Struct Multidisc Optim (2008) 35:315–326
DOI 10.1007/s00158-007-0132-4

RESEARCH PAPER

Metamodeling by symbolic regression and Pareto simulated annealing

Erwin Stinstra · Gijs Rennen · Geert Teeuwen

Received: 3 March 2006 / Revised: 30 January 2007 / Accepted: 19 March 2007 / Published online: 3 May 2007

Abstract The subject of this paper is a new approach to symbolic regression. Other publications on symbolic regression use genetic programming. This paper describes an alternative method based on Pareto simulated annealing. Our method is based on linear regression for the estimation of constants. Interval arithmetic is applied to ensure the consistency of a model. To prevent overfitting, we merit a model not only on predictions in the data points, but also on the complexity of a model. For the complexity, we introduce a new measure. We compare our new method with the Kriging metamodel and against a symbolic regression metamodel based on genetic programming. We conclude that Pareto-simulated-annealing-based symbolic regression is very competitive compared to the other metamodel approaches.

Keywords Approximation · Metamodeling
Pareto simulated annealing · Symbolic regression

1 Introduction

In many scientific areas, it is important to relate output of a system to its input. Getting insight into the sensitivities of outputs with respect to inputs or finding the best input values with respect to the outputs may require a large number of system evaluations. On many occasions, the number of evaluations that can be used to do this is limited. Therefore, metamodels (also called approximating models, compact models or response-surface models) are used to approximate the behavior of the system.

A typical example of such a process can be found in virtual prototyping tools, like finite element analysis (FEA) software. These tools are used to predict physical properties of a product. Such simulations are often very time-consuming. A simulation run that takes hours is no exception, so the amount of experiments may be very limited. Parameters that are input to the simulation software are referred to as design parameters. Parameters that quantify the simulated physical behavior are called response parameters. In simulation-based optimization, values for the design parameters are searched, such that the response parameters are in some sense optimal. See, e.g., Barthelemy and Haftka (1993), Alexandrov et al. (1998), Jones et al. (1998), and Simpson et al. (2004).

In literature, many types of metamodels are used. The most obvious choice is the first or second degree polynomial model (see Montgomery 1984). More complicated model structures include rational functions (see Cuyt and Verdonk 1992), Kriging models (see Sacks et al. 1989), support vector regression machines (see Vapnik et al. 1997), neural network structures like RBFs (see Powell 1987), and symbolic regression (see Koza 1992). Comparative studies on metamodel types can be found in Jin et al. (2001, 2003).

Symbolic regression searches for the best metamodel by systematically altering operators in a set of explicit formulas. These formulas are represented using a tree structure. The best tree structure is found using a combinatoric optimization technique, An important advantage of symbolic regression is its flexibility. A metamodel

E. Stinstra (✉) · G. Teeuwen
Center for Quantitative Methods,
P.O. Box 414, 5600 AK,
Eindhoven, The Netherlands
e-mail: stinstra@cqm.nl

G. Rennen
Tilburg University,
P.O. Box 90153, 5000 LE,
Tilburg, The Netherlands

that is found by symbolic regression is not restricted to a certain class of functions. Symbolic regression models can be used for extrapolation since they capture the underlying physics, which is an advantage in comparison to, e.g., Kriging models. Further, symbolic regression models are usually more interpretable, which makes it easier to extract new knowledge from symbolic regression models. To the best of our knowledge, symbolic regression models are always used in combination with genetic programming [e.g., see Koza (1992), and for an example in engineering, see Gambling et al. (2001)]. This paper introduces a symbolic regression approach that is not based on genetic programming, but on simulated annealing (see Aarts and Korst 1989). This optimization technique usually requires less iterations than genetic algorithms to converge to an optimal solution. The algorithm is extended with a number of new concepts. Complexity control is used to ensure interpretability of the resulting model. Pareto simulated annealing is used to find not only one best model, but a range of models on a best fit/complexity trade-off curve (see also Smits and Kotanchek 2004). The best-fit metamodel may not be the metamodel that eventually will be chosen, since it may be less intuitive than metamodels that have a worse fit. Further, linear regression (see Montgomery 1984) is used to fit coefficients in the formulas. Finding the best coefficient values is often recognized as a difficult problem in symbolic regression. A binary tree data structure is used for fast neighborhood exploration. The resulting approximating model is compared to Kriging models and to genetic-programming-based symbolic regression on a number of typical cases from simulation based optimization.

