | **ES / 41.5K** | ES / 43.7K | ES / 43.5K | PSO/ 38.1K | ES / 40.2K | 2.9K | 3.2K | 8.1K | 2.1K |
| ES hard (1 c) | **DE / 45.7K** | PSO/ 41.6K | PSO/ 40.3K | DE / 51.8K | DE / 43.6K | 4.1K | 4.5K | 5.8K | 3.1K |
| PSO hard (1 c) | **DE / 37.2K** | DE / 38.9K | DE / 35.1K | PSO/ 57.3K | DE / 39.1K | 2.3K | 2.2K | 25.3K | 1.5K |
| DE hard (2 c) | **ES / 45.2K** | ES / 42.5K | ES / 43.1K | ES / 57.9K | PSO/ 41.9K | 2.6K | 2.0K | 20.8K | 1.9K |
| ES hard (2 c) | **DE / 46.4K** | DE / 47.2K | DE / 48.5K | PSO/ 53.7K | DE / 45.8K | 2.4K | 1.6K | 14.3K | 1.9K |
| PSO hard (2 c) | **DE / 43.6K** | DE / 41.5K | DE / 42.3K | DE / 55.7K | DE / 41.8K | 1.8K | 1.6K | 15.3K | 2.2K |
| DE hard (3 c) | **ES / 45.2K** | ES / 43.1K | ES / 42.9K | PSO/ 57.3K | ES / 43.8K | 2.7K | 2.7K | 14/3K | 2.6K |
| ES hard (3 c) | **PSO/ 48.2K** | PSO/ 43.5K | DE / 49.3K | DE / 62.7K | PSO/ 42.9K | 4.8K | 2.5K | 15.0K | 4.6K |
| PSO hard (3 c) | **DE / 49.6K** | DE / 47.2K | DE / 47.5K | ES / 43.2K | DE / 46.8K | 2.4K | 2.6K | 11.3K | 3.0K |
| DE hard (4 c) | **ES / 47.2K** | PSO/ 49.2K | ES / 49.2K | PSO/ 49.1K | ES / 49.2K | 2.5K | 2.2K | 3.5K | 1.8K |
| ES hard (4 c) | **PSO/ 48.9K** | PSO/ 46.3K | PSO/45.1K | DE / 39.8K | PSO/ 46.8K | 2.8K | 3.0K | 8.1K | 2.5K |
| PSO hard (4 c) | **ES / 50.8K** | DE / 49.2K | ES / 51.9K | DE / 63.9K | ES / 50.3K | 2.6K | 1.9K | 9.5K | 2.6K |
| DE hard (5 c) | **ES / 53.3K** | ES / 49.3K | ES / 55.9K | PSO/ 49.2K | ES/ 52.4K | 2.4K | 2.6K | 7.1K | 2.1K |
| ES hard (5 c) | **DE / 53.8K** | DE / 51.4K | DE / 51.9K | PSO/ 47.1K | DE / 52.5K | 2.1K | 2.0K | 7.1K | 1.2K |
| PSO hard (5 c) | **DE / 51.3K** | DE / 49.1K | ES / 53.1K | ES / 45.2K | DE / 50.2K | 2.8K | 3.1K | 9.7K | 1.6K |
| DE easy (1 c) | **DE / 48.9K** | DE / 45.3K | PSO/ 71.2K | DE / 41.2K | DE / 46.4K | 3.6K | 12.0K | 8.8K | 2.6K |
| ES easy (1 c) | **ES / 54.7K** | ES / 48.3K | ES / 47.2K | ES / 64.2K | ES / 48.2K | 4.8K | 4.4K | 10.5K | 4.4K |
| PSO easy (1 c) | **PSO/ 41.9K** | PSO/ 48.2K | PSO/ 45.8K | ES / 62.1K | PSO/ 45.7K | 4.8K | 3.9K | 15.9K | 2.1K |
| DE easy (2 c) | **DE / 44.5K** | DE / 49.4K | DE / 48.1K | DE / 54.2K | DE / 48.1K | 4.6K | 3.3K | 11.5K | 3.3K |
| ES easy (2 c) | **ES / 55.3K** | ES / 51.4K | ES / 52.9K | PSO/ 65.1K | ES / 52.3K | 3.1K | 2.8K | 12.0K | 3.6K |
| PSO easy (2 c) | **PSO /49.4K** | PSO/ 51.6K | PSO/ 47.1K | PSO/ 58.1K | PSO/ 48.1K | 3.5K | 3.0K | 10.6K | 2.8K |
| DE easy (3 c) | **DE / 48.4K** | DE / 41.8K | DE / 42.4K | DE / 59.1K | DE / 42.5K | 4.0K | 4.1K | 8.6K | 3.8K |
| ES easy (3 c) | **ES / 46.2K** | ES / 48.2K | ES / 47.9K | PSO/ 42.4K | ES/ 46.9K | 2.9K | 2.2K | 9.5K | 2.3K |
| PSO easy (3 c) | **PSO/ 49.1K** | PSO/ 46.8K | PSO/ 47.0K | PSO/ 58.2K | PSO/ 47.9K | 3.1K | 3.7K | 13.4K | 2.9K |
| DE easy (4 c) | **DE / 53.2K** | DE / 50.6K | DE / 50.8K | DE / 64.2K | DE / 51.8K | 1.8K | 2.6K | 8.1K | 2.2K |
