Master's Thesis

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Symbolic regression as a surrogate model in evolutionary algorithms

Bc. Vladimir Perić

May 2016
Supervisor: Ing. Petr Pošík, Ph.D.

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Computer Science and Engineering

DIPLOMA THESIS ASSIGNMENT

Student: **Bc. Vladimir Perić**

Study programme: Open Informatics
Specialisation: Software Engineering

Title of Diploma Thesis: **Symbolic regression as a surrogate model in evolutionary algorithms**

Guidelines:

1. Learn the principles of existing methods of surrogate modeling in evolutionary algorithms.
2. Study existing methods for fast symbolic regression.
3. Choose/create/modify a symbolic regression method suitable for building surrogate models for evolutionary algorithms. Give a special attention to the dynamic nature of the modeling task.
4. Choose 2 types of evolutionary algorithms and compare their symbolic regression surrogate-assisted versions with surrogate-free baselines, and if possible, also with other surrogate-assisted versions based on conventional surrogate models.
5. Assess the contributions of symbolic regression-based surrogate modeling in evolutionary algorithms.

Bibliography/Sources:

[1] Ilya Loshchilov. Surrogate-assisted Evolutionary Algorithms. PhD Thesis, University Paris-Sud, 2013.
[2] Yaochu Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. Swarm and Evolutionary Computation 1 (2011) 61&#8211;70
[3] Ignacio Arnaldo, Una-May O'Reilly, and Kalyan Veeramachaneni. 2015. Building Predictive Models via Feature Synthesis. In Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO '15), Sara Silva (Ed.). ACM, New York, NY, USA, 983-990.
[4] Searson, DP. GPTIPS 2: an open-source software platform for symbolic data mining. Chapter 22 in Handbook of Genetic Programming Applications, A.H. Gandomi et al., (Eds.), Springer, New York, NY, 2015.
[5] T. McConaghy, FFX: Fast, Scalable, Deterministic Symbolic Regression Technology, Genetic Programming Theory and Practice IX, Edited by R. Riolo, E. Vladislavleva, and J. Moore, Springer, 2011.

Diploma Thesis Supervisor: Ing. Petr Pošík, Ph.D.

Valid until the end of the summer semester of academic year 2016/2017

[illegible]
prof. Ing. Filip Železný, Ph.D.
Head of Department

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
-4-
L.S.

[illegible]
prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 12, 2016

Acknowledgement / Declaration

I would like to thank my mentor, Ing. Petr Pošík, Ph.D., whose wholehearted aid made this work possible, and all of my friends, who made slacking off so enjoyable.

Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme "Projects of Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated.

I declare that I worked out the presented thesis independently and I quoted all used sources of information in accord with Methodical instructions about ethical principles for writing academic thesis.

In Prague, 27th May 2016

[illegible]

Abstrakt / Abstract

Evoluční algoritmy jsou často omezeny počtem funkčních ohodnocení které jsou dostupné během řešení problémů optimalizace černé skříňky. Tato práce popisuje jeden z přístupu k zlepšení diferenciální evoluce pomocí náhradních modelů generovaných symbolickou regresi. Tři algoritmy pro symbolickou regresi — kvadratické modely, genetické programování a rozšířená rychlá těžba funkcí (extended fast function extraction) — jsou porovnány s učicí metodou náhodného lesa a s diferenciální evoluci bez použití náhradných modelů. Empirické výsledky ukázují, že použití náhradních modelů generovaných symbolickou regresi vede k zlepšení výkonu ve všech otestováných příkladech a během každé fáze vyhledávání.

Klíčová slova: optimalizace černé skříňky, symbolická regrese, náhradní model, diferenciální evoluce

Překlad titulu: Symbolická regrese jako náhradní model v evolučních algoritmech

Evolutionary algorithms are often limited by the number of function evaluations available in black-box optimization problems. One possible approach to enhance a representative evolutionary algorithm, differential evolution, with surrogate models built using symbolic regression is presented in this thesis. Three symbolic regression algorithms — quadratic models, genetic programming and extended fast function extraction — were compared against random forest surrogate models and regular differential evolution. Empirical results have shown that symbolic regression surrogate models improve performance in all tested problems and during each stage of the search process.

Keywords: black-box optimization, differential evolution, surrogate model, symbolic regression

Contents

Tables / Figures

Chapter 1
Introduction

An optimization problem is the computational problem of finding the optimal solution among the set of all feasible solutions, as defined by an objective function. In black-box optimization problems, no information is available on the nature of the objective function; instead, the box can only be queried for the function value at a specific point. Such problems often occur in practice, with an additional constraint on the total number of function evaluations available.

Evolutionary algorithms [1] are a set of metaheuristics which have been successfully used in solving optimization problems [2]. As these algorithms make no assumptions about the function they are optimizing, they are a good fit for black-box optimization problems. However, evolutionary algorithms require many function evaluations to produce a good solution.

One possible approach to mitigating this disadvantage is the use of surrogate models [3–4]: approximate models which attempt to match the behavior of the real simulation model while being computationally cheaper to evaluate. Surrogate models are constructed using data already evaluated using the real model. By occasionally using the surrogate model instead of the real model, the available budged is effectively increased. The exact method of combining the usage of the models is governed by model management strategies.

The construction of surrogate models is solved by regression analysis, which is a statistical process for estimating the relationship between the independent (input) variables and the dependent (output) variable. Symbolic regression [5] is one such process, which attempts to find a mathematical expression which best fits the given data set.

The goal of this thesis is to determine if symbolic regression models are able to better approximate the hidden objective function than classical regression models. Three algorithms are examined: quadratic models [6], genetic programming [7] and fast function extraction [5]. Each algorithm constructs the model differently and the goal is to evaluate if they can accurately model complex objective functions.

Tests will be run on the COCO benchmarking platform [8–9], which includes the choice and implementation of a well-motivated testbed, as well as tools for the post-processing and presentation of results. Differential evolution [10–11] is used as a representative evolutionary algorithm. Its surrogate-free baseline implementation is compared against four surrogate-assisted variants: the three symbolic regression algorithms and the random forest [12] algorithm.

The theoretical background and the characteristics of the algorithms used are described in chapters 2–4. More detailed description of the benchmarking functions used in experiments is given in chapter 5. The next chapter details the rationale behind the choice of algorithm control parameters, while empirical results are presented and discussed in chapter 7.

Chapter 2
Evolutionary algorithms

In the field of artificial intelligence, evolutionary algorithms represent a broad class of algorithms used for search, optimization and machine learning [1]. These algorithms are loosely inspired by Darwinian evolutionary processes in nature, and use operators such as *selection*, *mutation*, *recombination* and *reproduction* to generate solutions to a given problem. The terminology used is also inspired by biology: candidate solutions are termed *individuals* and a set of individuals is called a *population*; subsequent populations are *generations*. Each individual is evaluated using a *fitness function*. In general, an evolutionary algorithm contains the following steps:

- Generate an initial population
- Evaluate each individual using the fitness function
- Repeat until some stopping criteria is reached (number of generations, time, average fitness...):
  - select some individuals (the *parents*)
  - apply recombination and mutation operators on them produce new individuals (the *offspring*)
  - calculate the fitness of the offspring
  - combine the parents and the offspring to produce a new generation

Since there are a myriad of ways to implement these operators, many algorithms have been proposed since the field was developed in the 1960's. Historically, there have been two main paradigms: genetic algorithms and evolution strategies. Genetic algorithms represent individuals using simple data structures such as strings or integers, and emphasize recombination of individuals. Evolution strategies represent individuals as real-valued vectors and emphasize mutation over recombination. In practice, modern evolutionary algorithms are influenced by both fields and cannot be easily classified into either [1].

Evolutionary algorithms are used in solving numerical, real-valued ”black-box” optimization problems [2]. In a black-box setup, the goal is to optimize (minimize or maximize) an unknown objective function which exposes only one interface: the black-box can be queried to evaluate the value of the function at a vector x. The goal is to find the best possible value within given limits, such as the total number of function evaluations available. Since evolutionary algorithms do not assume anything about the evaluated function, they are a good fit for solving this class of problems.

2.1 Differential evolution

The differential evolution algorithm was introduced by Storn and Price in 1995. Differential evolution is used for optimizing multidimensional real-valued functions, but does not use the gradient of the function being optimized, which makes it well suited to black-box optimization problems [10].

The algorithm follows the same general steps as other evolutionary algorithms: the initial population of solutions is generated randomly using a uniform probability distribution within the search space.[1] Then, for each individual in the population, a new parameter vector is generated by adding the weighted difference of two randomly chosen population members to a third. If this new trial parameter vector has a higher fitness than the initially chosen mutant vector, it replaces it in the next generation. This perturbation process is repeated until the given termination criterion is reached.

To increase the diversity of the perturbed parameter vectors, crossover (recombination) is introduced. The trial vector is combined with the mutant vector in a randomly chosen number of parameters, while ensuring at least one parameter is used from the mutant vector. This increased diversity helps the algorithm avoid local optima [11]. The algorithm is conceptually relatively simple, as illustrated by the following pseudocode:

```
randomly generate the initial population
while generation < n_generations:
    for each vector x in the population:
        # mutation
        randomly choose vectors t1, t2, t3, such that t1 != t2 != t3 != x
        set v = t1 + F * (t2 - t3)

        # crossover
        set u, such that:
            for each parameter i, generate a random number ri:
                if ri < C:
                    u[i] = v[i]
                else:
                    u[i] = x[i]
            # at least one parameter from the mutant vector
            choose random index j
            set u[j] = v[j]

        # selection
        if fitness(u) > fitness(x):
            replace x with u in the population
```

The algorithm itself requires only three control parameters to be chosen: the size of the population $NP \geq 4$, the differential weight $F \in [0, 2]$ and the crossover probability $C \in [0, 1]$. The size of the population cannot be less than four due to how the mutated vector is generated. Storn and Price initially recommended values for NP between $5 * D$ and $10 * D$, where D is the number of dimensions (parameters); the suggested value for F is 0.5 and for C, 0.1. Further research on selecting appropriate parameters was done by Zaharie in 2002 [13].

Vesterstrøm and Thomsen [2] have concluded that differential evolution generally outperform other evolutionary algorithms and particle swarm optimization algorithms. When ran on a test suite of 34 widely used black-box optimization benchmarks, differential evolution had found the best fitness values for most functions. The authors did note that the algorithm had difficulties with noisy functions.

[1] Therefore, the bounds of the search space need to be known.

Chapter 3
Surrogate models

When optimizing practical problems, the outcome of interest usually cannot be easily measured or it is prohibitively expensive to do so. A commonly used motivational example is aerodynamic wing design, where a full 3D simulation takes considerable computing resources, while less complete simulations achieve better performance at the expense of accuracy. In such cases, it is possible to construct *surrogate models* (also known as metamodels) which attempt to simulate the behavior of the real model as closely as possible. They are constructed using data already evaluated with the real fitness function, making no assumptions on the inner working of the simulation. When used for optimization, evolutionary algorithms rely on performing many evaluations of the fitness function. If these evaluations are computationally expensive, surrogate models can be used to improve algorithm performance [3–4].

Surrogate models alone cannot be used to solve an optimization task; per definition, they can only make use of existing data to provide an approximate model. As such, when used in evolutionary algorithms, they should be combined with evaluations of the real fitness function [4]. For example, surrogate models which introduce false optima can unfavorably impact the convergence properties of the algorithm [14]. How real fitness evaluations are combined with surrogate model evaluations is defined by *model management* strategies.

According to [4], there are many different model management strategies. They can be divided into individual-based, generation-based and population-based. In individual-based strategies, the real fitness function is used for some of the individuals in each generation. Conversely, in generation-based strategies, whole generations are evaluated by either the real or surrogate fitness function exclusively. Population-based strategies co-evolve multiple populations, each with their own surrogate model, while allowing migration between populations. A related method is the pre-selection strategy [15]: assuming the population size is λ, $\lambda_1 > \lambda$ individuals are first evaluated using the surrogate fitness and the best λ individuals are then evaluated with the real fitness function. In the best strategy method, $\lambda^* < \lambda$ best individuals are evaluated using the real fitness function; therefore, it is possible that the fitness of some of the chosen parents is based on just the surrogate fitness function.

3.1 Managing a single surrogate model

In individual-based model management strategies, the key question is how to choose which individuals should be re-evaluated using the real fitness function [4]. With the assumption that these evaluations are computationally expensive, they should be used as rarely as possible, while still allowing the algorithm to find the global optimum. One approach is to re-evaluate those individuals with the highest surrogate fitness values; alternatively, the population can be clustered, with the best (or mean) individual of each cluster being re-evaluated.

Another possibility is to re-evaluate individuals with the most uncertainty. This is motivated by two arguments: a large degree of uncertainty implies that the fitness function landscape around the given value is not well explored; the re-evaluation of these points may be the most efficient way to improve the quality of the surrogate model. However, this uncertainty also needs to be estimated. This is most often done using Gaussian process regression [4] (also known as Kriging), which provide an estimate of the fitness as well as its variance. This additional information comes at the expense of higher computational costs.

A further aspect to consider is how often the surrogate model should be used. To accomplish this, surrogate model quality needs to be measured. The most straightforward metric is calculating the approximation error after re-evaluation, however, large approximation errors do not mean the quality of the surrogate model is low. For example, if the real fitness function is $y = sin(x)$ and the surrogate model $\hat{y} = sin(x) + 5$, the error is large even though an evolutionary algorithm searching using the surrogate model only would find the correct solution. Given a set of input values, it is sufficient for the surrogate model to correctly identify the value closest to the optimum; the surrogate models' prediction of the outcome is not important if this estimate is to be re-evaluated using the real fitness function.

Therefore, a more informed measure of model quality might be to the correlation between the surrogate fitness estimations and the real function evaluations. This can be done using well-known methods such as the Pearson's r, Spearman's rho or Kendall's tau correlation coefficients. As Pearson's r is a linear correlation coefficient, the other two metrics, which calculate rank correlation, might be a closer fit to the requirements placed on surrogate model quality. More complex metrics have also been proposed, for example in [16]. By using one of these metrics, it is possible to adapt surrogate model usage frequency based on model quality, thereby increasing the overall efficiency of the evolutionary algorithm.

Although approximation error is not a key metric of model quality, improving it should also be considered. One of the main obstacles to do so is the high dimensionality of the design space.[1] This issue can be mitigated by only using local surrogate models constructed with data in the vicinity of the point of interest [3] or by employing regularized learning to prevent overfitting[2] [14].

3.2 Managing multiple surrogate models

Generation-based and especially population-based model management strategies can benefit from using multiple surrogate models. There are two main approaches to do so: homogenous and heterogeneous surrogate models. Homogenous surrogate models are of the same type and fidelity. Constructing an ensemble of such models confers two main advantages: estimation quality is improved compared to each individual model, and the variance of the predicted values can be used to identify large prediction errors which helps avoid false optima [4].

Heterogeneous models are models with differences in fidelity and potentially type. For example, models might perform better at a certain region of the search space. Furthermore, such approaches can be used in conserving computing resources by constructing coarser models at the start of the search and improving fidelity as search progresses and more data is available; low-fidelity models can also be constructed from incomplete

[1] Also known as the curse of dimensionality.
[2] Overfitting means that the model describes noise instead of the underlying relationship.

data. Population-based strategies can employ different models in each sub-population. In [17], the authors identify that uncertainty in surrogate models can have both positive and negative consequences; they named these effects the *bless* and *curse of uncertainty*, respectively. As such, the proposed algorithm includes both lower and higher fidelity models. Empirically, the algorithm even outperformed a version using the real fitness function as a surrogate model.

Using multiple surrogate models is also more easily parallelizable, which can confer further benefits in computing resource utilization and efficiency.

3.3 Other properties

There are other desirable properties of surrogate models. When used in evolutionary algorithms, new data is gradually made available and re-learning the model can be time-consuming. It can also be assumed that the newer data will be more accurate or closer to the sought optimum, so it might be prudent to weight these newer samples more when generating surrogate models. These issues might be mitigated by using some form of incremental (online) learning techniques [18]. Such properties would also allow surrogate models to be used in dynamic optimization problems, where the goal is to optimize a moving optimum, or in robust optimization, where the secondary optimization goal is to minimize sensitivity to small changes in the objective function.

3.4 k-Nearest Neighbors

One of the simplest machine learning algorithms is the k-Nearest Neighbor algorithm [19]. Given a set of training vectors, the algorithm estimates a feature of the input vector by finding k of its ”nearest neighbors” and deciding based on their properties. When used for classification, the outcome is the class membership of the input vector, decided by a majority vote amongst the neighbors; if used for regression, the output is a property value, calculated as the mean of the nearest neighbors. This variant of the algorithm can be used as a surrogate model during optimization tasks.

Other than choosing the parameter k, the choice of the distance metric used can have a major effect on algorithm performance. Commonly used metrics are the L_2 (Euclidean distance) and the L_1 norms (Manhattan distance); however, adapting the distance norm to the particular problem being solved, for example by utilizing any statistical regularities in the training data, can significantly improve performance [20]. When used for regression tasks, it can also prove beneficial to weight the contribution of the neighbors based on the computed distance, so that nearer neighbors contribute more to the estimated value. One method is to weight the vectors by the inverse of their distance, although any arbitrary kernel function[1] [19] can be used.