The remainder of this paper is organized as follows: In Section 2, we describe the basic approach. Section 3 describes a number of extensions to the basic algorithm. In Section 4, we test our approach on several cases and compare the results to other metamodel types. In Section 5, we draw conclusions.

Table 1 The operators that are used in the transformation functions

Operator	Formula	Operator	Formula
Sum	$R+L$	RootB	$\sqrt{R}$
DiffA	$L-R$	LogA	$\log(L)$
DiffB	$R-L$	LogB	$\log(R)$
Mult	$L\cdot R$	ExpA	e^L
DivA	L/R	ExpB	e^R
DivB	R/L	SinA	$\sin(L)$
PowerA	L^R	SinB	$\sin(R)$
PowerB	R^L	Left	L
RootA	$\sqrt{L}$	Right	R

L refers to the value of the left subtree, and R refers to the value of the right subtree.

2 Symbolic regression approach

Symbolic regression is a technique for finding the best model in a very large class of candidates. The candidates are explicit symbolic formulas. In this section, we first describe the set of all possible approximating models. After that, we describe how to find the best approximating model. The approach assumes that all simulation data are already generated.

2.1 Model structure

The set of functions in which we search for the best metamodel consists of all functions that can be described as a linear combination of transformation functions. Each transformation function defines a transformation of the original parameters into a transformed parameter, using operators like addition, subtraction, multiplication, division, exponents, sines, and cosines. Thus, an approximating model can be written as:

$$f(x_1,\ldots,x_n)=\sum_i \alpha_i g_i(x_1,\ldots,x_n), \tag{1}$$

where x_i is the ith input parameter and α_i is the multiplier of transformation function $g_i(x)$. Suppose for example, we have a problem with three inputs, x_1, x_2, and x_3, with the following approximating model:

$$f(x_1,x_2,x_3)=0.341+1.231\sin\left(x_1+\frac{x_2}{x_3}\right)-2.114\times\exp(x_3). \tag{2}$$

Once we know the transformation functions, the coefficients α_i can be calculated using linear regression. The problem of finding suitable transformation functions g_i, which lead to a model that fits the data well and is not too complex, remains.

Each transformation function can be represented by a binary tree, consisting of nodes that represent (binary or unary) operators, input parameters, or constants. The operators that we can select for a transformation tree are

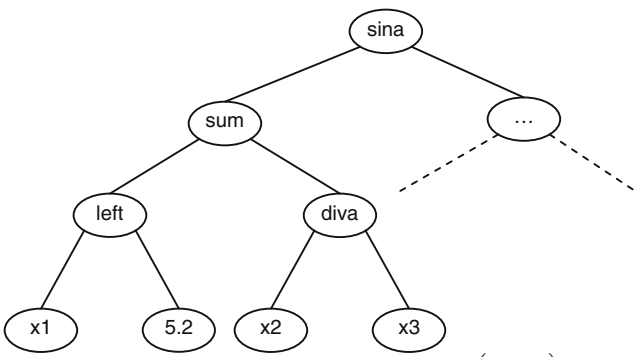

Fig. 1 Example of the transformation function $\sin\left(x_1+\frac{x_2}{x_3}\right)$

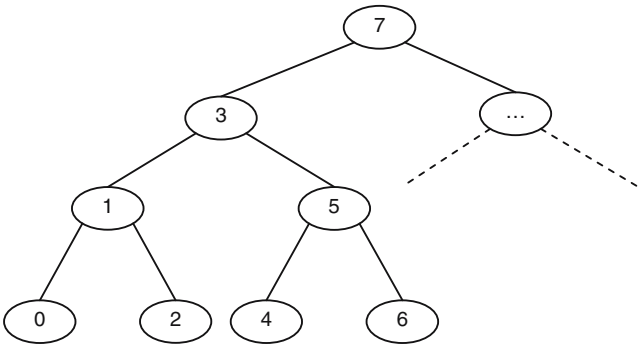

Fig. 2 The data structure of the model tree

listed in Table 1. A root node can only be a constant or an input parameter; other nodes can only be operators.