| ES easy (4 c) | **ES / 48.9K** | ES / 51.2K | ES / 46.2K | ES / 59.1K | ES / 50.8K | 2.8K | 2.9K | 8.6K | 2.0K |
| PSO easy (4 c) | **DE / 55.5K** | PSO/ 54.2K | PSO/ 58.5K | PSO/ 60.4K | DE / 55.2K | 8.2K | 6.4K | 6.9K | 6.2K |
| DE easy (5 c) | **DE / 57.8K** | DE / 59.2K | DE / 56.7K | PSO/ 70.9K | DE / 55.9K | 1.2K | 1.8K | 9.3K | 1.6K |
| ES easy (5 c) | **ES / 55.7K** | ES / 53.2K | ES / 52.8K | ES / 62.3K | ES / 56.3K | 3.7K | 3.9K | 4.6K | 2.6K |
| PSO easy (5 c) | **PSO/ 57.6K** | PSO/ 55.8K | PSO/ 56.1K | PSO/ 64.3K | PSO/ 56.2K | 1.7K | 2.2K | 6.3K | 1.2K |
| Rnd. 1 (1 c) | **PSO/53.2K** | DE / 53.2K | PSO/ 48.8K | PSO/ 50.9K | PSO/ 51.7K | 10.0K | 4.2K | 4.8K | 1.7K |
| Rnd. 2 (1 c) | **DE / 61.9K** | ES / 43.5K | DE / 67.3K | DE / 55.2K | DE / 64.4K | 16.4K | 5.1K | 4.8K | 1.2K |
| Rnd. 3 (2 c) | **DE / 59.8K** | ES / 48.1K | DE / 51.5K | DE / 49.2K | DE / 55.0K | 12.7K | 6.4K | 7.2K | 3.4K |
| Rnd. 4 (2 c) | **ES / 59.4K** | PSO/ 55.3K | ES / 64.1K | ES / 63.2K | ES / 61.4K | 12.1K | 5.9K | 6.0K | 2.9K |
| Rnd. 5 (3 c) | **DE / 65.4K** | DE / 45.2K | ES / 57.1K | ES / 63.5K | DE / 61.9K | 13.5K | 5.8K | 3.9K | 3.6K |
| Rnd. 6 (3 c) | **PSO/ 61.4K** | ES / 45.6K | PSO/ 55.1K | PSO/ 56.8K | PSO/ 57.8K | 18.1K | 7.6K | 8.4K | 4.0K |
| Rnd. 7 (4 c) | **ES / 65.8K** | DE / 71.6K | ES / 71.3K | PSO/ 71.2K | ES / 69.4K | 12.1K | 5.5K | 6.7K | 3.6K |
| Rnd. 8 (4 c) | **DE / 71.1K** | PSO/ 62.7K | ES / 70.5K | ES / 68.3K | ES / 73.7K | 11.5K | 6.8K | 6.6K | 11.6K |
| Rnd. 9 (5 c) | **DE / 82.7K** | ES / 83.9K | DE / 87.9K | DE / 88.3K | DE / 84.2K | 9.1K | 5.4K | 5.2K | 3.6K |
| Rnd. 10 (5 c) | **PSO/ 68.4K** | DE / 78.4K | PSO/ 75.6K | PSO/ 75.3K | PSO/ 65.8K | 15.3K | 8.4K | 8.4K | 5.7K |
| Rnd. 11 (6 c) | **DE / 44.6K** | ES / 49.3K | DE / 51.2K | ES / 52.4K | DE / 48.5K | 9.8K | 6.9K | 8.3K | 2.5K |
| Rnd. 12 (6 c) | **DE / 45.2K** | ES / 48.2K | DE/ 49.2K | DE / 50.2K | DE / 48.7K | 9.3K | 5.9K | 6.5K | 4.8K |
| Rnd. 13 (7 c) | **PSO/ 59.1K** | DE / 48.2K | PSO/ 52.7K | PSO/ 49.2K | PSO/ 54.9K | 11.8K | 6.0K | 6.5K | 3.7K |
| Rnd. 14 (7 c) | **PSO/ 63.4K** | ES / 61.2K | PSO/ 59.6K | PSO/ 57.9K | PSO/ 61.7K | 10.2K | 6.4K | 7.0K | 4.5K |
| Rnd. 15 (8 c) | **PSO/ 65.3K** | DE / 63.4K | ES / 59.9K | ES / 55.2K | ES / 62.7K | 8.1K | 10.2K | 12.0K | 7.3K |
| Rnd. 16 (8 c) | **DE / 69.2K** | PSO/ 74.3K | DE / 63.1K | DE / 61.2K | DE / 66.5K | 8.2K | 6.9K | 6.8K | 4.4K |
| Rnd. 17 (9 c) | **ES / 75.9K** | ES / 68.5K | ES / 68.4K | ES / 65.2K | ES / 70.3K | 6.5K | 6.2K | 8.3K | 4.9K |
| Rnd. 18 (9 c) | **PSO/ 68.3K** | DE / 71.4K | PSO/ 72.6K | PSO/ 70.2K | PSO/ 71.2K | 7.9K | 6.8K | 5.5K | 4.2K |
| Rnd. 19 (10 c) | **ES /91.2K** | DE / 81.2K | DE / 85.1K | DE / 83.1K | ES / 86.9K | 9.8K | 5.7K | 7.1K | 3.7K |
| Rnd. 20 (10 c) | **PSO/ 87.3K** | PSO/ 78.2K | PSO/ 82.5K | PSO/ 83.5K | PSO/ 85.9K | 10.2K | 5.3K | 5.9K | 2.6K |