3.5 Random forest

Decision tree learning is another commonly used method in data mining and machine learning. The algorithm creates models which predict the value of the dependent variable based on the independent variables. The model is represented as a decision tree, where each inner node represents one of the independent variables and has outgoing

[1] Kernel functions $K()$ of the distance d are functions which monotonically decrease with the growing absolute value of d and have a global maximum at $d = 0$.

edges for each possible variable value. The leaf nodes contain the predicted dependent variable value, represented by independent variable values found on the path from the root node to the given leaf node. Trees are constructed by recursively splitting the training vector set into subsets based on an attribute value test. This greedy algorithm was named *top-down induction of decision trees* by Quinlan in 1986 [21]. Decision trees can be used for both classification problems, where the terminal nodes take a finite set of values (classes), and regression problems, where the dependent variable takes real values. As such, it is possible to use regression decision trees as a surrogate model in optimization problems.

Decision trees can easily overfit to the given input set. However, combining multiple decision trees usually produces a model with higher accuracy than each individual decision tree [22]. In the machine learning community, such algorithms are called ensemble learning algorithms. The ensemble variant of decision trees, *random forests*, were first proposed by Breiman [12]. Random forests use bootstrap aggregating (also known as bagging) to generate training sets for each tree. The bagging algorithm, when given a training set D of size n, generates new training sets D_i of size n' by sampling from D uniformly with replacement. This method leads to better model performance as it decreases the variance and does not increase the bias of the final model. Random forests also utilize "attribute bagging" — a random subset of all independent variables is used at each candidate split, instead of all of them. In this way, the potential impact of one or a few strongly correlated variables is diminished, ensuring that the individual trees in the forest do not become correlated.

Another advantage of random forests is that they do not need many parameters to be set [23]: essentially, only the number of trees in the forest and the number of variables to consider at each node. The algorithm is also somewhat insensitive to the parameter values and results of many random forests will generally converge if enough trees are generated. As such, random forests are suitable for use in black-box optimization problems.

Chapter 4
Symbolic regression

Symbolic regression is a process used to find the mathematical expression which best fits the given dataset. It is a type of regression analysis, which is a statistical process used to estimate the relationship between variables. In regression analysis, the focus is on finding the relationship between a dependent variable and one or more independent variables; specifically, how the value of the dependent variable changes when varying any of the independent variables. In symbolic regression, the goal is to construct a mathematical expression containing the independent variables which best describes the dependent variable. Symbolic regression is usually implemented using evolutionary algorithms, most commonly genetic programming.

Formally, given **X** and **y**, a set of $\{x_i, y_i\}, i = 1..N$ data samples where x_i is the i-th n-dimensional point and y_i the corresponding output value, determine the symbolic model m mapping the n-dimensional input points to an output value $\hat{y}$: $\hat{y} = m(x)$, which minimizes a given error metric $err()$ [5].

Theoretically, there is an infinite number of models perfectly describing the given data as the search space of all possible mathematical expressions is infinite. To make search feasible, the set of mathematical functions used in constructing potential models must be finite and reasonably small — this requires at least some knowledge about the underlying data. Commonly used functions include power, exponential, logarithmic and hyperbolic functions, with specific algorithms using their own rules on the allowed degree of interaction between them.

The large search space can be considered an advantage in the context of evolutionary algorithms, since they can leverage the higher diversity of the produced models to generate better results. For many practical applications, it is also important for the predicted model to be ”understandable” from a human perspective.

Therefore, the symbolic regression problem is usually expanded to include a trade-off between model complexity and accuracy. Various complexity metrics have been explored, such as the number of base functions [5] or the order of non-linearity [24]. The formal definition of symbolic regression can therefore be expanded: the goal is to determine the set of symbolic models $M = m_1, m_2, ...$ that provide the Pareto-optimal trade-off minimizing both model complexity and error, using *some* chosen metrics.

In general, symbolic regression requires no *a priori* information about the relationships between the inputs and outputs; it discovers both the model structure and its parameters. This feature is further accentuated by returning the Pareto frontier, which avoids having to *a priori* specify the desired trade-off between complexity and accuracy.

4.1 Polynomial regression

Linear regression is one of the most well known forms of regression analysis. The relationship between the independent and dependent variables is modeled using a linear combination of the predictor functions. In classical linear regression, the predictor variables are the independent variables themselves. However, it is also possible to arbitrarily transform each predictor variable or groups of them and use the results as the linear predictor function. One way to accomplish this is to use all polynomial functions (up to a given rank) of independent variables as predictor variables. This form of linear regression is called polynomial regression and was first examined in 1815 [6].

Given a sufficiently high rank, a polynomial function of the independent variables will always perfectly fit the given data. This is undesirable, since such a fit will almost certainly result in overfitting the model to the training data. Additionally, estimated polynomials, especially higher-order ones, may display suboptimal non-local properties [25]; where the fit at a certain value depends strongly on data points far from the chosen value. Therefore, it is advantageous to limit the polynomial rank to two.

In this case, the set of predictor values contains the pairwise product of each independent variable and the variables themselves. Let the model fitted by linear regression using the least squares method be called a *quadratic model*. The quadratic model can be considered a simple symbolic model: it maps an n-dimensional input to an output value while minimizing the sum of the squares of the errors. As only one model can be generated for the given data, there is no need to examine its complexity.[1]

Although conceptually simple, these models have several advantages: the algorithm is deterministic, the same data will always produce the same model; the computational demands of linear regression are not high, so models can be constructed quickly; there are no parameters to tune, other than the polynomial rank which is set to two. However, such models could also easily overfit the data, might still exhibit undesirable non-local properties, may be susceptible to noisy data and cannot accurately model even some simple function classes, for example, trigonometric functions.

4.2 Genetic programming

Genetic programming is an evolutionary algorithm which, instead of evolving solutions to the given problem, tries to find a computer program which can compute the requested solution [7]. Individuals are computer programs, whose output can be compared to some ideal, and the numerical representation of the difference takes the role of a fitness function. In this context, recombination represents the random combining of parent programs to create offspring and mutation randomly changes a selected part of a program. To make these operations easier, these programs are typically represented as tree structures. Genetic programming can also be considered as an application of evolutionary algorithms on computer programs themselves.

These features make genetic programming suitable for symbolic regression. The ”computer programs” to be optimized are mathematical formulas, which are very suitable for representation in tree structures. Tree nodes contain mathematical functions and leaf nodes contain either one of the independent variables or a constant. Recombination can be performed by exchanging random subtrees of the parents, and mutation can change random nodes (while maintaining arity if changing tree nodes). Various other mutation operations have been proposed, some of which are shown in, for example, [26–27]. Evaluating a formula produces the predicted value, which can be compared to the expected value using a variety of metrics, which allows for efficient selection.

A major disadvantage of genetic programming is the presence of many tunable parameters. Not only do appropriate recombination and mutation operators need to be chosen, but also the probabilities of each of them occurring. Populations may suffer

[1] Alternatively, complexity may be considered constant.

from *bloat* — the increase of program size without any corresponding increase in fitness [7]. This can be mitigated by some mutation operators or by penalizing the program size using the *parsimony pressure* [28], another tunable parameter. There are also several initialization methods. Care needs to be taken when choosing the set of allowed functions, so division by zero and other numerical errors do not occur.[1] Population size, selection criteria and the termination condition must also be defined. Attempting to control these parameters or other features of the generated solutions usually results in the need for more parameters. Finally, it is impossible to determine how changing any particular parameter will change the final solution, so the final choice usually boils down to trial-and-error, intuition or adopting proposed default values.

Nevertheless, genetic programming remains a good tool for symbolic regression. As many potential models are generated, there is a high diversity of solutions. Many complexity measures can be calculated, such as the order of nonlinearity [24], the length or depth of the model tree and others. Therefore, a high quality Pareto frontier can be returned.

4.3 Fast Function Extraction

Fast Function Extraction (FFX) [5] is a non-evolutionary, deterministic algorithm for symbolic regression. FFX generates a large set of linear and nonlinear basis function and then uses pathwise regularized learning to find coefficient values for the basis functions in mapping to the dependent variable. Regularized learning is an answer to the tendency of least squares learning to overfit the data by introducing minimization terms that depend on the L_1 or L_2 norms of the coefficients. In FFX, the elastic net [29] formulation of regularized learning is used:

$$a^* = minimize||\mathbf{y} - \mathbf{X} * a||^2 + \lambda_2||a||^2 + \lambda_1||a||_1$$

where $\mathbf{y}$ is the dependent variable, $\mathbf{X}$ are the independent variables and λ_1 and λ_2 are parameters. The above equation is calculated across a set of possible λ values. Due to the L_1 part of regularization, the coefficients a_i take nonzero values one at a time. Finally, a non-dominated Pareto frontier of the found results is returned, trading off the number of bases with the normalized mean square error.

Like in genetic programming, the set of used basis functions defines the expressivity of the resulting models. FFX generates the set of basis functions as the union of three sets: the first set contains each independent variable raised to each allowed exponent; the second applies one of the allowed unary mathematical operators to each of the function from the first set; and the third set contains "interacting-variable" bases — the pairwise product of each basis function from the first two sets. The third set does not include products of two basis functions form the second set,[2] as these basis functions are deemed too complex. By default, the set of allowed exponents is $\{-0.5, -1, 0.5, 1\}$ and allowed operators are $log_{10}(x), abs(x), max(0, x - thr), max(0, thr - x)$, where thr is a constant; by default, five such constants are generated for each variable.[3] Basis function whose evaluation results in numerical errors are not allowed.

[1] This can be solved by using "protected" functions; for example, the protected square root $psqrt(x) = \sqrt{|x|}$ is defined for all real numbers.
[2] Expressions of the form $op() * op()$.
[3] The last two classes of operators are called *hinge* functions. They are used to "turn off" a region of the input space.

Other than choosing the set of allowed exponents and operators, there are no parameters to be tuned. Implementations of FFX introduce various stopping criteria (such as stopping regularized learning early if the error metric is not improving), but none of these are essential and only serve to reduce running time. The maximum number of basis functions can also be set, which can drastically improve running time and assure only less complex, "human-understandable" models are produced.

The FFX algorithm has two main advantages: it is fast, as the learning speed of regularized learning is comparable to the least squares method, and it is deterministic. Due to the use of regularized learning, it can learn a model even with few data samples given and can learn thousands or more coefficients. The generated models have progressively increasing complexity, making the resulting Pareto frontier very fine-grained. The crucial disadvantage is that given the same set of allowed functions, FFX produces less complex models than genetic programming and other methods, due to the artificially limited maximal complexity of each predictor function. Additionally, FFX cannot learn the inner parameters of a function — functions like $log_{10}(2x)$ or $log_{10}(x+2)$ can never be generated. While the first limitation could be trivially removed at the expense of running time, modifying inner function parameters would require further research.

Chapter 5
Testing suite

Quantifying and comparing the performance of optimization algorithms on black-box optimization problems is an important aspect of research. However, accomplishing this task in a statistically rigorous and sound way can be difficult and tedious. To aid in this process, the COCO (COmparing Continuous Optimisers) platform was created for the Black-Box-Optimization-Benchmarking workshop at the GECCO conference in 2009 [8]. This platform has been used continuously since its creation, with an updated version scheduled for release in 2016. The COCO platform includes the choice and implementation of a well-motivated benchmark function testbed, the design of an experimental setup and routines for post-processing and presentation of results. The platform is compatible with Python, C, Java, MATLAB and R.

5.1 Function definitions

According to the authors of COCO [8], the intention behind the selection of benchmark problems was to represent typical difficulties occurring in continuous domain search. They have attempted to use comprehensible functions, so that algorithm behavior is understood in the topological context. They hope this will allow easier analysis of algorithm deficiencies and eventually lead to the improvement of algorithms.

All defined benchmark functions are scalable with dimension and have an artificially chosen optimal function value. Most functions are shifted in x-space — they have no specific value for their optimal solution. It is thus possible to generate different instances for each function; benchmarking is done across a set of uniquely selected instances. The global optimum for all functions is in $[-5, 5]^D$; the goal is to minimize the given functions. The following sections will list the 24 used noiseless benchmark functions in order, along with their interesting properties. Full function definitions are available in [8], while figures are available in [30].

5.1.1 Separable functions

The *sphere function* is unimodal, highly symmetric and rotationally invariant. It is presumed to be the easiest continuous domain search problem.

The *ellipsoid function* is a globally quadratic, ill-conditioned function with smooth local regularities. It is unimodal with a condition number of about 10^6.

The *Rastrigin function* is a multimodal function with relatively regularly spaced optima commonly used in benchmarking. In COCO it is further transformed to alleviate the symmetry and regularity of the original function. It has roughly 10^D local optima and a condition number of about 10.

The *Buche-Rastrigin function* is highly multimodal with a structured, but highly asymmetric, placement of optima. It was constructed as a deceptive function for symmetrically distributed search operators and has about 10^D local optima with a condition number of around 10.

Linear slope is a purely linear function testing whether the search can go outside the initial convex hull of solutions right into the domain boundary, which contains the optimal $\mathbf{x}$ value.

5.1.2 Functions with low or moderate conditioning

The *attractive sector function* is highly asymmetric, where only a "hypercone" with a volume of roughly $1/2^D$ yields low function values. The function is unimodal — the tip of the cone contains the optimum.

The *step ellipsoidal function* consists of many plateaus of different sizes. The gradient is zero everywhere apart from a small area close to the global optimum. The condition number is about 100.

The *original Rosenbrock function* is another commonly used benchmarking function. It features a local optimum with an attraction volume of roughly a quarter of the search space, while the global optimum can only be reached by "following" long, narrow valley.

The *rotated Rosenbrock function* is a rotated version of the original Rosenbrock function.

5.1.3 Functions with high conditioning and unimodal

The *ellipsoidal function* is globally quadratic, ill-conditioned, with smooth local irregularities. It is the non-separable counterpart to the ellipsoid function and has a condition number of about 10^6.

The *discus function* is globally quadratic with local irregularities. It has a single direction in the search space which is a thousand times more sensitive than all others. Its condition number is about 10^6.

The *bent cigar function* has a smooth, but very narrow ridge which needs to be "followed". The overall shape differs remarkably from being quadratic; the condition number is about 10^6.

The *sharp ridge function* has a gradient which is independent of the distance to the ridge. The ridge needs to be "followed" to the optimum and approaching it is initially effective, but becomes ineffective close to the ridge. The necessary change in search behavior is predicted to be difficult to diagnose.

The *different powers function* has continuously worsening sensitivity relations of the variables when approaching the optimum.

5.1.4 Multi-modal functions with adequate global structure

The *Rastrigin function* is a non-separable, less regular counterpart to the already defined Rastrigin function.

The *Weierstrass function* is highly rugged with a moderately repetitive landscape and a non-unique global optimum.

Schaffer's F7 function is highly multimodal with varying frequency and amplitude of the modulation and a low condition number.

The *'moderately ill-conditioned' Schaffers F7 function* is a counterpart to the above function with a condition number of about 1000.

The *composite Griewank-Rosenbrock function F8F2* is a highly multimodal counterpart to the Rosenbrock function.

5.1.5 Multi-modal functions with weak global structure

The *Schwefel function* has a penalized search area, with the most prominent optima located comparatively close to its corners.

Gallagher's Gaussian 101-me peaks function consists of 101 optima with randomly chosen and unrelated positions and heights. The condition number around the global optimum is about 30.

Gallagher's Gaussian 21-hi peaks function consists of 21 optima with randomly chosen and unrelated positions and heights. Compared to the previous function, it has a much higher condition number at about 1000.

The *Katsuura function* is highly rugged and highly repetitive with more than 10^D global optima.

The *Lunacek bi-Rastrigin function* is a highly multimodal function with two "funnels" and a highly multimodal function within them. This function was constructed to be deceptive for evolutionary algorithms with a large population size.

5.2 Noisy function testbed

The COCO platform also provides a benchmarking suite of noisy function [9]. Such functions are often more challenging to optimize, while simultaneously more closely approximating real-world experiments, where noise can be added as a result of measuring errors. Three different noise models are used: Gaussian, uniform and Cauchy. The rest of this section will note the important properties of benchmark functions and noise models. Full definitions are available in [9] and a graphical presentation is given in [31].

The Gaussian noise model is defined as:

$$f_{GN}(f, \beta) = f \times exp(\beta N(0,1))$$

This model is scale invariant, with the noise strength being controlled by β. As the distribution of the noise is log-normal, no negative values can be sampled.

The uniform noise model is more severe and is defined as:

$$f_{UN}(f, \alpha, \beta) = f \times U(0,1)^{\beta} max\left(1, \left(\frac{10^9}{f+\epsilon}\right)^{\alpha U(0,1)}\right)$$

This model is not scale invariant. The noise strength increases with decreasing value of f, making the noise more severe when approaching the optimum.

The third model is the Cauchy noise model, which is defined as:

$$f_{CN}(f, \alpha, p) = f + \alpha max\left(0, 1000 + \mathbb{I}_{\{U(0,1)<p\}} \frac{N(0,1)}{|N(0,1)| + \epsilon}\right)$$

In the Cauchy noise model, only a comparatively small percentage of function values is affected by noise. Among these values large outliers can occur, which cannot be easily detected as they stem from a continuous distribution.

The testbed is then divided into three groups of functions based on the severity of noise added. Each function is sampled using all three noise models, with appropriately chosen constants. The sphere and Rosenbrock functions are tested with moderate noise. The second group adds severe noise to the sphere, Rosenbrock, step-ellipsoid, ellipsoid and different powers functions. The third group has three highly multi-modal functions with severe noise: Schaffer's F7, the composite Griewant-Rosenbrock and the globally rotated Gallagher's Gaussian 101-me peaks functions.