The second transformation function of example (2) is depicted in Fig. 1.

To make reference to a node in the parse tree easier, we assign an index to each node. The assignment is done from left to right as depicted in Fig. 2. This numbering has the great advantage that we can distinguish between function and terminal nodes just by looking at the index. All function nodes have an odd numbered index and all terminal nodes an even numbered index. Since terminal nodes have a distinct application in our algorithm (they can only contain variables or constants), this speeds up the algorithm.

2.2 Finding the best transformation functions

In this section, we describe how to find good transformation functions that compile the metamodel. First, we describe the basic algorithm. Then, we zoom into details on data representation, the quality aspects of the model that we want to optimize, and how they are calculated. The simulated-annealing-based algorithm is formulated in algorithm 1. For a general description of simulated annealing, see also Aarts and Korst (1989).

Algorithm 1. The Symbolic Regression algorithm based on Simulated Annealing

In the following, we describe each of these steps:

Step 0 Initialization

The transformation function i is initialized as depicted in Fig. 3. The abbreviations "r.o." and "r.v." mean random operator and random variable, respectively.

The number of transformation trees and their depth are user-defined and fixed during the algorithm. Not all trees need to have the same depth.

Step 1 Adapting the annealing temperature

We describe the changing of the annealing temperature in Step 7.

Step 2 Selecting a transformation function

A random transformation function is selected for change.

Step 3 Changing a transformation function

A transformation function is changed by randomly selecting a position in the function tree by drawing a random integer between 0 and $2^{T+1}-1$, where T is equal to the tree depth. If the selected integer is odd, its contents are set to a randomly chosen operator. If the selected integer is even, its contents are set to a random input parameter or a constant. In case a constant is set, the value of this constant is chosen randomly. An example of such a change is the following. Suppose that at some point during the algorithm a transformation function has the form depicted in Fig. 1. This tree evaluates to

$$f(x_1, x_2, x_3) = \sin\left(x_1 + \frac{x_2}{x_3}\right). \tag{3}$$

After changing the node containing the operator DIVA to ROOTA, the transformation function will become

$$f(x_1, x_2, x_3) = \sin\left(x_1 + \sqrt{x_2}\right). \tag{4}$$

Step 4 Integrity checking

Since the transformation functions are changed randomly, it is possible that a proposed transformation cannot be

```
0: Initialize: select a good initial set of transformation functions
   Repeat
1:   Adapt annealing temperature
2:   Select a transformation function
3:   Change the selected transformation function
4:   Check the integrity of the model
     If the integrity-check is OK then
5:     Calculate the coefficients of the model
6:     Evaluate the quality of the approximating model
7:     Accept the change if the model is better, or accept with probability based on
         annealing temperature when the model is worse
       If the model is changed
8:       Simplify the selected transformation function
       Endif
     Endif
9: Until stopping criterion is reached
```

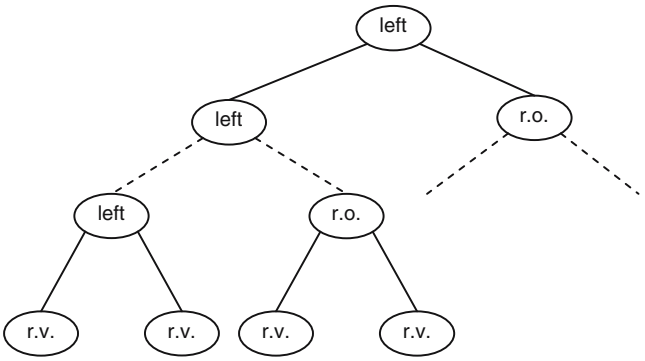

Fig. 3 The initialized transformation function

evaluated in one or more points of the domain for which the model is used. Examples are singularities caused by division by zero and square roots of a negative number. Usually, this is not the desired behavior. It is therefore necessary that we restrict the algorithm to those transformation functions that can be evaluated on the entire domain. The validity on a domain can be calculated using interval arithmetic (see Keijzer 2003).