To determine the actual accuracy of our extreme point prediction model (EP-PM), we select 1500 extreme instances. To analyse and test the quality of this prediction model, we use two sets of testing problem instances that we already know their best algorithm and required FEN. The first one is the set of problem instances that are hard/easy for one algorithm but easy/hard for the others. This set can improve the accuracy of EP-PM for given problems that fall into extreme-like evolved problem instances. However, it is very likely that the real world given COP is similar to other evolved instance subset types. Therefore, as a second set, we use random (general) testing problem instances to analyse the EP-PM with a potential real world given problem. We need to mention that we already know about their best algorithm and required FEN.

Our results for EP-PM are summarised in Table I for instances that are hard/easy for one and easy/hard for the other algorithms. Also, the results for testing random instances are shown in this table. Moreover, DE hard (1 c) denotes testing Sphere problem instances with 1 linear constraint that are hard for DE algorithm but still easy for the others. The information about actual correct algorithm and required FEN to solve a problem instance is indicated. The model not only suggests the best algorithm with its required FEN but also predicts the FENs required to solve a given problem with other algorithms (not the best ones). Then, to compare various models in depth, we include the information about the average of all various algorithm FENs deviations from their actual ones.

Based on the results, EP-PM performs acceptable on extreme-like testing instances. This performance is acceptable on predicted best algorithm type and FEN along with average value for FENs deviation for other algorithms. But, the model is not capable of predicting algorithm and required FEN for testing random (more general) instances. This means, the proposed model (EP-PM) is not accurate enough for randomly generated real world instances and the difference between predicted and actual FEN is considerable. Also, the average deviation of required FEN for other algorithm is high and not accurate.