5.3 Experimental setup

The algorithm under consideration is run on the defined testbed of functions [32]. Each function is ran for 15 specified instances for each search space dimension: 2, 3, 5, 10, 20.[1] In each run, the algorithm should reach a target function value $f_{target} = f_{opt} + \Delta f$, where f_{opt} is the optimal function value and Δf the precision to reach. The final, smallest precision to reach is $\Delta f = 10^{-8}$, although larger values are also considered.

The algorithm is given some information at initialization: the dimensionality D of the search space, the search domain $[-5, 5]^D$, whether the testbed has noisy or noise-free functions, the final target precision $\Delta f = 10^{-8}$ and the final target function value. This value is only provided for conclusive termination of trials and should not be used otherwise. The authors discourage the use of any additional information, such as the function characteristics, as algorithm input, assuming that such information would not be available in a true black-box optimization problem. Therefore, the algorithm setting should be identical for all functions in a testbed (differences are permitted between the noisy and noise-free testbeds).

The COCO platform uses the *expected running time* (ERT) performance measure of algorithms. ERT was chosen because it is quantitative, with a ratio scale and a wide variation, well-interpretable, practically relevant and relatively simple [32]. Assuming minimization, the ERT is defined as:

$$ERT(f_{target}) = \frac{\#FEs(f_{best} \geq f_{target})}{\#succ}$$

where $\#succ$ is the number of successful trial runs and $\#FEs(f_{best} \geq f_{target})$ the number of function evaluations across all trials reaching the given f_{target}.

[1] Dimensionality 40 is also available but is considered optional.

Chapter 6
Implementation

The examined algorithms were implemented in the Python programming language [33], using the NumPy extension [34] for numerical computing. Python is widely-used high-level, general-purpose programming language with a free and open-source reference implementation commonly used in scientific computing [33]. Many libraries for use in scientific computing have been developed for Python, some of which [35–37] are used in this thesis.

This chapter discusses the implemented algorithms, the chosen parameter values and the rationale behind those choices. The crucial algorithm design aspect proposed in this thesis is the incorporation of surrogate models in the basic differential evolution algorithm.

6.1 Differential evolution

There are three major choices to be made when implementing differential evolution the algorithm control parameters, NP, F and C, the exact algorithm variant to use, and how to incorporate surrogate models in the algorithm. The population NP value suggested in [11] is between $5 * D$ and $10 * D$, where D is the dimensionality of the problem. In the implementation used in this thesis, population size was set to be $NP = 10 * D$ — this ensures the algorithm will have enough mutually different vectors to work with. The suggested value for the differential weight parameter F is $F = 0.5$, which was kept as such in this thesis. The crossover probability parameter C has a large impact on convergence speed; the initial suggested value is $C = 0.1$, while settings as high as $C = 0.9$ or $C = 1.0$ can be used if a quick solution is possible. Preliminary testing have shown that a compromise between the two extremes, i.e. $C = 0.5$, performed the best across the whole testbed. Higher values of the parameter C performed better in linear function at the expense of other functions, and vice versa for lower values.

There are several subtly different versions of differential evolution. In order to distinguish between them, the authors of [11] introduce the notation $DE/x/y/y$, where:

- x defines the selection of the vector to be mutated; allowed values are ”rand”, a randomly chosen vector, or ”best”, the vector with the best fitness value.
- y is the number of difference vectors used.
- z defines the crossover scheme used; the first variant proposed in ”bin”, where crossover vectors are chosen using independent binomial experiments.

Using this notation, the basic DE strategy can be described as $DE/rand/1/bin$. The authors described another variant as ”highly promising”, namely $DE/best/2/bin$. In this variant, the best vector is always mutated using the weighted differences between two randomly chosen vectors. In this paper, the $DE/rand/1/bin$ variant is implemented to provide a baseline result, as it is the most commonly tested variant.

6.2 Model management

The proposed model management strategy is individual-based and inspired by the pre-selection strategy. Since each individual reproduces (and potentially improves itself) every generation, the idea is to generate several candidates for each trial vector, which would be evaluated using a surrogate model. The number of generated candidates is named $kmax$. The best of these candidate vectors, as quantified by the surrogate fitness function, is then evaluated using the real fitness function. In the standard pre-selection strategy, $kmax$ candidates would be generated from each individual, and then the best NP vectors in this candidate set would be chosen for re-evaluation by the real fitness function. That way, a single individual could theoretically populate the whole next generation, which is not inline with the basic differential evolution algorithm, and is especially troublesome if the surrogate model introduces false optima. The approach proposed in this thesis avoids both of these issues, since each individual is represented exactly once in the subsequent generation, either by itself or its offspring, even though several candidate offspring individuals are generated.

Choosing an appropriate $kmax$ value is another issue to consider. Initial tests were performed with $kmax_1 = 2$ and $kmax_2 = 10$. These settings provided inconsistent results: $kmax_1$ performed better on most functions than the baseline algorithm, but was occasionally vastly outperformed by $kmax_2$, presumably on functions which were accurately identified using the surrogate model. Conversely, the $kmax_2$ setting often performed statistically worse than even the baseline algorithm, especially on lower dimensions, presumably due to low surrogate model quality. However, one of the two approaches always performed better than the baseline algorithm. Therefore, an adaptive strategy was proposed: $kmax$ would start at 2 and be raised by 1 every generation, provided the model was of sufficient quality; if the model was deemed to be low quality, $kmax$ would be decreased. Irregardless of model quality, $kmax$ would never be lower than 2 nor higher than 10.

The model quality measures explored were Pearson's r and Kendall's tau correlation coefficients. Pearson's r performed well only on linear functions and poorly otherwise, which was somewhat expected due to the linear nature of the correlation coefficient. Since only rank correlation between the real and surrogate fitness function is needed for the surrogate model to provide useful solutions, Kendall's tau is a better choice. During preliminary experiments, it was discovered that the correlation coefficients stays relatively high during the initial phases of the search and eventually drops, without ever recovering. Such behavior was surprising, as the initial expectation was that the correlation coefficient would periodically oscillate between high and low values. Considering this experimental observation, the adaptive approach was deemed inadequate: when correlation is high, the $kmax$ parameter needs too many generations to adapt, losing on the benefits of using surrogate models; once the coefficients falls, the $kmax$ value is too high for several more generations, potentially misleading the search algorithm with false optima introduced by the surrogate model. Therefore, the final proposed setting is to directly set $kmax$ based on the correlation coefficient, according to the following empirical formula:

$$kmax = max(\lfloor 10 * correlation(y, \hat{y}) \rfloor - 1, 2)$$

where y are the real fitness values and $\hat{y}$ the fitness values predicted by the surrogate model. Using this formula, the value of $kmax$ will always be between 2 and 9. By directly setting this value, the algorithm can quickly adapt to model quality, reaping the benefits of both a high and low setting, depending on the ability of the surrogate model to accurately model the underlying function.

However, it is still possible for the model to completely misinterpret the underlying function being optimized. This is considered to have happened when the calculated correlation coefficient is negative, meaning that between any two data points, the surrogate model is more likely to predict a higher fitness for the less fit individual; such behavior can vastly impact algorithm convergence. In this case, the use of surrogate models is disabled. As it was earlier noted that convergence never seems to improve after the initial pronounced fall, this disabling is never reverted and the algorithm will not use any surrogates for the rest of its run.

The last important question is which data should be used to train the surrogate model. Although it is possible to use all available data, this might not be advantageous, as the evolutionary algorithm will by itself find fitter individuals as the search progresses. The initial population in particular can be very far from the optimal solution. Including these data points in the training set of the model can lead to a fit displaying suboptimal non-local properties. Under the proposed model management strategy, the surrogate model is used to predict the most fit individual from a given set. The actual predicted fitness value is not important; this is the same reasoning used for choosing a rank correlation coefficient. Therefore, the behavior of the model near the optima is more important than its behavior far from it. As such, surrogate models are trained using the data with fitness higher than the median[1] fitness of all available data, i.e. the better half of all data. The model is generated once every generation — doing it more often would increase the computational costs significantly while providing mostly the same models in each generation. Finally, model quality is measured on the NP new data points generated while evaluating the current generation, as those are the only points available which had not been used to train the model.

6.3 Surrogate models

Several surrogate models were used to assist the differential evolution algorithm. They are listed and described in the following subsections.

6.3.1 Quadratic model

The simplest surrogate model implemented, the quadratic model, has no parameters to set. It is implemented using the scikit-learn [35] package. Ordinary linear squares regression is used to generate the coefficients once the used bases are generated.

6.3.2 Genetic programming

The gplearn [36] package provides a genetic programming implementation of a symbolic regression algorithm. This code is available under the open-source BSD 3 clause and was used in this thesis as a representative genetic programming solution. The implemented algorithm has many parameters to set, nearly all of which were left at default values.

- $population\_size = 1000$, the number of individuals in each generation, raised from the default 500.
- $generations = 20$, the number of generations to evolve, raised from the default 10.
- $tournament\_size = 20$, the number of competing programs.

[1] Using the mean value would result in using a higher percentage of all data as the search progresses.

- $stopping\_criteria = 0$, the required metric value for premature algorithm termination.
- $const\_range = (-1, 1)$, the range of constant which could be included in formulas.
- $init\_depth = (2, 6)$, when generating the initial population, individuals will randomly choose a maximal depth from this range.
- $metric$ defines the raw fitness metric used. It is set to mean square error.
- $p\_crossover = 0.9$, the probability of replacing a random subtree of the tournament winner; the subtree inserted is chosen randomly from a second tournament winner.
- $parsimony\_coefficient = 0.001$, the penalization applied to large programs so that they are less likely to be used in selection. This parameter controls program bloat.

The initial population is generated using the "ramped half and half" initialization method. Half the trees are grown by choosing random nodes from both functions and terminals, while the second half is grown by randomly selecting functions until the $init\_depth$ is reached and only then adding terminals. The first half could be of smaller depth than $init\_depth$ allows, allowing for a mix of tree shapes in the initial population.

The function set contains the protected square root, protected logarithm, absolute value, negative, inverse, maximum and minimum functions. Tournament selection is used to select the most fit individual from a randomly chosen set to enter the next generation. Three mutation operators are included, each with a probability 0.01 of being performed on the tournament winner. The subtree mutation replaces a randomly selected subtree with a randomly generated one; the hoist mutation selects a random subtree and replaces it with one of its own subtrees in an effort to control bloat; point mutation selects random nodes to be replaced by a node of the same type. In this case, the probability of changing each particular node is 0.05. In case no mutation is performed, the tournament winner passes into the next generation unchanged.

6.3.3 Fast Function Extraction

This thesis includes a completely new implementation of the FFX algorithm based on the definition in [5] and the initial implementation available at [38].[1] The reimplementation relies on the scikit-learn and SymPy [37] libraries. It has been named Extended Fast Function Extraction (EFFX), as it supports several additional features and is more extensible in general. Both SymPy and scikit-learn are available under the BSD license, making them suitable for use in scientific projects. EFFX will also be open-sourced at a later date.

EFFX relies on SymPy to represent mathematical expressions. SymPy is a computer algebra system which can be used as a Python library or standalone application. As such, EFFX supports all functions offered by SymPy whilst making it easy to implement additional functions. SymPy can also be used to simplify the resulting mathematical expressions. Internally, each basis function is represented by a SymPy expression encapsulated in the class `Base`. This class is constructed with a function and its arguments, either independent variables or other `Base` objects. The arity of the function is in no way restricted. In this way, the class can be built using arbitrarily complex sub-expressions.

All other functionality is provided by the `ModelFactory` class. The class is structured according to the algorithm steps: the `generate_bases` function generates bases from the set of allowed exponents and operators; the `pathwise_learn` function performs pathwise regularized regression and the `nondominated_filter` function returns the non-dominated set that trades off complexity and error.[2] EFFX supports three methods

[1] There are several subtle differences between the two.
[2] Several helper function are also used to improve code readability.

for regularized regression: the method used in the FFX implementation and elastic net and lasso regression implementations available in the scikit-learn package. The original FFX paper [5] also suggests the use of a ”trick” in order to support rational functions without a large increase in computing costs; this feature is not currently implemented in EFFX. The normalized mean square error is used as an error metric.

Instead of using all allowed exponents and operators to generate a single large set of possible basis functions, the author of FFX suggests using their subsets and running the core algorithm several times, with the final output being the non-dominated Pareto frontier of the results across all runs. These individual runs are termed ”approaches”. EFFX uses the same approaches as in FFX, except those which use the rational functions ”trick”.[1] Some of the approaches construct univariate bases only — bases of only one variable. If only the first power is allowed and no operators, then that approach degenerates to ordinary linear regression. The approaches used are summarized in table 6.1.

exponents	operators	note
$\{1\}$	$\emptyset$	univariate bases only
$\{0.5, 1, -0.5, -1\}$	$\emptyset$	
$\{0.5, 1, -0.5, -1\}$	$\{abs(), log_{10}(), hinge()\}$	univariate bases only
$\{1\}$	$\{hinge()\}$	
$\{0.5, 1, -0.5, -1\}$	$\{abs(), log_{10}()\}$	
$\{1\}$	$\{abs(), log_{10}()\}$	
$\{1\}$	$\{abs(), log_{10}(), hinge()\}$	

Table 6.1. The approaches used in EFFX. In each approach, the allowed set of exponents and operators is listed, and whether only univariate bases are constructed.

When $hinge()$ functions are enabled, 10 basis functions are added per variable, five each of the forms $max(0, x - thr)$ and $max(0, thr - x)$, where x is the variable and thr is a constant. The five threshold values thr used are uniformly distributed between $min(x) + 0.2 * (max(x) - min(x))$ and $min(x) + 0.8 * (max(x) - min(x))$, where $min(x)$ and $max(x)$ are the minimum and maximum values seen among the training samples for the given variable. Basis functions whose evaluation results in numerical errors are disregarded.

Once a function set is generated, regularized learning is performed on it many times with decreasing values for the constant α which multiplies the penalty terms. With decreasing penalties, more basis functions are used in the resulting expression; this feature allows the creation of a set of expressions of increasing complexity. To improve computational efficiency, several stopping criteria are introduced. If the normalized mean square error has not changed in 15 iterations to 4 significant places or if the error is under 1% or if the number of bases is higher than 250, then the algorithm is terminated. In practice, the first condition is encountered in most cases — this happens when the penalization constant α is low, so that all bases are used.

6.3.4 k-Nearest Neighbors

The scikit-learn package provides the implementation of the k-Nearest Neighbors algorithm which was used in this thesis. The number of neighbors, k, is set to five. It was estimated that higher values would decrease the quality of the model, particularly

[1] In EFFX, it is also easy to define additional approaches if needed.

during the start of the search, when few data points (neighbors) are available. Due to the model management strategy used, this could mean that the model would not be used at all. All points were weighted uniformly and the Euclidean distance metric was used to compute the nearest neighbors.

6.3.5 Random forest

The implementation provided in the scikit-learn package was also used for the Random forest algorithm. All parameters were left at their default values. Ten trees were used in the forest and all features were considered when looking for the best split. The criterion for the quality of the split in each tree was the mean square error. There is no limit for the maximum depth of the trees — nodes were expanded until none of the leaf nodes could be split further. Bootstrap aggregating was used to choose slightly different training data for each tree.

Chapter 7 Results

All experiments were run with the budget of 1000 function evaluations per dimension. The graphs shown in this chapter have all been automatically generated by COCO. There are two graph types used when comparing multiple optimizers, which provide different viewpoints on the results, as explained in the rest of this section.

In the first graph type (for instance, figure 7.2), the expected running time (ERT) in the log_{10} value of the number of function evaluations divided by the dimension is plotted against the dimension of the problem. Each algorithm is plotted with a different color; light symbols give the maximum number of function evaluations available divided by the dimension. If the algorithm ERT is above the maximum number of evaluations given, this means that the target result was not reached in all tested instances and the final value is an estimate; such values are therefore less reliable. The target function value is chosen such that the *bestGECCO2009* artificial algorithm just failed to achieve an ERT of $10 \times dim$ [32].[1] If a label is denoted with a black star, the algorithm performed statistically better than all other algorithms with p-value $p < 0.01$.

These graphs clearly show algorithm behavior in each dimension. This makes it easy to spot if the given algorithm performs poorly at higher dimensions and at which dimension this change occurs; conversely, algorithms occasionally perform relatively better at higher dimensions. Given a dimension, it is also possible to quickly compare the number of function evaluations needed to reach the given target. Since each function is shown separately, it is possible to identify the exact functions where the algorithm performs well or poorly.

The second graph type (for instance, figure 7.1 shows the bootstrapped empirical cumulative distribution of the number of target function evaluations divided by dimension for all examined functions or their subset; functions are grouped into subsets according to their class as defined in section 5.1. The targets are chosen from $10^{[-8..2]}$ such that the *bestGECCO2009* artificial algorithm had just not reached them within a given budget of $k \times dim$, with $k \in \{0.5, 1.2, 3, 10, 50\}$. The performance of the artificial algorithm is marked with "best 2009".

These graphs can show how performance of each algorithm improves with additional function evaluations, for a given dimension. As such, it is easy to see how the algorithms would perform given fewer evaluations. Algorithm behavior can be analyzed with respect to the function evaluation budget — a given algorithm could perform relatively better at lower numbers of evaluations, or perform poorly before reaching the same final value as another algorithm. Since many functions are presented at a single graph, it is possible to evaluate the overall algorithm performance or the performance on a class of functions.