The basic idea is the following: Given the domain of the input parameters, we can easily calculate a (sometimes conservative) domain for each node in the function tree. Using these node domains, we can easily compute whether or not a function is valid on the domain; for example, a partly negative domain in combination with a square root leads to an invalid function.

The interval arithmetic rules are denoted in Table 2. The columns lower bound and upper bound describe the formulas needed to calculate the domain of the function represented by a (sub) tree based on the domain of the left and the right subtree. The "invalid if" column describes when a (sub) tree is considered to be possibly invalid.

As mentioned, this approach may be quite conservative: Valid functions exist that are rejected on the basis of the above rules. For example, consider Fig. 4. Suppose that variable $x1$ has a range of [−1,1]. Using the rules, the domain of the subtree starting at "mult" will be estimated at [−1,1]. Therefore, this will be considered an invalid tree because the square root could be taken of a negative number. In reality, of course, the range of the subtree starting at "mult" should be [0,1], leading to a valid tree. This is not a significant problem though, since we simplify the function and check again if the function is considered invalid.

Step 5 Calculating the coefficients

After the transformation functions are determined, the problem of calculating the coefficients is a linear regression problem. Note that only the linear constants by which the transformation functions are multiplied can be calculated; the remaining constants are entered randomly during a transformation function change.

Table 2 Rules for interval arithmetic

Operator	Lower bound	Upper bound	Invalid if
Sum	lb_1+lb_2	ub_1+ub_2	
DiffA	lb_1-ub_2	ub_1-lb_2	
DiffB	lb_2-ub_1	ub_2+lb_1	
Mult	Min $\{lb_1lb_2, lb_1ub_2, ub_1lb_2, ub_1ub_2\}$	Max $\{lb_1lb_2, lb_1ub_2, ub_1lb_2, ub_1ub_2\}$	
DivA	$\min\{lb_1/lb_2, lb_1/ub_2, ub_1/lb_2, ub_1/ub_2\}$	$\max\{lb_1/lb_2, lb_1/ub_2, ub_1/lb_2, ub_1/ub_2\}$	$0 \in [lb_2, ub_2]$
DivB	$\min\{lb_2/lb_1, lb_2/ub_1, ub_2/lb_1, ub_2/ub_1\}$	$\max\{lb_2/lb_1, lb_2/ub_1, ub_2/lb_1, ub_2/ub_1\}$	$0 \in [lb_1, ub_1]$
PowerA	$min\{lb_1^{lb_2}, lb_1^{ub_2}, ub_1^{lb_2}, ub_1^{ub_2}, 0\} if\ 0 \in [lb_1, ub_1]$	$max\{lb_1^{lb_2}, lb_1^{ub_2}, ub_1^{lb_2}, ub_1^{ub_2}, 0\} if\ 0 \in [lb_1, ub_1]$	$lb_1<0$
	$min\{lb_1^{lb_2}, lb_1^{ub_2}, ub_1^{lb_2}, ub_1^{ub_2}\} otherwise$	$max\{lb_1^{lb_2}, lb_1^{ub_2}, ub_1^{lb_2}, ub_1^{ub_2}\} otherwise$	$lb_2<0$
PowerB	$\min\{lb_2^{lb_1}, lb_2^{ub_1}, ub_2^{lb_1}, ub_2^{ub_1}, 0\}$ if $0 \in [lb_2, ub_2]$	$\max\{lb_2^{lb_1}, lb_2^{ub_1}, ub_2^{lb_1}, ub_2^{ub1}, 0\}$ if $0 \in [lb_2, ub_2]$	
	$\min\{lb_2^{lb_1}, lb_2^{ub_1}, ub_2^{lb_1}, ub_2^{ub_1}\}$ otherwise	$\min\{lb_2^{lb_1}, lb_2^{ub_1}, ub_2^{lb_1}, ub_2^{ub_1}\}$ otherwise	$lb_1<0$
RootA	$\sqrt{lb_1}$	$\sqrt{ub_1}$	
			$lb_2<0$
RootB	$\sqrt{lb_2}$	$\sqrt{ub_2}$	
LogA	Log (lb_1)	Log (ub_1)	$lb_1<0$
LogB	Log (lb_2)	Log (ub_2)	$lb_2<0$
ExpA	Exp (lb_1)	Exp (ub_1)	
ExpB	Exp (lb_2)	Exp (ub_2)	
SinA	−1	1	
SinB	−1	1	
Left	lb_1	ub_1	
Right	lb_2	ub_2	