To summarize, although the EP-PM model performs fairly accurate on instances that are grouped into extreme points evolved instances, still needs improvement to handle other subsets (such as random generated instances). The likelihood of given COP which is more similar to random evolved instances are higher. Thus, this motivates us to examine other subsets from evolving algorithm population instances for our training phase. This could be moving along the Pareto front line and choosing more instances from this category.

## B. Pareto Front Instances

We now show that in order to improve the accuracy of our prediction model, we can include different varieties of evolved instance subsets for its training step. The idea behind this choice is to obtain a model that can predict more general forms of given constrained problems. Of course it is vital to predict best algorithm for a given problem which is considerably hard/easy for one and still easy/hard for the other algorithms, but we also need to include more forms of generality to our prediction model. So, we need to move along Pareto front line in multi-objective evolver population set for our learning phase. This could increase the ability of our model to predict algorithms for more general given COP which is not similar to extreme point instances.

Given a total number of 3000 instances from evolver Pareto front line, we train our Pareto front prediction model (PF-PM). This preference could increase PF-PM ability to predict more general forms of given COPs. To compare the quality of our prediction model and other models with different learning phases, we use same extreme and random generated testing instances used in previous section. The results shown in Table I indicate an improving accuracy for random testing instances used for previous model EP-PM (using extreme instances). Looking at the Table I, we see that moving towards Pareto front line in evolver population set for choosing learning instances increases the accuracy of predicted FEN for predicted (best) algorithm. Also, the average deviations of FEN for all other algorithms indicate the PF-PM is more accurate than EP-PM for testing random instances. This improvement is acceptable in algorithm type prediction but we still need to improve the predicted required FEN.

The Pareto front prediction model (PF-PM) has some strengths and weaknesses. Although its error rate for predicting correct algorithms is improved, there is still considerable difference between the actual required FENs and predicted ones. As testing instances are selected mostly from more general instances (not close to extreme points), we need to experiment other training instances subsets. In order to address more general form, based on results so far, our next move is to choose more random instances from evolver population set for our training set. This could result in increasing the accuracy of our model for more general forms of given COPs.

## C. Random Instances

The initial prediction models discussed earlier (EP-PM and PF-PM) have some limitations. The results for PF-PM show an increase in accuracy of prediction for testing COPs which are not similar to extreme points, but the predicted required FEN still needs an improvement. Our goal is to design a prediction model with an ability to predict all possible given COPs. These COPs are within the range of extreme to random like instances. Based on previous results, it is shown that moving from extreme to Pareto front line instances increases the model accuracy (see Section III-B). Hence, to decrease the error rate for required FEN for more general testing COPs we choose only random instances for testing phase. These random instances are selected from evolver population set.

We use 3000 random instances from multi-objective evovler population set for our random only prediction model (RO-PM). To assess the accuracy of our random only prediction model we use the same testing instances applied to EP-PM and PF-PM. Table I indicates the actual and predicted FEN and algorithm of hard/easy and random testing instances for RO-PM. As it is observed, the random only prediction model (RO-PM) fails to predict suitable algorithm for a given COP. This failure include both predicted algorithm and required FEN. Results show that moving through random instances in evolver population and choose only random instances increase the number of incorrect predictions. Also, comparing to previous models, RO-PM accuracy is decreased for testing instances similar to extreme points (easy/hard instances).

It is shown that the accuracy of resulting model with Pareto front instances (PF-PM) is improved by selecting different subsets (Pareto front line) than extreme points. This improvement is analysed by experimenting more general form of testing COPs. Therefore, this motivated us to experiment only random solutions in order to build more accurate prediction model for given COPs. By selecting only random instances to train the new model, it is observed the predicted algorithm and required FEN is not accurate as Pareto front (PF-PM) and extreme points (EP-PM) models. It can be translated as excluding instances from Pareto front line for training step decreases the accuracy of prediction model. Also, the RO-PM model failed to predict testing instances which are similar to evolver extreme points (easy/hard instances). So, other possibility is to use a combination of both Pareto front and random instances from evolver population for model training.