Only noteworthy graphs are shown in the rest of this chapter. Full results are available in appendix B.

[1] The target algorithm is artificial in the sense that it does not represent any particular algorithm, but the theoretical best performance among all the algorithms presented at the GECCO 2009 conference.

7.1 Baseline algorithms

The baseline algorithm used for all comparisons is a differential evolution algorithm without surrogate models. Two algorithms, the k-nearest neighbor and the random forest, were used to provide a baseline example with surrogate models. Figure 7.1 shows the cumulative distribution graphs for all functions at dimensions 5 and 20. It can be seen that both algorithms perform similarly, with the random forest performing slightly better at higher dimensions.

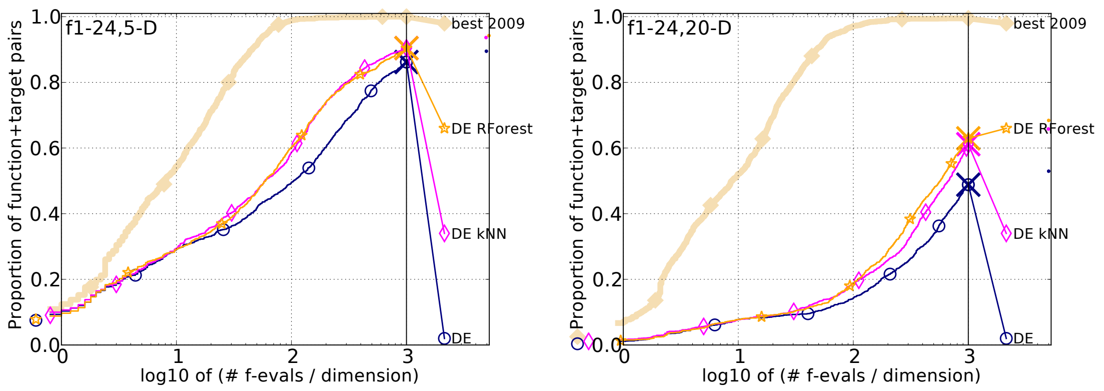


Figure 7.1. Comparison of all noise-free functions in $5D$ and $20D$ of surrogate-free differential evolution (blue), with k-Nearest neighbors (pink) and random forest (light yellow) as surrogate models.

When examining behavior across function groups, the two algorithms performed similarly in most cases, with two exceptions: the random forest algorithm performed better with separable functions, especially at higher dimensions, and the k-nearest neighbor algorithm was slightly better with weakly structured multi-modal functions, especially the two Gallagher's gaussian peaks functions. Both of these functions contain randomly chosen optima; it is thus conceivable that the "less informed" algorithm, as the k-nearest neighbor algorithm seems to be, would perform better. This effect only occurs at dimensions 10 and 20, as can be seen in figure 7.2.

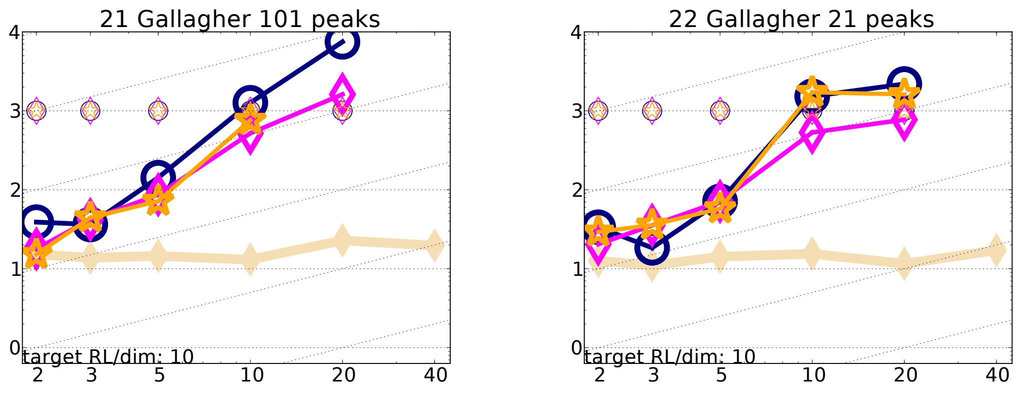


Figure 7.2. Comparison of the Gallagher's Gaussian peaks functions of surrogate-free differential evolution (blue) with k-Nearest neighbors (pink) and random forest (yellow) as surrogate models.

Since both algorithms perform similarly, the k-nearest neighbor algorithm will not considered in later comparisons, in an effort to make the results more readable.

7.2 Symbolic regression

Figure 7.3 shows the empirical cumulative distribution graphs for all algorithms on dimensions 3, 5, 10 and 20. It can be seen that all surrogate models improved overall performance, since the surrogate-free differential evolution performed worse in all dimensions; it is clearly dominated in dimensions 10 and 20 and only performed negligibly better than some algorithms in dimensions 3 and 5. This can be considered a success of the model management strategy used, since its main purpose is to ensure the model is used only when it is favorable to do so. Results for dimension 2 are not shown as all algorithms performed roughly the same, presumably as there were not enough function evaluations available to allow surrogate models to have a larger effect. Additionally, the lower dimension cases are not particularly useful for practical usages.

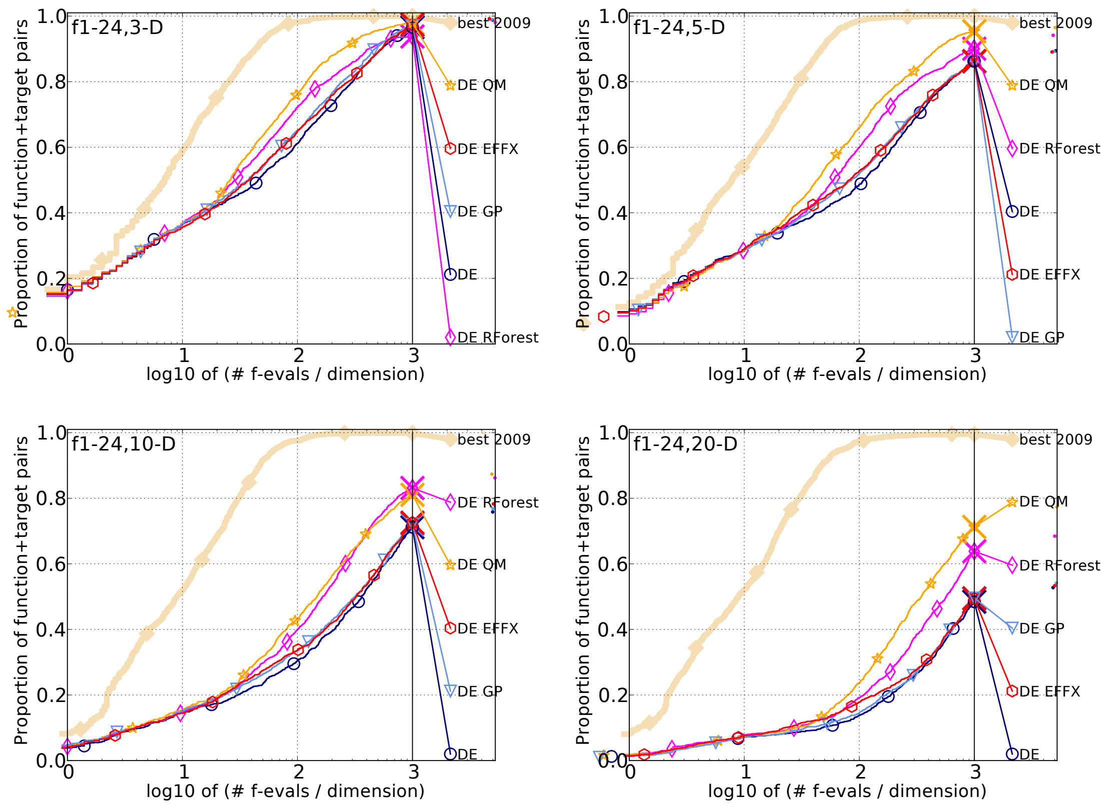


Figure 7.3. Comparison of all noise-free functions in 3, 5, 10 and 20 dimensions of surrogate-free differential evolution (blue) with random forest (pink), quadratic model (yellow), genetic programming (light blue) and EFFX (red) as surrogate models.

The algorithms shown can be divided into two groups, the quadratic model and random forest forming the first one with higher performance and the other algorithms the one with lower performance; this separation is more clearly seen at higher dimensions. This effect can again be somewhat attributed to the model management strategy: the EFFX and genetic programming models are not able to correctly model the function with given input data, and they are quickly disabled, after which search proceeds without surrogate models. Nevertheless, both algorithms improved the search process before converging to the same result as baseline differential evolution; the improvement is most noticeable between budgets of approximately 30 to 500 function evaluations per dimension.

All algorithms only provide an improvement after about 30 function evaluations per dimension, which is to be expected. Considering that the population size is ten times the dimension, each generation uses ten function evaluations per dimension. The first generation is generated randomly and the second generation has a fixed $kmax = 2$ parameter. Only in the third generation and further is the correlation coefficient considered and the $kmax$ parameter potentially raised. In other words, surrogate models are always used soon after the start of the search process and seem to be helpful regardless of the specific algorithm used. At some later point, the quality of the surrogate models (as measured by the rank correlation coefficient) seems to decrease and they are therefore disabled. Nevertheless, the differential evolution algorithm is able to leverage the better individuals generated at the start to continue producing better results than the baseline algorithm. Even if the final target value converged with the given budget, it is clear that surrogate models help with at least some budgets.

Between the two better performing algorithms, the quadratic model dominates the random forest model in most cases and both outperform the other symbolic regression algorithms. Although it might be expected that the quadratic model will overfit to the training data, this does not seem to be the case; in fact, it must fit the data quite well as it manages to substantially help the differential evolution algorithm find better solutions faster.

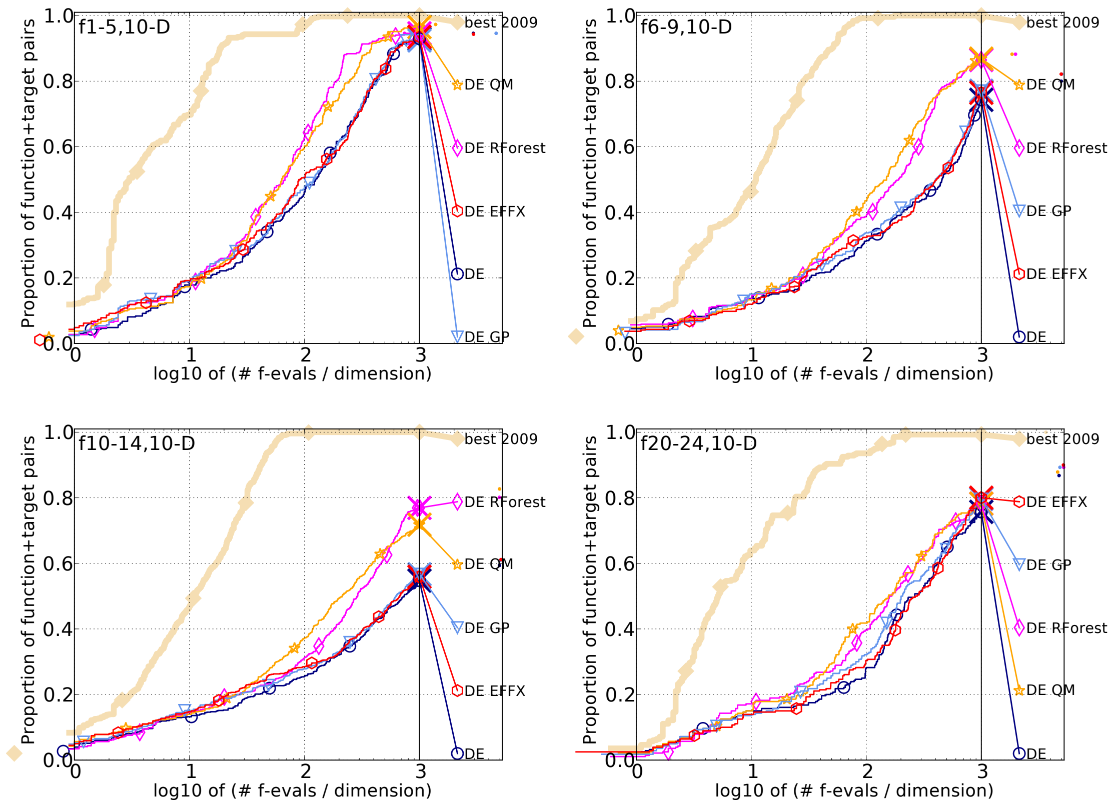


Figure 7.4. Comparison of separable, moderate, ill-conditioned and weakly structured multi-modal functions in 10 dimensions of surrogate-free differential evolution (blue) with random forest (pink), quadratic model (yellow), genetic programming (light blue) and EFFX (red) as surrogate models.

Interestingly, in dimension 10 the random forest and quadratic model algorithms are more closely matching than in any other dimension tested. The random forest surrogate model produces relatively better results in later evaluations, overcoming the earlier "lead" of the quadratic model. Comparison across the function groups is shown in figures 7.4 and 7.5, while the aggregate performance across all functions is shown in figure 7.3.

In 10 dimensions, the random forest algorithm performs better on separable functions than the quadratic algorithm. This is especially surprising, because it is expected that the quadratic model will be able to construct the exact symbolic model of the target functions, and seems to do so in all other dimensions. On moderate and weakly structured multi-modal functions, the quadratic model offers only somewhat better performance, without producing a better optimum by the end of optimization. On ill-conditioned functions it performs better for most of the given budget, before ultimately being outperformed by the random forest model. All of this behavior, except the behavior on adequately structured multi-modal functions which will be discussed later, is not seen in any other dimension.

When compared across all dimensions, the random forest model performs relatively better on multi-modal functions. Figure 7.5 shows the empirical cumulative distribution graphs for functions $f_{15} - f_{19}$. In the 10 dimensional case, both algorithms perform similarly at first, but the random forest model is clearly better at higher function evaluation budgets. With 20 dimensions, the quadratic model is better at times although it still produces a worse final target value. In other function groups, the quadratic model performs consistently better, with larger differences occurring at higher dimensions, except dimension 10 as noted above.

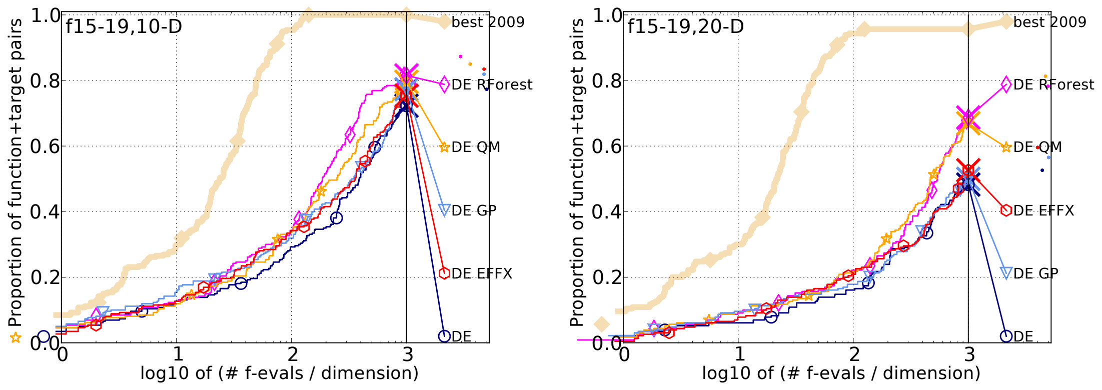


Figure 7.5. Comparison of performance on multi-modal functions with adequate structure at dimensions 10 and 20 of surrogate-free differential evolution (blue) with random forest (pink), quadratic model (yellow), genetic programming (light blue) and EFFX (red) as surrogate models.

Comparing results for each function separately, with the data points at 0.5, 1.2, 3, 10 and 50 function evaluations per dimension, the quadratic algorithm performs the best in most cases (as seen in figure 7.3). However, other algorithms also occasionally had the best results on particular functions (and often only at one dimension). The random forest algorithm performed better on at least one data point in functions $f_1 - f_5, f_9, f_{17}, f_{18}, f_{21}, f_{22}, f_{24}$ in dimension 5, but only in functions f_{17} and f_{18} in the 20-dimensional case. The EFFX and genetic programming models were also the most performant in a few specific cases, mostly early during the search process and more often at lower dimensions.

When considering only the final target value reached, the quadratic model found a statistically better result in at least one dimension for functions $f_1, f_2, f_4, f_7 - f_9, f_{12} - f_{15}$. No other algorithm produces a statistically better result in any of the functions.

7.3 Noisy functions

Although a higher variety of noise-free functions is tested, the noisy functions testbed is arguably more important, as it more closely resembles practical applications of optimization problems, where measurement error cannot really be avoided. Figure 7.6 shows the empirical cumulative distribution graphs for all algorithms on dimensions 3, 5, 10 and 20,[1] on the noisy testbed. A parallel can be drawn with the results on the noise-free functions: none of the surrogate models performed worse than the baseline differential evolution, and the random forest and quadratic model algorithms were noticeably better than the other algorithms. However, there are various differences.