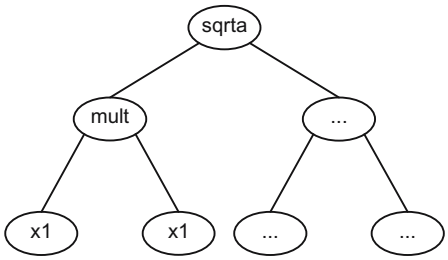

Fig. 4 Example of a model tree for which the interval arithmetic rules are too strict

Step 6 Evaluating the quality of a metamodel

The quality of the changed metamodel is evaluated on the basis of two criteria. First, the metamodel should fit well to the data in the training set. For this, we calculate the root mean squared error (RMSE):

$$RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(y^i - f(x^i)\right)^2} \tag{5}$$

in which y^i represents the output of simulation i, x^i represents the input of the model of simulation i, m the number of simulations, and $f(x)$ the approximating model. Since trying to find a perfect RMSE does not prevent us from overfitting, we use the leave-one-out cross-validation measure, which is defined as

$$CV-RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(y^i - f_{-i}(x^i)\right)^2}, \tag{6}$$

where f_{-i} denotes the metamodel obtained by fitting all simulation data except simulation i. Therefore, m times a fit is made based on $m-1$ simulations, where the remaining simulation is only used for model validation. Note that the CV-RMSE can be quickly evaluated using the projection matrix [this method uses only one calculation of a solution to a system of linear equations, whereas m solutions to a linear system of equations are needed in the straightforward method for calculating formula (6)]:

$$CV-RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(\frac{y^i - f(x^i)}{1-H_{i,i}}\right)^2}, \tag{7}$$

where

$$H = X\left(X^TX\right)^{-1}X^T. \tag{8}$$

Note that for the calculation of the CV-RMSE, we only refit the linear coefficients and not the model structure.

Further, the metamodel should be interpretable, i.e., not too complex. A measure that estimates this quality is described in Section 3.2. For now, we only consider the objective function as a user specified linear combination of the RMSE and the CV-RMSE. In Section 3.3, we describe how the approach handles multiple objectives.

Step 7 Acceptance of a change

If the change in the transformation function results in an improvement of the objective function, the change is always accepted. To avoid getting stuck in a local minimum, we sometimes have to accept a deterioration of the objective. The greater the deterioration, the smaller is the probability of acceptance. The probability of acceptance is given by:

$$P = e^{-|\Delta objective|/c}, \tag{9}$$

where Δobjective is the change in the quality of the metamodel compared to the previous iteration, and parameter c is the annealing temperature. In simulated annealing, this parameter gradually decreases for every iteration, lowering the probability of accepting big deteriorations of the objective. The general idea is that at the start of the search, we would like to have a broad look at all parts of the solution space. Thus, we have to accept relatively large deteriorations. The closer we get to the end of the search, the smaller the chance that a large deterioration will eventually lead to an improvement of the objective; hence, the smaller the probability should be that large deteriorations are accepted.