## D. Pareto Front with Random Instances

We already considered three types of prediction models (EP-PM, PF-PM and RO-PM). These models have different prediction accuracy upon choosing different varies of evolver subsets for their training phases. Our goal of building a prediction model is to minimise its error rate in both algorithm and required FEN. So far, we understand that moving from extreme points (EP-PM) to Pareto front (PF-PM) for training step increases the accuracy of prediction model. However, moving further and choosing only random points from evolver (RO-PM) is not the solution for covering all possible given COPs (extreme and random like instances). In other words, there should be a trade-off relation between moving towards random points from extreme and random instances in multi-objective evolver population. Therefore, our preference for training phase is a combination set of Pareto front and random instances from multi-objective evolver. Not only it covers instances that are hard/easy for one and easy/hard for the other algorithms, but also it can predict general given COPs more accurately.

To train our Pareto front with random instances prediction model (PFR-PM), we use 3000 points from evolver population set (1500 each). To compare and assess the prediction model accuracy we experiment our PFR-PM with the same testing COPs for the former models. The results for Pareto front and random prediction model (PFR-PM) are shown in Table I. It is observed that including both Pareto front line and random only instances can be effective in accuracy improvement. Analysing the results, it is obvious the error rate for both predicted algorithm and required FEN are decreased. Also, the model is able to predict required FEN using other algorithms (not the best one) more accurate. This can be concluded using lower average deviation of FENs for PFR-PM. The reason behind this is that to cover all possible given COPs, we use both Pareto front and random points from evolver population for out training phase. In other words, our model is trained with constraint characteristics and features of both types of COPs (Pareto front and random instances).

It is shown that choosing the proper subsets for training phase is effective in prediction model accuracy. The results for four prediction models suggest that moving from extreme points to random instances can improve the quality of prediction model. It is found that there is a trade-off relationship in choosing instances that are close to random or extreme points (from evolver) for training phase. Results analysis shows by selecting a combination subsets from both random and Pareto front instances (PFR-PM) for training step, we can improve our model prediction quality. This improvement is in both selected algorithm and also its required FEN. By selecting the best model (PFR-PM), in the following, we examine it in a more detailed approach.

## IV. Experiments on Benchmarks

Our goal is to design highly accurate prediction model for a given COP based on its constraint features. As mentioned earlier, in order to improve the model accuracy, it needs to be trained with COP instances that are generated using multi-objective evolver. So far, we analysed the results for all four types of prediction models trained with extreme points (EP-PM), Pareto front (PF-PM), random only (RO-PM) and combination of Pareto front with random (PFR-PM) instances. We have experimented our prediction model with various subsets of training data to analyse the best preference. As

TABLE II: Comparison of PFR-PM with RO-PM models for Sphere function with various types of constraints (linear (L), quadratic (Q) and their combination). Average deviation of FEN denotes the average of differences between actual and predicted required FEN for PFR-PM and RO-PM.

| Problem | Success rate RO-PM | Success rate FR-PM | Average deviation of FEN for RO-PM | Average deviation of FEN for PFR-PM | p-value |
|---|---|---|---|---|---|
| Sphere, 1L | 2 | 26 | 7.8K | 2.4K | 0.004 |
| Sphere, 2L | 1 | 27 | 8.6K | 1.8K | 0.013 |
| Sphere, 3L | 1 | 26 | 12.6K | 3.2K | 0.018 |
| Sphere, 4L | 5 | 28 | 13.6K | 3.5K | 0.006 |
| Sphere, 5L | 1 | 28 | 17.4K | 2.7K | 0.028 |
| Sphere, 1Q | 1 | 27 | 11.9K | 2.1K | 0.035 |
| Sphere, 2Q | 1 | 26 | 13.4K | 2.6K | 0.038 |
| Sphere, 3Q | 3 | 28 | 15.8K | 3.7K | 0.043 |
| Sphere, 4Q | 1 | 29 | 19.3K | 3.1K | 0.026 |
| Sphere, 5Q | 5 | 28 | 21.6K | 4.3K | 0.035 |
| Sphere, 4L, 1Q | 2 | 24 | 13.4K | 2.5K | 0.007 |
| Sphere, 3L, 2Q | 2 | 26 | 13.8K | 2.8K | 0.004 |
| Sphere, 2L, 3Q | 3 | 28 | 16.1K | 3.8K | 0.031 |
| Sphere, 1L, 4Q | 5 | 27 | 18.9K | 3.7K | 0.016 |