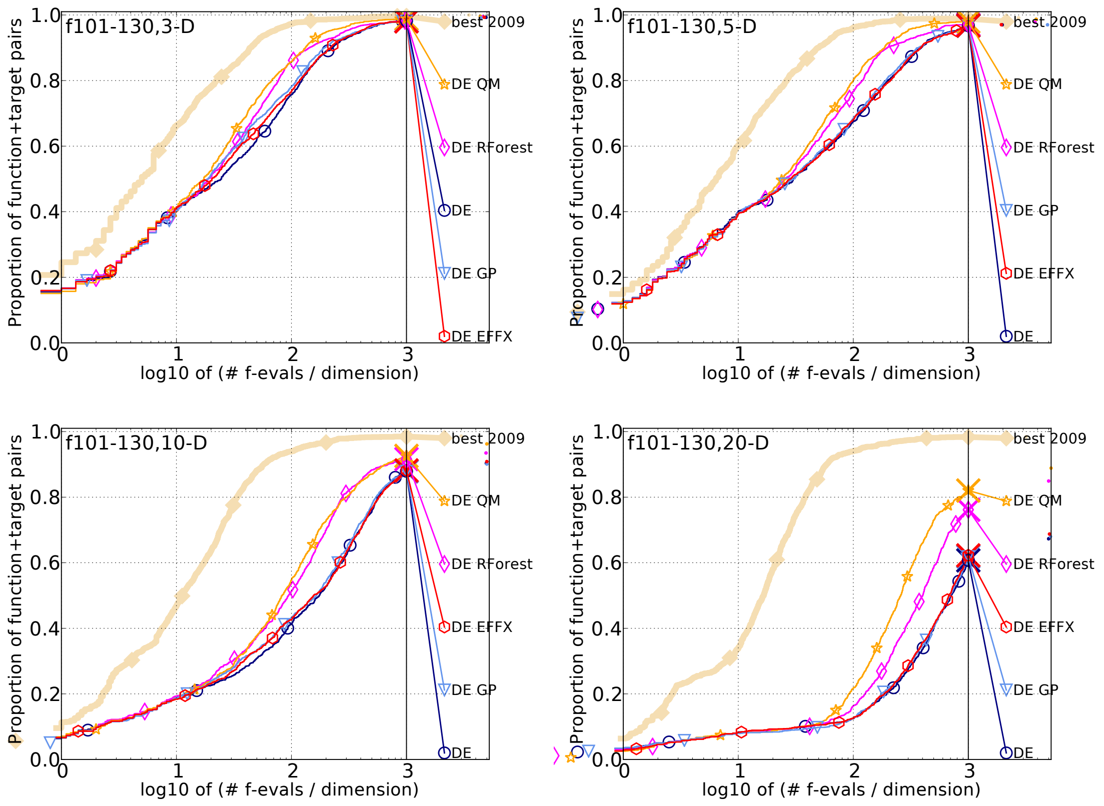


Figure 7.6. Comparison of all noisy functions in 3, 5, 10 and 20 dimensions of surrogate-free differential evolution (blue) with random forest (pink), quadratic model (yellow), genetic programming (light blue) and EFFX (red) as surrogate models.

The EFFX and genetic programming algorithms performed rather poorly. Other than the three dimensional case, where a small performance improvement can be noticed, it seems that these algorithms are not able to produce models which are accepted by the model management strategy. Therefore, they are simply disabled and perform roughly the same as the baseline algorithm. As already noted, the random forest and quadratic

[1] Results for dimension 2 are again omitted, due to their dubious practical relevance and in order to improve readability.

model algorithms performed the best; however, the performance differences between them are much smaller with noisy functions. In dimensions 3 and 5, the quadratic model is slightly better; dimension 10 is again somewhat of an exception, with the random forest models being slightly better by the end of the search. In the 20 dimension case, the quadratic model clearly performs better. It is also interesting that all the algorithms perform identically for a larger part of the initial search in dimension 20 compared to other dimensions, before ”taking off” and improving the search. Generally, this initial search seems to require more function evaluations than on noise-free functions. This means the models produced are not highly correlated to the data, but the correlation coefficient is still positive and they can still be used later.

The most interesting algorithm behavior can be seen in the group of highly multi-modal functions with severe noise, as shown in figure 7.7. In the 10 dimensional case, algorithm performance is similar, with the EFFX model generally performing better at the start of the search. The final best value reached is produced by the baseline differential evolution algorithm, although the difference is negligible. The random forest and quadratic model algorithms perform similarly and are not noticeably more performant as they are overall. Genetic programming behaves worse than the baseline algorithm at the start of the search. In 20 dimensions, the random forest algorithm performs slightly better than all other algorithms and all other algorithms are worse than the baseline model during at least part of the search process. These noisy multi-modal functions can be considered the most difficult to optimize out of all presented functions.

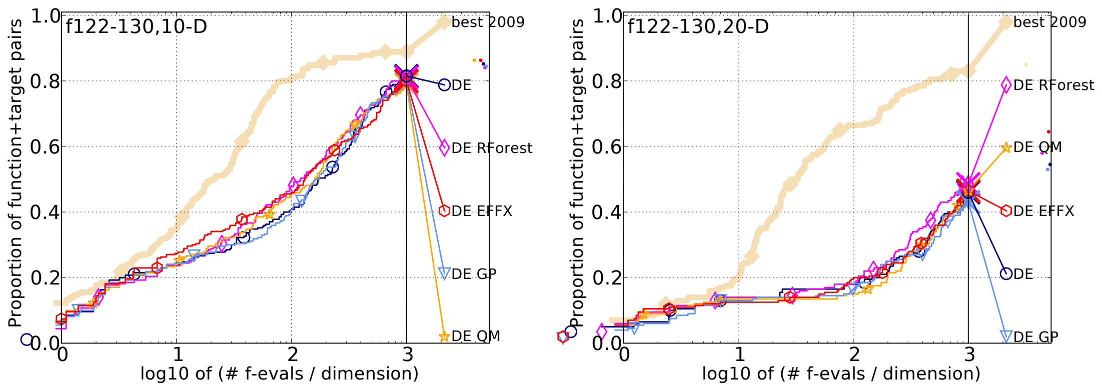


Figure 7.7. Comparison of performance on highly multi-modal functions with severe noise in 10 and 20 dimensions of surrogate-free differential evolution (blue) with random forest (pink), quadratic model (yellow), genetic programming (light blue) and EFFX (red) as surrogate models.

On the two functions with moderate noise, the quadratic model algorithm performs the best in all cases, closely followed by random forests. On functions with severe noise, the difference between the two is only clearly visible in 20 dimensions, with the quadratic model generally performing better.

Quadratic models seem to perform well on functions with Cauchy noise, producing statistically better final target values in 6/10 functions, in at least one dimension; this is also the case on all functions with moderate noise. No other algorithm produced statistically better results.

7.4 Discussion

The performance of symbolic regression as a surrogate model strongly depends on the models the specific algorithm can construct. Each algorithm presented in this thesis has certain limits: the genetic programming models are limited by the tree structure used and the functions available in building them; EFFX models do not contain interacting functions, cannot tune inner function parameters and are also limited by the set of allowed functions. All quadratic models are very similar between themselves, as only the coefficients can differ. Therefore, it is expected that these algorithms will perform well on simple linear functions, as they can easily be represented symbolically. However, both genetic programming and EFFX are outperformed by the simpler quadratic models in these cases.

The EFFX algorithm can and does generate quadratic models; it also generates more complex models and these are ultimately chosen as they better describe the given data. While the problem of overfitting can possibly be improved by using cross-validation, this implies the further division of the already limited data available, which might leave too few data points left to provide any meaningful model. Nevertheless, further thought should be given to this problem, as results show that the simpler, quadratic models are very effective as surrogates. EFFX is also limited in that it does not produce very complex individual expressions. Allowing more levels of function interaction considerably increases computation costs, which are already the highest among tested algorithms. This can possibly be mitigated by only constructing more complex expressions out of ”promising” basis functions, which is then another parameter to tune.

Genetic programming can also generate quadratic models, but probably will not. Models area built using a tree-like structure, and a quadratic model represented as such would require a very deep, asymmetric tree to be constructed. Due to the many measures present to combat bloat, and the nature of the crossover and mutation operators used, this is exceedingly unlikely. Many of the functions used in the COCO platform are calculated by summing over all dimensions and such expressions are also unlikely to be constructed by genetic programming. Indeed, the only times where genetic programming outperformed other algorithms during at least a part of the search, was in low (2 and 3) dimensional cases. This limitation could potentially be overcome by allowing functions of higher arity when constructing the model. With such a modification, generating a quadratic model becomes significantly more likely. This model can then be potentially modified with usual genetic programming tools to provide a better, quadratic-like model.

Although the simplest, quadratic models performed the best out of all tested algorithms in most cases. One possible explanation is that these models have the best local properties, specifically around the optimum. While all symbolic regression models are by definition global, this property is not required when they are used as a surrogate model; surrogate models only need to identify the point with the best fitness, not the fitness value itself. Quadratic models can be used to provide a baseline algorithm for further research into symbolic regression as a surrogate model.

None of the proposed algorithms performs particularly well in comparison with the the *bestGECCO2009* artificial algorithm. This is expected, since the artificial algorithm is an amalgamation of several algorithms, most of which are more sophisticated than differential evolution. Presumably, integrating surrogate models with more state of the art evolutionary algorithms will also lead to performance gains. Comparatively better results were achieved on the noisy function testbed. On several functions,[1] results were close to the artificial algorithm on at least some of the dimensions for several of the algorithms. On the step-ellipsoidal function with severe noise f_{114} all tested algorithms found a final value better than the *bestGECCO2009* algorithm in the 20 dimension case — the only better result across all tests ran.

Such results were anticipated, as symbolic regression methods should be able to "see through" the noise and identify the underlying function. The functions in the noisy testbed are also relatively less complex, meaning they can be represented by simpler symbolic expressions. The quadratic model performed especially well on functions with Cauchy noise. Since the Cauchy adds few (but potentially large) outliers, these might have only a negligible influence on the model near the optima. Nevertheless, this result is significant since the Cauchy noise model is designed as the most complex model of the three used. However, the noisy function testbed was also given less attention at GECCO workshops, especially in 2009 when the artificial algorithm was constructed. As such, it is possible that the results are comparatively better simply because noisy functions were not tested as extensively as noise-free functions.

Although the quadratic model clearly offered the best performance, followed by random forest models, neither of these models dominated the whole search process in all cases. This could be considered a testament to the quality of the COCO platform, as it covers a wide array of functions with differing properties. Results show that each model performed the best in some cases, and few blanket statements about behavior of a specific algorithm on a specific function group can be made. Dimension 10 is especially interesting; as shown in figures 7.3 and 7.4, performance of quadratic model and random forest algorithms was roughly similar, which did not happen in other dimensions. A similar effect is seen on the noisy function testbed, as illustrated in figures 7.6 and 7.7. EFFX also performed relatively better than expected on the group of multi-modal functions with severe noise, but only in the 10 dimensions case. No explanation for this phenomenon has been identified.

Due to the chosen model management strategy, the effect of surrogate models is first seen after the third generation of the differential evolution algorithm, i.e. after 30 function evaluations per dimension. At the start of the search, all algorithms provide an improvement, since the demands placed on model quality are not very high, thus differential evolution is able to approach the optima faster. At some point, especially noticeable with multi-modal functions, symbolic regression algorithms can no longer accurately model the area around the optima, and the surrogate models are no longer used. If the algorithm used cannot produce models with the required expressivity, this happens sooner. In these cases, the baseline algorithm is able to converge towards the same solution within the given budget, since surrogate models only help the evolutionary algorithm find the optima faster; given enough function evaluations, any algorithm should be able to reach the same results. In the results shown, this can be usually seen on the genetic programming and EFFX algorithms, since the usefulness of the other two models was not exhausted within the given budget.

The model management strategy used can be characterized as conservative, as it will disable surrogate models as soon as a negative correlation is observed. This is done in an effort to assure that differential evolution with surrogate models will never perform worse than the baseline, surrogate-free algorithm. This goal was mostly accomplished, with few exceptions. However, it seems that this choice has had a profound influence on overall behavior during the search, as all symbolic regression models follow the same

[1] Specifically, on functions $f_{108}, f_{111}, f_{113} - f_{117}, f_{119}, f_{120} - f_{122}, f_{128} - f_{130}$

basic ”shape” described in the previous paragraph.[1] By setting a constant parameter $kmax$, denoting the degree of model usage, the differential evolution algorithm might always be given ”bad” points to evaluate if the model is inaccurate and $kmax$ is high, or will underutilize the model if it is accurate and $kmax$ is low. Thus, the parameter should be varied based on model quality.

In the strategy used, quality is only measured after the model has been used in the current generation, on the newly generated data points — the only points it was not trained on. However, with the addition of these data points, the model generated in the next generation can be very different. This is particularly likely in genetic programming and EFFX, less so for quadratic models. Therefore, the $kmax$ parameter used might not be appropriate for the generated model. Dividing the available data between a training and testing set is one possibility of mitigating this issue. Alternatively, the current strategy might be modified as the search progresses. Currently, all points with fitness better than the median value are used in constructing the model; however, this can be misleading later during the search when many points, but less than half, are very close to the optimum (as in, for example, the Rosenbrock function). As symbolic regression constructs global models, these additional points can have a large negative effect on overall algorithm behavior. Other measures of model quality can also be considered.

Irregardless of possible improvements to the model management strategy or the individual algorithms, the results clearly show that using surrogate models noticeably improved overall algorithm performance during the entire search process.

[1] The random forest algorithm is the occasional exception.

Chapter 8
Conclusion

Differential evolution, a representative evolutionary algorithm, was implemented in the Python programming language. A model management strategy for integrating surrogate models with differential evolution was designed. This strategy uses the rank correlation coefficient between evaluations of the real and surrogate models to tune how often the surrogate model will be used; surrogate model are disabled completely if the correlation coefficient is negative. Surrogate model quality is further improved by using only a part of the available data in its construction.

Three symbolic regression algorithms were implemented and tested against two baseline regression models — random forests and k-nearest neighbors — using the proposed model management strategy. Polynomial regression of rank two was used to construct quadratic models, the simplest symbolic regression models considered. Extended fast function extraction is a recently proposed, non-evolutionary, deterministic method for symbolic regression which uses regularized learning on a massive set of generated basis functions. An existing implementation of genetic programming was used as the third tested algorithm. Tests were ran using the COCO platform, which includes a well-designed set of representative noiseless and noisy functions.

Results show that the model management strategy is well designed, as all surrogate models speed up the optimization during at least parts of the search process. Overall, quadratic models have the best performance in almost all functions considered, followed by random forest models. The other two symbolic regression methods performed worse, being the best only rarely and only during some parts of the search process.

Based on the results, I consider the use of symbolic regression as a surrogate model in evolutionary algorithms a promising topic for further research. I have identified two main areas for further research and algorithm refinement. First, the proposed solutions should be tested on a state of the art evolutionary algorithm. Quadratic models performed well and can be used as a baseline. The other two algorithms can also be improved, for example by introducing cross-validation or modifying the algorithms to generate a wider array of possible models. Alternatively, another symbolic regression algorithm can be used.

Second, an appropriate measure of model quality needs to be devised in order to design an efficient model management strategy. The model management strategy chosen seems to have a large effect on search performance: badly chosen strategies can overuse the model even if it is bad or underutilize a good model, which both lead to less effective search. The measure used in this thesis, the rank correlation coefficient, can be considered a good start.

In conclusion, black-box optimization problems regularly appear in practice and are often limited by the number of available evaluations of the black-box model. Surrogate models help reduce the number of evaluations needed during the optimization process, allowing higher effective budgets and thereby improving the final results reached. This thesis shows that symbolic regression is a promising regression analysis technique for use as a surrogate model.

References

[1] Darrell Whitley. An overview of evolutionary algorithms: practical issues and common pitfalls. *Information and software technology*. 2001, 43 (14), 817–831.

[2] Jakob Vesterstrøm, and Rene Thomsen. *A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems*. In: *Evolutionary Computation, 2004. CEC2004. Congress on*. 2004. 1980–1987.

[3] Yew S Ong, Prasanth B Nair, and Andrew J Keane. Evolutionary optimization of computationally expensive problems via surrogate modeling. *AIAA journal*. 2003, 41 (4), 687–696.

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

[5] Trent McConaghy. *FFX: Fast, scalable, deterministic symbolic regression technology*. In: *Genetic Programming Theory and Practice IX*. Springer, 2011. 235–260.

[6] JD Gergonne. The application of the method of least squares to the interpolation of sequences. *Historia Mathematica*. 1974, 1 (4), 439–447.

[7] Riccardo Poli, William B Langdon, Nicholas F McPhee, and John R Koza. *A field guide to genetic programming*. Lulu. com, 2008.

[8] Nikolaus Hansen, Steffen Finck, Raymond Ros, and Anne Auger. Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. 2009,

[9] Nikolaus Hansen, Steffen Finck, Raymond Ros, and Anne Auger. Real-parameter black-box optimization benchmarking 2009: Noisy functions definitions.

[10] Rainer Storn, and Kenneth Price. *Differential evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces*. ICSI Berkeley, 1995.

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

[12] Leo Breiman. Random forests. *Machine learning*. 2001, 45 (1), 5–32.

[13] Daniela Zaharie. *Critical values for the control parameters of differential evolution algorithms*. In: *Proceedings of MENDEL*. 2002.