results indicate, the most accurate prediction model is the one which is trained with combination of Pareto front and random subset of evolver population (PFR-PM). This model is capable of predicting algorithms for almost all possible given testing COPs such as the ones similar to extreme (hard/easy) or ordinary instances (random) in a multi-objective evolver population. Also, the prediction ability for required function evaluation number (FEN) has been significantly increased. Therefore, in order to assess our optimised prediction model (see Section III-D), we decide to experiment it with our newly designed benchmark. In order to analyse the capability of the prediction model (PFR-PM) on constraints, we use fixed objective function with various numbers of linear, quadratic (and their combination) constraints. Then, we test other well-known objective function to see the relationship of constraints and our prediction model.

For this experiment, we train our PFR-PM with 3000 instances from both Pareto front and random instances of evolving algorithm population set (1500 each). We also use optimised algorithm parameter settings for each algorithm suggested in [11], [13], [12]. In order to show the accuracy of model on prediction over constraints, we examine the model on various well-known objective functions such as Sphere (bowl-shaped), Ackley (many local optima) and Rosenbrock (valley-shaped). Also, to evaluate the effectiveness of our model on constrained problems, we use various numbers and types of constraints. Tables II, III and IV show the prediction results for Sphere, Ackley and Rosenbrock objective functions respectively. The results show the number of correct algorithm types prediction (success rate) from 30 different tests. Also, one step further, the average deviations of required FEN (the correct and predicted one) for the predicted algorithms are calculated.

Table II compares the prediction results for our proposed model (PFR-PM) and random only model (RO-PM) for Sphere COPs. The results indicate the effectiveness of choosing the proper subset training instances. It is observed that the prediction algorithm success rate for our proposed model (PFR-

TABLE III: Comparison of PFR-PM with RO-PM models for Ackley function with various types of constraints (linear (L), quadratic (Q) and their combination). Average deviation of FEN denotes the average of differences between actual and predicted required FEN for PFR-PM and RO-PM.

| Problem | Success rate RO-PM | Success rate FR-PM | Average deviation of FEN for RO-PM | Average deviation of FEN for PFR-PM | p-value |
|---|---|---|---|---|---|
| Ackley, 1L | 0 | 27 | 9.3K | 2.5K | 0.043 |
| Ackley, 2L | 1 | 27 | 11.5K | 3.2K | 0.016 |
| Ackley, 3L | 2 | 25 | 10.3K | 2.7K | 0.004 |
| Ackley, 4L | 1 | 28 | 14.7K | 3.6K | 0.008 |
| Ackley, 5L | 6 | 29 | 13.8K | 4.5K | 0.025 |
| Ackley, 1Q | 2 | 29 | 16.3K | 3.5K | 0.046 |
| Ackley, 2Q | 1 | 27 | 17.7K | 4.1K | 0.026 |
| Ackley, 3Q | 0 | 25 | 18.3K | 3.7K | 0.043 |
| Ackley, 4Q | 3 | 27 | 16.9K | 5.1K | 0.048 |
| Ackley, 5Q | 2 | 29 | 21.9K | 5.8K | 0.034 |
| Ackley, 4L, 1Q | 4 | 24 | 15.8K | 4.2K | 0.032 |
| Ackley, 3L, 2Q | 2 | 26 | 16.7K | 4.6K | 0.012 |
| Ackley, 2L, 3Q | 3 | 24 | 16.8K | 4.9K | 0.006 |
| Ackley, 1L, 4Q | 0 | 28 | 19.8K | 4.3K | 0.021 |

TABLE IV: Comparison of PFR-PM with RO-PM models for Rosenbrock function with various types of constraints (linear (L), quadratic (Q) and their combination). Average deviation of FEN denotes the average of differences between actual and predicted required FEN for PFR-PM and RO-PM.

| Problem | Success rate RO-PM | Success rate FR-PM | Average deviation of FEN for RO-PM | Average deviation of FEN for PFR-PM | p-value |
|---|---|---|---|---|---|
| Rosenbrock, 1L | 2 | 26 | 10.3K | 3.3K | 0.038 |
| Rosenbrock, 2L | 0 | 26 | 11.5K | 4.6K | 0.035 |
| Rosenbrock, 3L | 3 | 25 | 12.7K | 3.6K | 0.002 |
| Rosenbrock, 4L | 4 | 27 | 15.8K | 5.2K | 0.035 |
| Rosenbrock, 5L | 5 | 28 | 19.4K | 5.1K | 0.028 |
| Rosenbrock, 1Q | 2 | 27 | 17.4K | 4.1K | 0.017 |
| Rosenbrock, 2Q | 1 | 29 | 21.5K | 4.7K | 0.043 |
| Rosenbrock, 3Q | 4 | 26 | 21.3K | 5.7K | 0.037 |
| Rosenbrock, 4Q | 3 | 28 | 18.5K | 5.2K | 0.043 |
| Rosenbrock, 5Q | 2 | 28 | 24.6K | 6.9K | 0.004 |
| Rosenbrock, 4L, 1Q | 1 | 28 | 14.7K | 3.6K | 0.004 |
| Rosenbrock, 3L, 2Q | 0 | 24 | 17.4K | 4.7K | 0.024 |
| Rosenbrock, 2L, 3Q | 4 | 25 | 19.5K | 3.6K | 0.029 |
| Rosenbrock, 1L, 4Q | 2 | 27 | 21.3K | 5.2K | 0.006 |

PM) is significantly better than RO-PM for all Sphere COPs using various combinations of constraints. The success rate (out of 30 tests) for newly testing given COP is significantly higher for PFR-PM comparing to RO-PM. Also, the low value average deviation of predicted FEN and actual one for PFR-PM represents its higher accuracy in predicting the algorithm performance in terms of function evaluation number.

By observing the Tables III and IV, we realise that our prediction model (PFR-PM) is reliable in predicting with only constraints. In other words, experimenting other types of objective functions (Bowl-shaped, many local-optima and valley shaped) with accurate results shows the ability of the model to predict based on constraints. Based on the Table III, the lower value of FEN average deviation indicates the higher accuracy of PFR-PM for Ackley COPs. Also, the results for Rosenbrock COPs with 1 to 5 linear (L), quadratic (Q) constraints (and their combination) shows the accuracy of PFR-PM comparing to RO-PM. The average deviation of FEN for Rosenbrock problems denotes the significantly close predicted FEN with PFR-PM (see Table IV).

As mentioned before, the output of our proposed prediction model (PFR-PM) includes predicted algorithm with its required FEN. It is observed that the prediction model is capable of suggesting the best algorithm and required FEN based on constraint features of given COP. Due to the stochastic nature of evolutionary optimisation, the above benchmark tests are repeated 30 times and the two-tail t-test significance is performed for average deviations of FEN. The significant level $\alpha$ is considered as 0.05. The *p*-values for significance of a difference between FEN average deviation of Pareto front with random (PFR-PR) and random only (RO-PR) models for each Sphere, Ackley and Rosenbrock are shown in Tables II, III and IV respectively. The results show that the difference in FEN Average deviation are significant and less than 0.05.

As discussed earlier, the idea of designing a prediction model based on instance features is rather a novel approach in algorithm selection problem. Training a model with COP instances from multi-objective evolver improves the prediction accuracy. The performance prediction (FEN) and suggested algorithm can be used to produce the final output of our prediction model. As we know selecting a suitable algorithm for a given problem requires substantial amount of time. In contrast, in our approach, we only need to extract features of a problem once and the model produces the final output. It is observed that selecting different sets of training instances improves the prediction model success rate. We designed and examined various prediction model using different subsets of problem instances from evolver population set. In order to show the ability of the prediction model only based on constraints features we use various objective functions. Results for these COPs with different combinations of objective functions and constraints indicate that the model is highly accurate in algorithm and required FEN prediction.

## V. Conclusion

In this paper, we examined the impact of different types of problem instances that can be used in prediction models for constrained continuous optimisation. Our resulting prediction model captures the links between constraint features, algorithm type performance and the required function evaluation number. The model inputs are considered as constraint features and selected parameter settings. The outputs includes the required function evaluation number and most suited algorithm type to solve the given COP.

The model was trained (using NN learning strategy) with evolved COP instances. To improve the accuracy of the model we used evolved instances that are hard/easy for one and easy/hard for the other algorithms. These training instances are generated with multi-objective evolver. We first, chose various subsets of instances from multi-objective evolver population set. The experimental results show that our prediction model based on constraint features is able to give a good prediction when using the Pareto front instances in combination with random instances.