[14] Yaochu Jin, Markus Olhofer, and Bernhard Sendhoff. *On Evolutionary Optimization with Approximate Fitness Functions.*. In: *GECCO*. 2000. 786–793.

[15] Michael Emmerich, Alexios Giotis, Mutlu Özdemir, Thomas Bäck, and Kyriakos Giannakoglou. *Metamodel—assisted evolution strategies*. In: *Parallel Problem Solving from Nature—PPSN VII*. Springer, 2002. 361–370.

[16] Yaochu Jin, Michael Hüsken, and Bernhard Sendhoff. *Quality measures for approximate models in evolutionary computation*. In: *GECCO*. 2003. 170–173.

[17] Dudy Lim, Yaochu Jin, Yew-Soon Ong, and Bernhard Sendhoff. Generalizing surrogate-assisted evolutionary computation. *Evolutionary Computation, IEEE Transactions on.* 2010, 14 (3), 329–355.

[18] Piyabute Fuangkhon, and Thitipong Tanprasert. *An incremental learning algorithm for supervised neural network with contour preserving classification.* In: *Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, 2009. ECTI-CON 2009. 6th International Conference on.* 2009. 740–743.

[19] Naomi S Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician.* 1992, 46 (3), 175–185.

[20] Kilian Q Weinberger, John Blitzer, and Lawrence K Saul. *Distance metric learning for large margin nearest neighbor classification.* In: *Advances in neural information processing systems.* 2005. 1473–1480.

[21] J. Ross Quinlan. Induction of decision trees. *Machine learning.* 1986, 1 (1), 81–106.

[22] David Opitz, and Richard Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research.* 1999, 169–198.

[23] Andy Liaw, and Matthew Wiener. Classification and regression by randomForest. *R news.* 2002, 2 (3), 18–22.

[24] Ekaterina J Vladislavleva, Guido F Smits, and Dick Den Hertog. Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. *Evolutionary Computation, IEEE Transactions on.* 2009, 13 (2), 333–349.

[25] Lonnie Magee. Nonlocal behavior in polynomial regressions. *The American Statistician.* 1998, 52 (1), 20–22.

[26] Lawrence Beadle, and Colin G Johnson. *Semantically driven mutation in genetic programming..* In: *IEEE Congress on Evolutionary Computation.* 2009. 1336–1342.

[27] Sean Luke, and Lee Spector. A revised comparison of crossover and mutation in genetic programming. *Genetic Programming.* 1998, 98 (208-213), 55.

[28] Terence Soule, and James A Foster. Effects of code growth and parsimony pressure on populations in genetic programming. *Evolutionary Computation.* 1998, 6 (4), 293–309.

[29] Hui Zou, and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology).* 2005, 67 (2), 301–320.

[30] Steffen Finck, Nikolaus Hansen, Raymond Ros, and Anne Auger. Real-parameter black-box optimization benchmarking 2009: Presentation of the noiseless functions.

[31] Steffen Finck, Nikolaus Hansen, Raymond Ros, and Anne Auger. Real-parameter black-box optimization benchmarking 2009: Presentation of the noisy functions.

[32] Nikolaus Hansen, Anne Auger, Steffen Finck, and Raymond Ros. Real-parameter black-box optimization benchmarking 2010: Experimental setup.

[33] Travis E Oliphant. Python for scientific computing. *Computing in Science & Engineering.* 2007, 9 (3), 10–20.

[34] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The NumPy array: a structure for efficient numerical computation. *Computing in Science & Engineering.* 2011, 13 (2), 22–30.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research.* 2011, 12 2825–2830.

[36] *gplearn* [software]. *Documentation.*
`http://gplearn.readthedocs.io/en/stable/`, URL accessed on 2016-05-15.

[37] SymPy Development Team. *SymPy: Python library for symbolic mathematics.* 2016.
`http://www.sympy.org`, URL accessed on 2016-5-12.

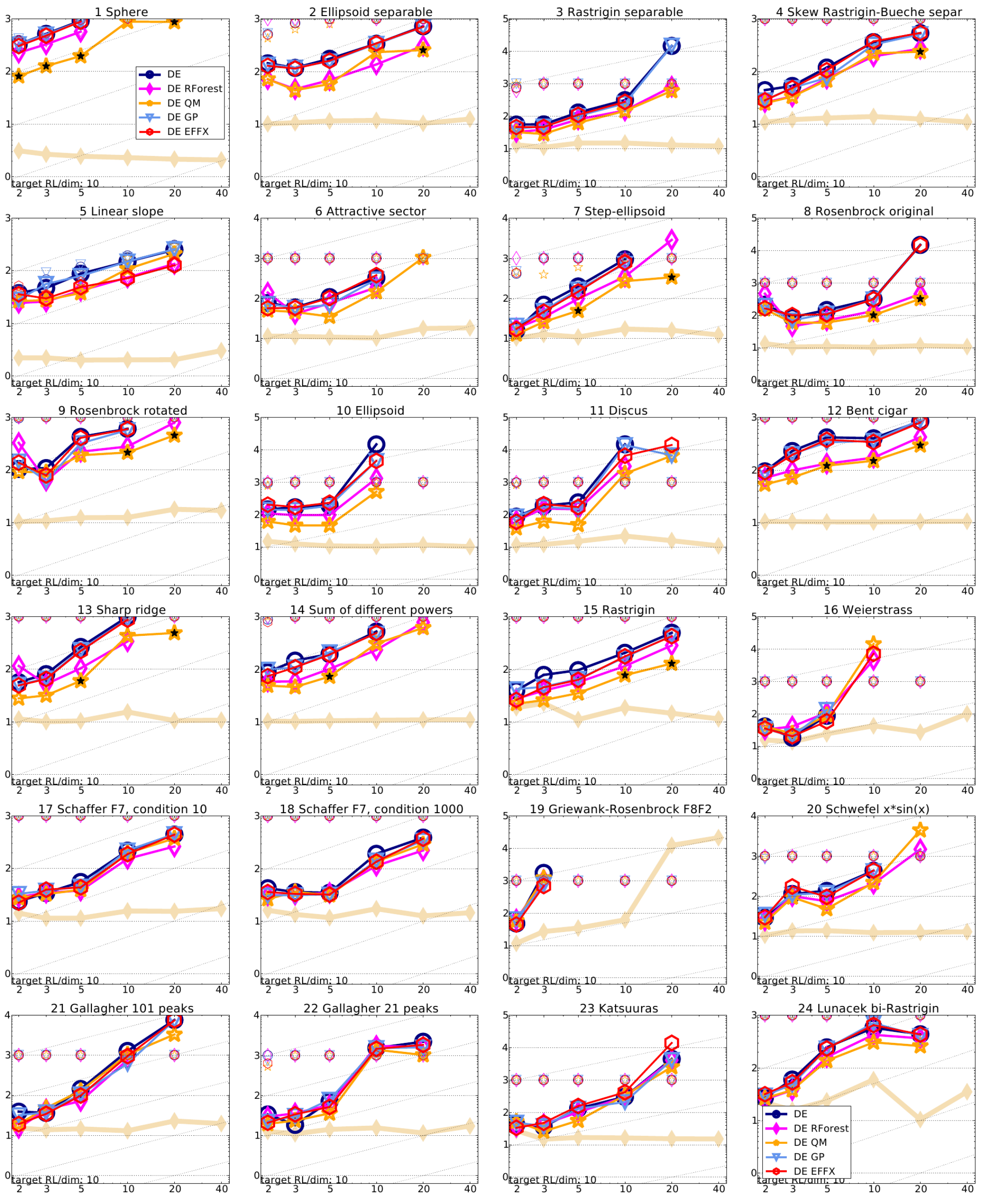
[38] *FFX* [software]. *GitHub homepage.*
`https://github.com/natekupp/ffx`, URL accessed on 2016-5-15.

Appendix A
CD contents

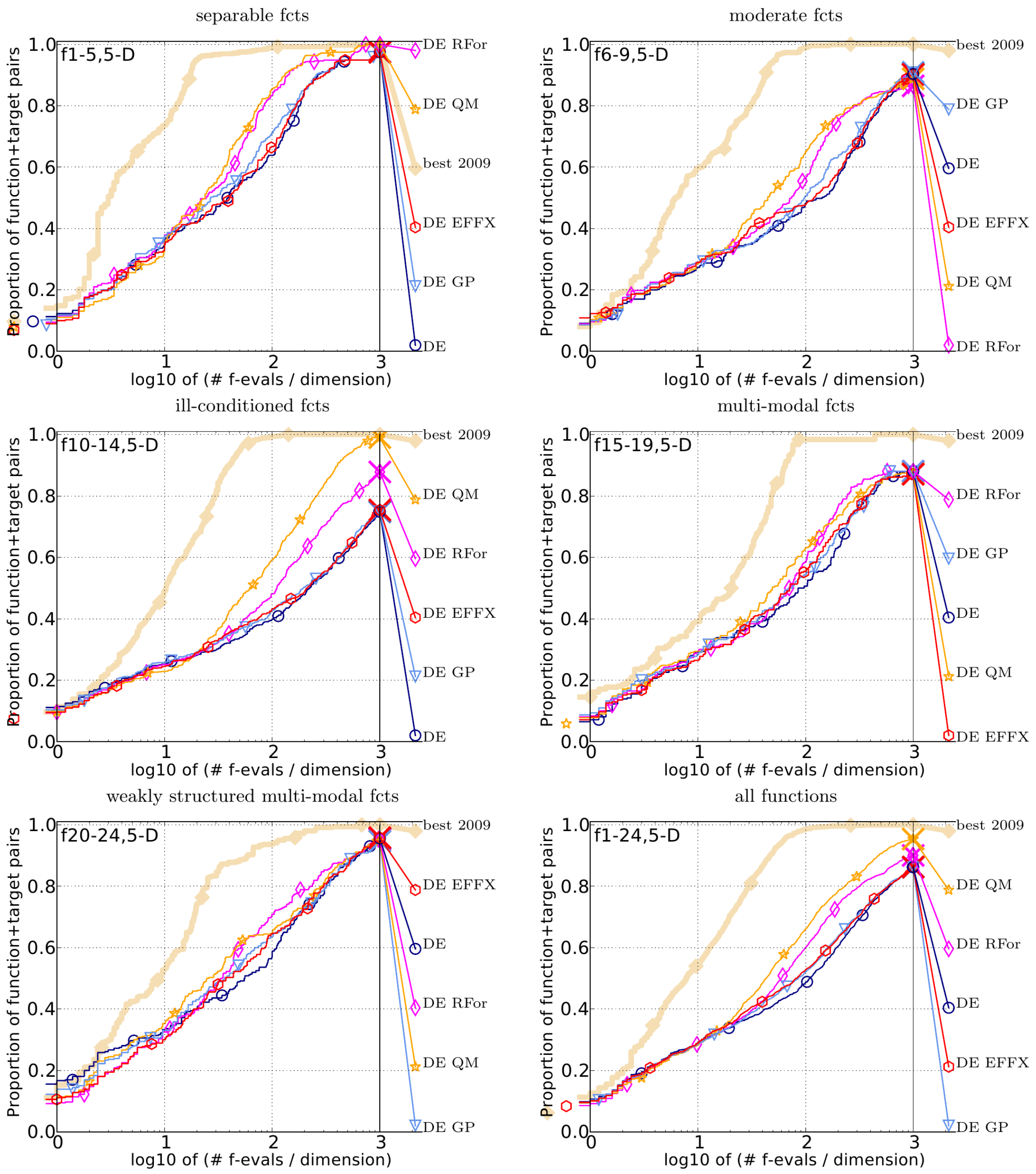
The attached CD contains:

- COCO platform v15.03 source code
- Python implementation of all used algorithms
- Raw experimental data
- Processed experimental data
- Thesis in PDF

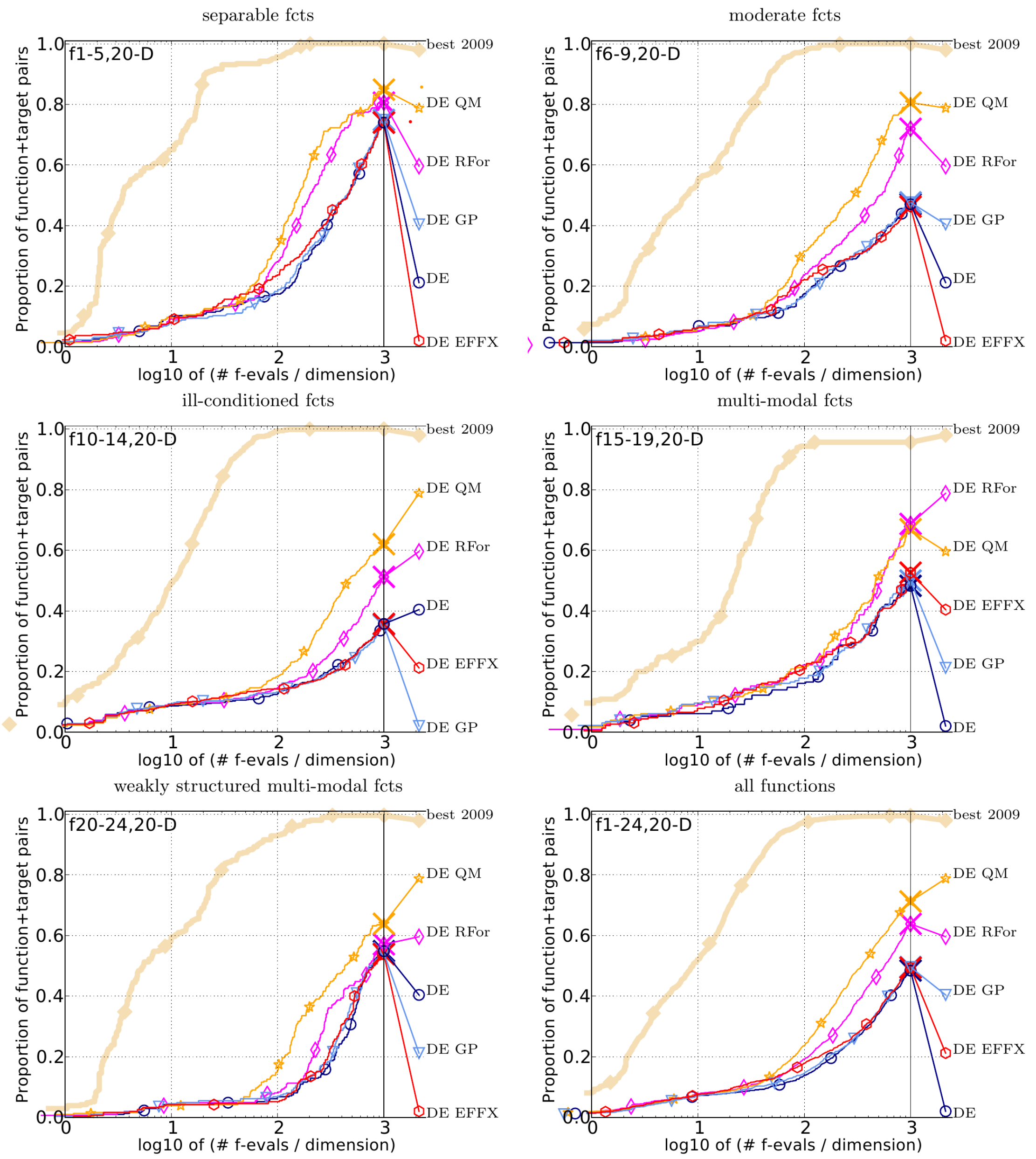
Appendix B
Experimental results



Expected running time (ERT in number of f-evaluations as $\log_{10}$ value) divided by dimension versus dimension. The target function value is chosen such that the bestGECCO2009 artificial algorithm just failed to achieve an ERT of $10\times$DIM. Different symbols correspond to different algorithms given in the legend of f_1 and f_{24}. Light symbols give the maximum number of function evaluations from the longest trial divided by dimension. Black stars indicate a statistically better result compared to all other algorithms with $p < 0.01$ and Bonferroni correction number of dimensions (six). Legend: ○: DE, ◇: DE RForest, ⋆: DE QM, ▽: DE GP, ⬡: DE EFFX



Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for all functions and subgroups in 5-D. The targets are chosen from $10^{[-8..2]}$ such that the bestGECCO2009 artificial algorithm just not reached them within a given budget of $k \times \mathrm{DIM}$, with $k \in \{0.5, 1.2, 3, 10, 50\}$. The "best 2009" line corresponds to the best ERT observed during BBOB 2009 for each selected target.



Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for all functions and subgroups in 20-D. The targets are chosen from $10^{[-8..2]}$ such that the bestGECCO2009 artificial algorithm just not reached them within a given budget of $k \times \mathrm{DIM}$, with $k \in \{0.5, 1.2, 3, 10, 50\}$. The “best 2009” line corresponds to the best ERT observed during BBOB 2009 for each selected target.