## Acknowledgements

This research has been supported by the Australian Research Council under grant agreements DP130104395 and DP140103400.

## References

[1] S. Poursoltan and F. Neumann, "A feature-based comparison of evolutionary computing techniques for constrained continuous optimisation," in *Neural Information Processing*, ser. Lecture Notes in Computer Science, S. Arik, T. Huang, W. K. Lai, and Q. Liu, Eds. Springer International Publishing, 2015, vol. 9491, pp. 332–343. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-26555-1_38

[2] ——, "A feature-based analysis on the impact of set of constraints for $\varepsilon$-constrained differential evolution," in *Neural Information Processing*, ser. Lecture Notes in Computer Science, S. Arik, T. Huang, W. K. Lai, and Q. Liu, Eds. Springer International Publishing, 2015, vol. 9491, pp. 344–355. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-26555-1_39

[3] E. Mezura-Montes and C. A. Coello Coello, "Constraint-handling in nature-inspired numerical optimization: past, present and future," *Swarm and Evolutionary Computation*, vol. 1, no. 4, pp. 173–194, 2011.

[4] J. R. Rice, "The algorithm selection problem," 1975.

[5] M. A. Muñoz, M. Kirley, and S. K. Halgamuge, "A meta-learning prediction model of algorithm performance for continuous optimization problems," in *Parallel Problem Solving from Nature-PPSN XII*. Springer, 2012, pp. 226–235.

[6] K. Smith-Miles, "Towards insightful algorithm selection for optimisation using meta-learning concepts," in *WCCI 2008: IEEE World Congress on Computational Intelligence*. IEEE, 2008, pp. 4118–4124.

[7] B. Bischl, O. Mersmann, H. Trautmann, and M. Preuß, "Algorithm selection based on exploratory landscape analysis and cost-sensitive learning," in *Proceedings of the 14th annual conference on Genetic and evolutionary computation*. ACM, 2012, pp. 313–320.

[8] K. A. Smith-Miles, R. J. James, J. W. Giffin, and Y. Tu, "A knowledge discovery approach to understanding relationships between scheduling problem structure and heuristic performance," in *Learning and intelligent optimization*. Springer, 2009, pp. 89–103.

[9] S. Poursoltan and F. Neumann, "A feature-based analysis on the impact of linear constraints for $\varepsilon$-constrained differential evolution," in *Evolutionary Computation (CEC), 2014 IEEE Congress on*. IEEE, 2014, pp. 3088–3095.

[10] R. Mallipeddi and P. N. Suganthan, "Problem definitions and evaluation criteria for the cec 2010 competition on constrained real-parameter optimization," *Nanyang Technological University, Singapore*, 2010.

[11] T. Takahama and S. Sakai, "Constrained optimization by the $\varepsilon$ constrained differential evolution with an archive and gradient-based mutation," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*. IEEE, 2010, pp. 1–9.

[12] D. V. Arnold and N. Hansen, "A (1+ 1)-cma-es for constrained optimisation," in *Proceedings of the 14th annual conference on Genetic and evolutionary computation*. ACM, 2012, pp. 297–304.

[13] Y. Wang and Z. Cai, "A hybrid multi-swarm particle swarm optimization to solve constrained optimization problems," *Frontiers of Computer Science in China*, vol. 3, no. 1, pp. 38–52, 2009.

[14] T. Robič and B. Filipič, "Demo: Differential evolution for multiobjective optimization," in *Evolutionary Multi-Criterion Optimization*. Springer, 2005, pp. 520–533.

[15] F. Hutter, Y. Hamadi, H. H. Hoos, and K. Leyton-Brown, "Performance prediction and automated tuning of randomized and parametric algorithms," in *Principles and Practice of Constraint Programming-CP 2006*. Springer, 2006, pp. 213–228.

[16] K. Leyton-Brown, E. Nudelman, and Y. Shoham, "Empirical hardness models: Methodology and a case study on combinatorial auctions," *Journal of the ACM (JACM)*, vol. 56, no. 4, p. 22, 2009.

[17] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the Society for Industrial & Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.