#FEs/D	0.5	1.2	3	10	50	#succ
f1	*2.5e+1*:4.8	*1.6e+1*:7.6	*1.0e-8*:12	*1.0e-8*:12	*1.0e-8*:12	15/15
DE	**2.8**(2)	4.7(3)	374(14)	374(21)	374(21)	15/15
DE RFor	**2.8**(2)	**4.0**(4)	232(15)	232(19)	232(15)	15/15
DE QM	**2.8**(2)	4.0(4)	**80**(2)$^{\star 4}$	**80**(3)$^{\star 4}$	**80**(2)$^{\star 4}$	15/15
DE GP	**2.8**(2)	4.6(7)	364(13)	364(17)	364(19)	15/15
DE EFFX	**2.8**(3)	4.0(4)	361(14)	361(18)	361(28)	15/15

#FEs/D	0.5	1.2	3	10	50	#succ
f2	*1.6e+6*:2.9	*4.0e+5*:11	*4.0e+4*:15	*6.3e+2*:58	*1.0e-8*:95	15/15
DE	**1.9**(3)	**2.2**(2)	10(6)	15(2)	∞ *5000*	0/15
DE RFor	**1.9**(3)	2.2(0.9)	**6.0**(3)	6.0(1)	37(2)	15/15
DE QM	**1.9**(3)	**2.2**(2)	6.4(4)	**5.0**(1)	**33**(7)	15/15
DE GP	**1.9**(3)	2.2(2)	8.0(7)	14(3)	∞ *5000*	0/15
DE EFFX	**1.9**(1)	**2.2**(0.6)	12(6)	14(3)	∞ *5000*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f3	*1.6e+2*:4.1	*1.0e+2*:15	*6.3e+1*:23	*2.5e+1*:73	*1.0e+1*:716	15/15
DE	**2.8**(2)	**2.0**(2)	6.0(5)	8.8(3)	2.2(1.0)	15/15
DE RFor	**2.8**(2)	2.1(3)	4.5(3)	5.4(1)	**1.2**(0.2)	15/15
DE QM	**2.8**(1)	2.1(2)	**4.1**(2)	**4.2**(1)	1.3(0.5)	15/15
DE GP	**2.8**(1)	2.1(2)	4.7(4)	6.9(3)	2.0(0.9)	15/15
DE EFFX	**2.8**(3)	2.1(3)	5.2(4)	7.5(3)	2.1(0.7)	15/15

#FEs/D	0.5	1.2	3	10	50	#succ
f4	*2.5e+2*:2.6	*1.6e+2*:10	*1.0e+2*:19	*4.0e+1*:65	*1.6e+1*:434	15/15
DE	**2.5**(2)	2.6(3)	5.6(5)	9.1(2)	3.1(1)	15/15
DE RFor	**2.5**(4)	**2.1**(2)	3.1(3)	5.6(1)	2.0(0.9)	15/15
DE QM	**2.5**(4)	2.5(6)	**2.7**(4)	**5.2**(3)	**1.9**(1)	15/15
DE GP	**2.5**(3)	2.2(3)	3.9(4)	5.8(4)	2.6(0.7)	15/15
DE EFFX	**2.5**(3)	2.2(3)	2.8(4)	8.1(3)	2.8(0.9)	15/15

#FEs/D	0.5	1.2	3	10	50	#succ
f5	*6.3e+1*:4.0	*4.0e+1*:10	*1.0e-8*:10	*1.0e-8*:10	*1.0e-8*:10	15/15
DE	**2.1**(2)	5.1(4)	44(9)	44(11)	44(12)	15/15
DE RFor	**2.1**(2)	**4.5**(3)	21(5)	21(4)	21(2)	15/15
DE QM	**2.1**(1)	4.6(3)	**19**(4)	**19**(4)	**19**(5)	15/15
DE GP	**2.1**(2)	5.2(4)	39(16)	39(11)	39(18)	15/15
DE EFFX	**2.1**(2)	4.6(2)	24(5)	24(4)	24(5)	15/15

#FEs/D	0.5	1.2	3	10	50	#succ
f6	*1.0e+5*:3.0	*2.5e+4*:8.4	*1.0e+2*:16	*2.5e+1*:54	*2.5e-1*:254	15/15
DE	**2.0**(3)	2.8(2)	7.8(7)	10(12)	15(2)	15/15
DE RFor	**2.0**(2)	2.1(3)	5.0(7)	7.0(4)	20(17)	11/15
DE QM	**2.0**(3)	2.0(2)	**3.3**(2)	**3.3**(4)	**11**(3)	15/15
DE GP	**2.0**(3)	**2.0**(4)	6.8(6)	6.3(6)	14(3)	15/15
DE EFFX	**2.0**(3)	2.1(5)	3.6(3)	9.5(8)	14(3)	15/15

#FEs/D	0.5	1.2	3	10	50	#succ
f7	*1.6e+2*:4.2	*1.0e+2*:6.2	*2.5e+1*:20	*4.0e+0*:54	*1.0e+0*:324	15/15
DE	**1**(0.5)	1.7(2)	6.2(6)	18(7)	6.3(3)	15/15
DE RFor	**1**(3)	1.8(5)	4.5(7)	9.0(5)	3.1(4)	15/15
DE QM	**1**(2)	**1.7**(4)	**3.5**(3)	**4.5**(1)*	**1.3**(0.3)$^{\star 2}$	15/15
DE GP	**1**(2)	1.9(2)	5.8(6)	15(10)	6.4(3)	15/15
DE EFFX	**1**(0.8)	**1.7**(3)	4.6(5)	14(6)	5.2(2)	15/15

#FEs/D	0.5	1.2	3	10	50	#succ
f8	*1.0e+4*:4.6	*6.3e+3*:6.8	*1.0e+3*:18	*6.3e+1*:54	*1.6e+0*:258	15/15
DE	**3.1**(3)	**3.2**(3)	6.6(6)	13(4)	18(4)	13/15
DE RFor	**3.1**(5)	**3.2**(3)	4.7(4)	6.4(1)	83(82)	3/15
DE QM	**3.1**(2)	**3.2**(3)	**4.4**(3)	**5.5**(3)	54(34)	5/15
DE GP	**3.1**(2)	**3.2**(3)	6.0(6)	10(5)	16(4)	13/15
DE EFFX	**3.1**(4)	**3.2**(2)	4.9(6)	10(4)	**16**(3)	14/15

#FEs/D	0.5	1.2	3	10	50	#succ
f9	*2.5e+1*:20	*1.6e+1*:26	*1.0e+1*:35	*4.0e+0*:62	*1.6e-2*:256	15/15
DE	51(11)	46(13)	42(12)	34(8)	∞ *5000*	0/15
DE RFor	20(6)	18(3)	**16**(3)	18(17)	∞ *5000*	0/15
DE QM	**19**(3)	**17**(5)	16(4)	**15**(4)	∞ *5000*	0/15
DE GP	32(17)	32(11)	26(15)	26(9)	∞ *5000*	0/15
DE EFFX	47(21)	45(17)	39(14)	33(9)	∞ *5000*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f10	*2.5e+6*:2.9	*6.3e+5*:7.0	*2.5e+5*:17	*6.3e+3*:54	*2.5e+1*:297	15/15
DE	**1.6**(2)	**1.0**(2)	**0.96**(1)	18(14)	∞ *5000*	0/15
DE RFor	**1.6**(2)	**1.0**(1)	**0.96**(0.8)	9.0(5)	83(47)	3/15
DE QM	**1.6**(2)	**1.0**(2)	**0.96**(0.8)	**4.3**(2)	**6.5**(1)$^{\star 4}$	15/15
DE GP	**1.6**(1)	**1.0**(1)	**0.96**(1)	17(16)	∞ *5000*	0/15
DE EFFX	**1.6**(3)	**1.0**(2)	**0.96**(1)	22(20)	∞ *5000*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f11	*1.0e+6*:3.0	*6.3e+4*:6.2	*6.3e+2*:16	*6.3e+1*:74	*6.3e-1*:298	15/15
DE	**1.5**(1)	**2.3**(3)	7.4(9)	16(13)	∞ *5000*	0/15
DE RFor	**1.5**(1)	2.4(4)	7.1(6)	10(7)	∞ *5000*	0/15
DE QM	**1.5**(2)	**2.3**(1)	**4.8**(4)	**3.2**(2)	**7.0**(2)$^{\star 4}$	15/15
DE GP	**1.5**(2)	2.3(3)	6.6(8)	10(19)	∞ *5000*	0/15
DE EFFX	**1.5**(1)	2.7(7)	7.4(7)	11(9)	∞ *5000*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f12	*4.0e+7*:3.6	*1.6e+7*:7.6	*4.0e+6*:19	*1.6e+4*:52	*1.0e+0*:268	15/15
DE	**1.2**(1)	2.7(4)	6.4(6)	41(12)	∞ *5000*	0/15
DE RFor	**1.2**(1)	2.2(3)	4.8(5)	13(3)	21(15)	10/15
DE QM	**1.2**(1)	**2.1**(2)	**3.9**(3)	**12**(12)*	**18**(10)	12/15
DE GP	**1.2**(2)	2.5(6)	5.7(5)	33(11)	∞ *5000*	0/15
DE EFFX	**1.2**(2)	**2.1**(1)	4.1(3)	36(15)	∞ *5000*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f13	*1.0e+3*:2.8	*6.3e+2*:8.4	*4.0e+2*:17	*6.3e+1*:52	*6.3e-2*:264	15/15
DE	**2.3**(2)	3.1(2)	8.0(7)	25(9)	∞ *5000*	0/15
DE RFor	**2.3**(2)	3.0(3)	6.4(5)	10(3)	134(110)	2/15
DE QM	**2.3**(3)	**2.9**(3)	**4.8**(2)	**5.8**(1)$^{\star 4}$	**11**(3)$^{\star 3}$	15/15
DE GP	**2.3**(2)	3.4(3)	7.8(5)	22(8)	∞ *5000*	0/15
DE EFFX	**2.3**(3)	3.1(2)	5.5(4)	21(7)	∞ *5000*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f14	*1.6e+1*:3.0	*1.0e+1*:10	*6.3e+0*:15	*2.5e-1*:53	*1.0e-5*:251	15/15
DE	**2.3**(4)	1.8(1)	3.3(4)	19(2)	∞ *5000*	0/15
DE RFor	**2.3**(3)	1.7(1)	3.1(4)	10(2)	∞ *5000*	0/15
DE QM	**2.3**(2)	**1.6**(2)	**2.9**(2)	**6.9**(0.9)$^{\star 2}$	**15**(2)$^{\star 3}$	14/15
DE GP	**2.3**(2)	2.2(1)	4.4(4)	20(3)	∞ *5000*	0/15
DE EFFX	**2.3**(4)	**1.6**(2)	2.9(3)	18(6)	∞ *5000*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f15	*1.6e+2*:3.0	*1.0e+2*:13	*6.3e+1*:24	*4.0e+1*:55	*1.6e+1*:289	5/5
DE	**2.0**(2)	2.3(2)	6.8(6)	8.7(3)	6.3(2)	15/15
DE RFor	**2.0**(2)	2.0(2)	5.0(4)	5.1(1)	4.4(4)	15/15
DE QM	**2.0**(2)	**1.9**(2)	**4.3**(3)	**3.2**(1)	**2.4**(2)	15/15
DE GP	**2.0**(2)	2.0(2)	5.5(3)	5.9(2)	4.8(4)	15/15
DE EFFX	**2.0**(2)	2.2(2)	4.4(3)	5.7(4)	5.7(4)	15/15

#FEs/D	0.5	1.2	3	10	50	#succ
f16	*4.0e+1*:4.8	*2.5e+1*:16	*1.6e+1*:46	*1.0e+1*:120	*4.0e+0*:334	15/15
DE	**1.5**(4)	1.2(2)	**1.2**(1)	3.6(7)	21(16)	7/15
DE RFor	**1.5**(1)	1.2(0.9)	1.5(3)	4.5(4)	27(34)	6/15
DE QM	**1.5**(2)	1.1(1)	1.4(2)	3.8(3)	**20**(26)	9/15
DE GP	**1.5**(4)	1.4(2)	2.7(9)	5.5(11)	25(26)	7/15
DE EFFX	**1.5**(4)	**1.1**(0.5)	1.6(3)	**2.4**(2)	38(19)	5/15

#FEs/D	0.5	1.2	3	10	50	#succ
f17	*1.0e+1*:5.2	*6.3e+0*:26	*4.0e+0*:57	*2.5e+0*:110	*6.3e-1*:412	15/15
DE	3.0(3)	3.1(2)	4.9(3)	4.6(3)	4.4(0.9)	15/15
DE RFor	3.0(2)	**2.4**(2)	**3.3**(2)	3.9(2)	**2.3**(1)	15/15
DE QM	3.1(4)	2.7(4)	3.5(1)	**3.4**(2)	2.7(1)	15/15
DE GP	**3.0**(5)	3.1(3)	4.0(3)	3.6(3)	3.7(1)	15/15
DE EFFX	3.1(4)	2.4(2)	3.8(2)	3.7(3)	3.5(0.9)	15/15

#FEs/D	0.5	1.2	3	10	50	#succ
f18	*6.3e+1*:3.4	*4.0e+1*:7.2	*2.5e+1*:20	*1.6e+1*:58	*1.6e+0*:318	15/15
DE	**1.5**(1)	3.2(4)	3.4(2)	3.0(2)	7.4(3)	15/15
DE RFor	**1.5**(1)	3.2(3)	**2.9**(2)	2.9(3)	**4.4**(2)	15/15
DE QM	**1.5**(1)	**3.0**(4)	3.2(4)	**2.7**(3)	6.4(3)	15/15
DE GP	**1.5**(1)	3.1(5)	3.0(2)	2.8(2)	7.7(1)	15/15
DE EFFX	**1.5**(2)	3.2(2)	3.1(4)	2.7(2)	7.8(2)	15/15

#FEs/D	0.5	1.2	3	10	50	#succ
f19	*1.6e-1*:172	*1.0e-1*:242	*6.3e-2*:675	*4.0e-2*:3078	*2.5e-2*:4946	15/15
DE	∞	∞	∞	∞	∞ *5000*	0/15
DE RFor	∞	∞	∞	∞	∞ *5000*	0/15
DE QM	∞	∞	∞	∞	∞ *5000*	0/15
DE GP	∞	∞	∞	∞	∞ *5000*	0/15
DE EFFX	∞	∞	∞	∞	∞ *5000*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f20	*6.3e+3*:5.1	*4.0e+3*:8.4	*4.0e+1*:15	*2.5e+0*:69	*1.0e+0*:851	15/15
DE	**1.9**(2)	2.1(3)	12(10)	10(3)	4.4(2)	14/15
DE RFor	**1.9**(2)	2.2(1)	8.9(6)	5.4(2)	3.7(0.5)	15/15
DE QM	**1.9**(2)	**2.0**(4)	**6.4**(3)	**3.5**(2)	**3.6**(1)	15/15
DE GP	**1.9**(2)	2.3(3)	8.5(6)	8.8(3)	5.5(3)	12/15
DE EFFX	**1.9**(2)	**2.0**(3)	9.4(8)	7.0(5)	7.0(3)	10/15

#FEs/D	0.5	1.2	3	10	50	#succ
f21	*4.0e+1*:3.9	*2.5e+1*:11	*1.6e+1*:31	*6.3e+0*:73	*1.6e+0*:347	5/5
DE	**1.6**(1)	1.4(0.9)	1.6(3)	10(8)	9.4(4)	13/15
DE RFor	**1.6**(1)	1.3(0.8)	1.7(1)	**4.9**(2)	**6.2**(5)	15/15
DE QM	**1.6**(1)	**1.2**(1.0)	1.7(2)	8.8(7)	7.9(6)	14/15
DE GP	**1.6**(0.9)	1.3(3)	1.7(2)	7.2(6)	8.9(6)	13/15
DE EFFX	**1.6**(1)	**1.2**(2)	**1.3**(1)	6.7(7)	9.0(9)	14/15

#FEs/D	0.5	1.2	3	10	50	#succ
f22	*6.3e+1*:3.6	*4.0e+1*:15	*2.5e+1*:32	*1.0e+1*:71	*1.6e+0*:341	5/5
DE	**2.0**(3)	**1.2**(2)	2.6(2)	5.0(5)	10(12)	12/15
DE RFor	**2.0**(1)	1.3(2)	2.6(3)	4.1(3)	**7.6**(6)	14/15
DE QM	**2.0**(3)	1.3(2)	2.5(3)	**2.5**(3)	9.1(9)	13/15
DE GP	**2.0**(3)	1.2(1)	3.3(4)	5.5(8)	8.2(6)	13/15
DE EFFX	**2.0**(3)	1.6(4)	**2.2**(2)	3.6(2)	13(15)	11/15

#FEs/D	0.5	1.2	3	10	50	#succ
f23	*1.0e+1*:3.0	*6.3e+0*:9.0	*4.0e+0*:33	*2.5e+0*:84	*1.0e+0*:518	15/15
DE	**2.5**(5)	2.3(3)	3.3(3)	7.9(6)	**32**(66)	4/15
DE RFor	**2.5**(4)	2.4(1)	3.6(8)	6.1(8)	140(258)	1/15
DE QM	**2.5**(5)	**2.1**(0.9)	**1.9**(1)	**3.3**(3)	66(92)	2/15
DE GP	**2.5**(2)	2.5(3)	4.2(13)	7.7(5)	144(133)	1/15
DE EFFX	**2.5**(1)	2.2(2)	2.4(3)	9.4(9)	41(65)	3/15

#FEs/D	0.5	1.2	3	10	50	#succ
f24	*6.3e+1*:15	*4.0e+1*:37	*4.0e+1*:37	*2.5e+1*:118	*1.6e+1*:692	15/15
DE	3.8(3)	8.1(8)	8.1(3)	10(5)	11(15)	8/15
DE RFor	3.9(3)	**6.9**(5)	**6.9**(2)	6.8(2)	6.0(8)	12/15
DE QM	**3.6**(3)	6.9(3)	6.9(6)	**5.8**(5)	**4.3**(3)	13/15
DE GP	5.0(5)	7.5(5)	7.5(8)	10(6)	7.0(5)	11/15
DE EFFX	4.1(3)	9.1(4)	9.1(5)	10(2)	6.8(4)	12/15

Expected running time (ERT in number of function evaluations) divided by the respective best ERT measured during BBOB-2009 in dimension 5. The ERT and in braces, as dispersion measure, the half difference between 90 and 10%-tile of bootstrapped run lengths appear for each algorithm and run-length based target, the corresponding best ERT (preceded by the target Δf-value in *italics*) in the first row. #succ is the number of trials that reached the target value of the last column. The median number of conducted function evaluations is additionally given in *italics*, if the target in the last column was never reached. Entries, succeeded by a star, are statistically significantly better (according to the rank-sum test) when compared to all other algorithms of the table, with $p = 0.05$ or $p = 10^{-k}$ when the number k following the star is larger than 1, with Bonferroni correction by the number of instances. A ↓ indicates the same tested against the best algorithm of BBOB-2009. Best results are printed in bold.

#FEs/D	0.5	1.2	3	10	50	#succ
f1	*6.3e+1*:24	*4.0e+1*:42	*1.0e-8*:43	*1.0e-8*:43	*1.0e-8*:43	15/15
DE	94(22)	88(16)	∞	∞	∞ *2e4*	0/15
DE RFor	69(12)	61(11)	∞	∞	∞ *2e4*	0/15
DE QM	**43**(12)	**35**(19)^{*2}	**403**(120)^{*2}	**403**(119)^{*2}	**403**(5)^{*2}	14/15
DE GP	69(32)	78(22)	∞	∞	∞ *2e4*	0/15
DE EFFX	64(25)	72(26)	∞	∞	∞ *2e4*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f2	*4.0e+6*:29	*2.5e+6*:42	*1.0e+5*:65	*1.0e+4*:207	*1.0e-8*:412	15/15
DE	**1.9**(3)	3.1(3)	98(10)	70(5)	∞ *2e4*	0/15
DE RFor	**1.9**(2)	2.9(4)	41(6)	30(2)	∞ *2e4*	0/15
DE QM	**1.9**(3)	**2.8**(3)	**39**(34)*	**25**(15)*	∞ *2e4*	0/15
DE GP	**1.9**(1)	3.0(3)	91(13)	68(4)	∞ *2e4*	0/15
DE EFFX	**1.9**(2)	**2.8**(4)	91(16)	69(4)	∞ *2e4*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f3	*6.3e+2*:33	*4.0e+2*:44	*1.6e+2*:109	*1.0e+2*:255	*2.5e+1*:3277	15/15
DE	16(16)	66(22)	107(16)	1157(884)	∞ *2e4*	0/15
DE RFor	11(9)	36(19)	63(12)	63(24)	∞ *2e4*	0/15
DE QM	12(10)	**28**(8)	**34**(6)^{*4}	**47**(12)	∞ *2e4*	0/15
DE GP	15(11)	55(13)	110(11)	1178(1002)	∞ *2e4*	0/15
DE EFFX	**11**(8)	53(18)	105(6)	∞	∞ *2e4*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f4	*6.3e+2*:22	*4.0e+2*:91	*2.5e+2*:250	*1.6e+2*:332	*6.3e+1*:1927	15/15
DE	79(44)	61(8)	43(5)	66(20)	∞ *2e4*	0/15
DE RFor	60(44)	34(12)	22(1)	27(2)	∞ *2e4*	0/15
DE QM	**45**(29)	**29**(6)	**19**(2)^{*2}	**26**(2)	∞ *2e4*	0/15
DE GP	87(45)	61(6)	41(4)	55(23)	∞ *2e4*	0/15
DE EFFX	75(57)	57(11)	43(6)	59(3)	∞ *2e4*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f5	*2.5e+2*:19	*1.6e+2*:34	*1.0e-8*:41	*1.0e-8*:41	*1.0e-8*:41	15/15
DE	**5.2**(4)	27(10)	122(13)	122(23)	122(15)	15/15
DE RFor	5.7(8)	20(6)	66(8)	66(4)	66(7)	15/15
DE QM	5.3(3)	29(8)	102(42)	102(43)	102(23)	15/15
DE GP	5.2(6)	27(11)	125(23)	125(11)	125(14)	15/15
DE EFFX	6.0(4)	**17**(2)	**62**(15)	**62**(12)	**62**(15)	6/7

#FEs/D	0.5	1.2	3	10	50	#succ
f6	*2.5e+5*:16	*6.3e+4*:43	*1.6e+4*:62	*1.6e+2*:353	*1.6e+1*:1078	15/15
DE	29(20)	39(28)	38(21)	∞	∞ *2e4*	0/15
DE RFor	24(10)	22(12)	27(10)	∞	∞ *2e4*	0/15
DE QM	24(22)	**21**(10)	**21**(7)	**59**(6)	∞ *2e4*	0/15
DE GP	33(30)	35(12)	36(13)	∞	∞ *2e4*	0/15
DE EFFX	**21**(22)	24(11)	25(16)	∞	∞ *2e4*	0/6

#FEs/D	0.5	1.2	3	10	50	#succ
f7	*1.0e+3*:11	*4.0e+2*:39	*2.5e+2*:74	*6.3e+1*:319	*1.0e+1*:1351	15/15
DE	**2.4**(5)	33(38)	76(46)	∞	∞ *2e4*	0/15
DE RFor	**2.4**(5)	16(17)	35(31)	181(169)	∞ *2e4*	0/15
DE QM	**2.4**(2)	**11**(9)	**16**(8)*	**21**(4)^{*4}	∞ *2e4*	0/15
DE GP	**2.4**(1)	19(15)	65(39)	∞	∞ *2e4*	0/15
DE EFFX	**2.4**(5)	18(15)	48(47)	∞	∞ *2e4*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f8	*4.0e+4*:19	*2.5e+4*:35	*4.0e+3*:67	*2.5e+2*:231	*1.6e+1*:1470	15/15
DE	119(45)	97(39)	124(20)	1298(1799)	∞ *2e4*	0/15
DE RFor	78(39)	58(10)	59(6)	39(3)	∞ *2e4*	0/15
DE QM	**67**(11)	**44**(8)*	**42**(5)^{*4}	**28**(5)^{*4}	∞ *2e4*	0/15
DE GP	138(48)	107(10)	134(17)	∞	∞ *2e4*	0/15
DE EFFX	93(34)	81(26)	127(9)	1295(1690)	∞ *2e4*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f9	*1.0e+2*:357	*6.3e+1*:560	*4.0e+1*:684	*2.5e+1*:756	*1.0e+1*:1716	15/15
DE	∞	∞	∞	∞	∞ *2e4*	0/15
DE RFor	44(30)	31(19)	28(8)	31(1)	∞ *2e4*	0/15
DE QM	**25**(16)^{*3}	**18**(11)^{*3}	**17**(1.0)^{*3}	**18**(20)^{*3}	∞ *2e4*	0/15
DE GP	∞	∞	∞	∞	∞ *2e4*	0/15
DE EFFX	∞	∞	∞	∞	∞ *2e4*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f10	*1.6e+6*:15	*1.0e+6*:27	*4.0e+5*:70	*6.3e+4*:231	*4.0e+3*:1015	15/15
DE	30(26)	60(99)	555(687)	∞	∞ *2e4*	0/15
DE RFor	25(31)	37(37)	321(250)	∞	∞ *2e4*	0/15
DE QM	19(24)	**22**(15)	**95**(100)	∞	∞ *2e4*	0/15
DE GP	26(40)	50(54)	499(478)	∞	∞ *2e4*	0/15
DE EFFX	**13**(12)	39(28)	631(542)	∞	∞ *2e4*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f11	*4.0e+4*:11	*2.5e+3*:27	*1.6e+2*:313	*1.0e+2*:481	*1.0e+1*:1002	15/15
DE	**4.2**(6)	5.5(6)	∞	∞	∞ *2e4*	0/15
DE RFor	**4.2**(4)	5.9(8)	∞	∞	∞ *2e4*	0/15
DE QM	**4.2**(4)	4.4(6)	**421**(913)	∞	∞ *2e4*	0/15
DE GP	**4.2**(3)	**4.4**(4)	433(304)	∞	∞ *2e4*	0/15
DE EFFX	**4.2**(2)	5.1(5)	905(784)	∞	∞ *2e4*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f12	*1.0e+8*:23	*6.3e+7*:39	*2.5e+7*:76	*4.0e+6*:209	*1.0e+1*:1042	15/15
DE	89(71)	108(39)	120(12)	81(9)	∞ *2e4*	0/15
DE RFor	50(39)	64(30)	66(9)	41(4)	∞ *2e4*	0/15
DE QM	**50**(30)	**54**(14)	**48**(7)^{*2}	**28**(27)^{*2}	∞ *2e4*	0/15
DE GP	77(32)	112(47)	114(23)	86(6)	∞ *2e4*	0/15
DE EFFX	59(49)	89(20)	104(28)	77(7)	∞ *2e4*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f13	*1.6e+3*:28	*1.0e+3*:64	*6.3e+2*:79	*4.0e+1*:211	*2.5e+0*:1724	15/15
DE	59(30)	100(19)	157(22)	∞	∞ *2e4*	0/15
DE RFor	43(32)	64(20)	95(23)	∞	∞ *2e4*	0/15
DE QM	**29**(17)	**26**(2)^{*2}	**32**(3)^{*4}	**46**(5)^{*4}	∞ *2e4*	0/15
DE GP	49(33)	91(21)	153(29)	∞	∞ *2e4*	0/15
DE EFFX	36(15)	94(16)	144(25)	∞	∞ *2e4*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f14	*2.5e+1*:15	*1.6e+1*:42	*1.0e+1*:75	*1.6e+0*:219	*6.3e-4*:1106	15/15
DE	100(74)	153(46)	172(96)	∞	∞ *2e4*	0/15
DE RFor	89(64)	88(37)	97(23)	72(7)	∞ *2e4*	0/15
DE QM	**83**(133)	**79**(86)	**66**(9)	**56**(2)	∞ *2e4*	0/15
DE GP	92(81)	141(57)	167(55)	∞	∞ *2e4*	0/15
DE EFFX	95(55)	123(61)	157(36)	∞	∞ *2e4*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f15	*6.3e+2*:15	*4.0e+2*:67	*2.5e+2*:292	*1.6e+2*:846	*1.0e+2*:1671	15/15
DE	52(31)	52(14)	34(4)	348(591)	∞ *2e4*	0/15
DE RFor	33(26)	31(13)	19(2)	15(4)	∞ *2e4*	0/15
DE QM	30(24)	**19**(9)	**8.8**(2)^{*4}	**8.7**(2)^{*3}	∞ *2e4*	0/15
DE GP	29(15)	48(21)	30(6)	177(260)	∞ *2e4*	0/15
DE EFFX	**26**(26)	41(15)	29(8)	115(64)	∞ *2e4*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f16	*4.0e+1*:26	*2.5e+1*:127	*1.6e+1*:540	*1.6e+1*:540	*1.0e+1*:1384	15/15
DE	**3.9**(4)	174(162)	∞	∞	∞ *2e4*	0/15
DE RFor	4.1(5)	289(139)	∞	∞	∞ *2e4*	0/15
DE QM	4.1(4)	188(128)	∞	∞	∞ *2e4*	0/15
DE GP	4.6(3)	250(99)	∞	∞	∞ *2e4*	0/15
DE EFFX	4.0(3)	**143**(163)	∞	∞	∞ *2e4*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f17	*1.6e+1*:11	*1.0e+1*:63	*6.3e+0*:305	*4.0e+0*:468	*1.0e+0*:1030	15/15
DE	17(18)	50(22)	29(7)	41(25)	∞ *2e4*	0/15
DE RFor	14(21)	**25**(18)	**17**(12)	**19**(8)	**47**(53)	6/15
DE QM	**12**(7)	42(25)	24(12)	28(11)	54(63)	5/15
DE GP	18(29)	41(36)	29(13)	42(15)	∞ *2e4*	0/15
DE EFFX	13(36)	31(19)	29(8)	37(23)	∞ *2e4*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f18	*4.0e+1*:116	*2.5e+1*:252	*1.6e+1*:430	*1.0e+1*:621	*4.0e+0*:1090	15/15
DE	18(11)	31(13)	43(30)	∞	∞ *2e4*	0/15
DE RFor	12(12)	**18**(9)	**19**(4)	**21**(3)	∞ *2e4*	0/15
DE QM	**12**(12)	23(16)	26(15)	38(11)	**36**(42)	7/15
DE GP	16(10)	27(12)	39(27)	483(234)	∞ *2e4*	0/15
DE EFFX	12(10)	29(10)	39(18)	478(491)	∞ *2e4*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f19	*1.6e-1*:2.5e5	*1.0e-1*:3.4e5	*6.3e-2*:3.4e5	*4.0e-2*:3.4e5	*2.5e-2*:3.4e5	3/15
DE	∞	∞	∞	∞	∞ *2e4*	0/15
DE RFor	∞	∞	∞	∞	∞ *2e4*	0/15
DE QM	∞	∞	∞	∞	∞ *2e4*	0/15
DE GP	∞	∞	∞	∞	∞ *2e4*	0/15
DE EFFX	∞	∞	∞	∞	∞ *2e4*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f20	*1.6e+4*:38	*1.0e+4*:42	*2.5e+2*:62	*2.5e+0*:250	*1.6e+0*:2536	15/15
DE	61(18)	73(25)	145(26)	∞	∞ *2e4*	0/15
DE RFor	38(13)	48(14)	72(7)	**118**(89)	∞ *2e4*	0/15
DE QM	**22**(10)	**28**(3)*	**37**(2)^{*4}	354(557)	∞ *2e4*	0/15
DE GP	48(21)	66(37)	133(20)	∞	∞ *2e4*	0/15
DE EFFX	47(20)	71(5)	140(31)	∞	∞ *2e4*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f21	*6.3e+1*:36	*4.0e+1*:77	*4.0e+1*:77	*1.6e+1*:456	*4.0e+0*:1094	15/15
DE	185(116)	314(216)	314(392)	328(285)	∞ *2e4*	0/15
DE RFor	172(90)	347(310)	347(233)	∞	∞ *2e4*	0/15
DE QM	**136**(137)	**180**(123)	**180**(65)	**143**(269)	**87**(66)	3/15
DE GP	146(93)	252(264)	252(154)	322(372)	∞ *2e4*	0/15
DE EFFX	154(36)	308(224)	308(151)	322(177)	∞ *2e4*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f22	*6.3e+1*:45	*4.0e+1*:68	*4.0e+1*:68	*1.6e+1*:231	*6.3e+0*:1219	15/15
DE	130(66)	204(73)	204(176)	189(204)	57(37)	4/15
DE RFor	150(54)	206(71)	206(118)	140(76)	58(83)	4/15
DE QM	**113**(50)	**129**(61)	**129**(49)	**88**(79)	**35**(38)	6/15
DE GP	135(85)	202(69)	202(41)	137(90)	57(61)	4/15
DE EFFX	147(67)	179(184)	179(73)	159(121)	58(107)	4/15

#FEs/D	0.5	1.2	3	10	50	#succ
f23	*6.3e+0*:29	*4.0e+0*:118	*2.5e+0*:306	*2.5e+0*:306	*1.0e+0*:1614	15/15
DE	**2.3**(2)	**27**(16)	293(370)	293(282)	∞ *2e4*	0/15
DE RFor	**2.3**(2)	39(73)	269(264)	269(341)	∞ *2e4*	0/15
DE QM	**2.3**(2)	40(49)	**165**(90)	**165**(109)	∞ *2e4*	0/15
DE GP	**2.3**(2)	30(14)	272(381)	272(220)	∞ *2e4*	0/15
DE EFFX	**2.3**(2)	30(19)	916(850)	916(1226)	∞ *2e4*	0/15

#FEs/D	0.5	1.2	3	10	50	#succ
f24	*2.5e+2*:208	*1.6e+2*:918	*1.0e+2*:6628	*6.3e+1*:9885	*4.0e+1*:31629	15/15
DE	42(9)	324(387)	∞	∞	∞ *2e4*	0/15
DE RFor	35(14)	103(110)	∞	∞	∞ *2e4*	0/15
DE QM	**25**(6)	**20**(12)*	∞	∞	∞ *2e4*	0/15
DE GP	39(8)	∞	∞	∞	∞ *2e4*	0/15
DE EFFX	41(8)	∞	∞	∞	∞ *2e4*	0/15

Expected running time (ERT in number of function evaluations) divided by the respective best ERT measured during BBOB-2009 in dimension 20. The ERT and in braces, as dispersion measure, the half difference between 90 and 10%-tile of bootstrapped run lengths appear for each algorithm and run-length based target, the corresponding best ERT (preceded by the target Δf-value in *italics*) in the first row. #succ is the number of trials that reached the target value of the last column. The median number of conducted function evaluations is additionally given in *italics*, if the target in the last column was never reached. Entries, succeeded by a star, are statistically significantly better (according to the rank-sum test) when compared to all other algorithms of the table, with $p = 0.05$ or $p = 10^{-k}$ when the number k following the star is larger than 1, with Bonferroni correction by the number of instances. A ↓ indicates the same tested against the best algorithm of BBOB-2009. Best results are printed in bold.