A difficulty using simulated annealing is determining a start value for annealing temperature c, because this implies that we have to quantify a "large deterioration". This is particularly difficult in symbolic regression because this means that we need to be able to tell beforehand how well the quality of a function can probably become. Therefore, we introduce the new concept of reannealing: The annealing is started with a temperature that is a percentage of the first objective value. After a number of iterations, it is checked how often a change is accepted and rejected. If too many changes are accepted, the starting temperature was too high, and hence we start the annealing again with half the temperature. If on the other hand too few changes were accepted, the temperature was too low and we start with double the temperature. We continue this until a suitable temperature is found.

Step 8 Simplification

Obviously, random changes in the transformation functions may lead to functions that can be simplified. Symbolic simplification is very time-consuming though. Therefore, only some simple rules are checked after each iteration:

1. If a function node evaluates to a constant, the subtree starting at that node is replaced by a constant term. The value of the constant term is equal to the constant output of the function node.

2. If the left and right arguments of a function node are equal, we can make replacements for certain functions. For example,

- $f(x)+f(x)$ is replaced by $2\,f(x)$.
- $f(x)-f(x)$ is replaced by 0.
- $f(x)\,/\,f(x)$ is replaced by 1.
- $f(x)\,f(x)$ is replaced by $f^{2}(x)$.

The simplification is only carried out when the changed solution is accepted by the simulated annealing algorithm. The simplification rules are applied as long as changes are made during the previous application of the simplification rules. Whether this simplification step has a positive effect on the search depends very much on the case. The effect may be negative since simplifications make the transformation functions smaller, causing a change to be relatively large on average. Future research will show if this step is beneficial for the search.

Step 9 Stopping criterion

If the maximum number of iterations is reached, or if the approximation is evaluated as good enough, the algorithm stops. The maximum number of iterations is typically 200,000.

3 Extensions to the basic algorithm

3.1 Reasons for extension

As mentioned in the introduction, metamodels are often used in situations where getting data with the real model is too time consuming. Fitting functions on these data sets implies the risk that we find a function that only fits well on this particular data set but does not describe the general behavior of the system. This problem is called overfitting. Using the CV-RMSE instead of only the RMSE to measure the quality of the metamodel already reduces the risk of overfitting, but it does not remove it.

Furthermore, to improve interpretability of the metamodel, it is usually wise to limit the depth of the function trees. However, this has two drawbacks. Firstly, the function tree depth does not always represent the complexity of a function. For instance, a sine operator is often considered more difficult to interpret than a plus-operator although they both require one function node. Secondly, the upper bound on the tree depth has to be set by the user. It is difficult to set this value beforehand.

The extensions we describe in the next subsections are meant to reduce overfitting and improve interpretability. First, we introduce a complexity measure, which aims to measure interpretability and penalize possible overfitting. Second, instead of putting an upper bound on the complexity value we add minimization of complexity as a second objective. To deal with the two objectives, we use an algorithm based on Pareto simulated annealing.

3.2 Complexity measure

The basic idea of the complexity measure is that the complexity of a model is measured by the minimal degree of the polynomial necessary to approximate the model with a precision ε in a set of points S. The idea behind this is that overfitted metamodels often have high oscillation. This makes them difficult to approximate, and results in a polynomial of a high degree to obtain the required precision.

To determine the necessary complexity of the approximating polynomial, we use the function tree representation. We calculate the complexity in every node from the terminal nodes to the root node. We use calculation rules for the binary and nested operators. For the unary operators, we calculate the complexity by successively approximating the function by a polynomial of increasing degree. This method is based on Garishina and Vladislavleva (2004). However, we use different calculation rules and a different method for approximation of unary operators. The main difference is in the method for approximation of unary operators. In the measure of Garishina and Vladislavleva, a

Table 3 Complexity rules for binary and nested operators

Operator	Complexity rule
$f(x)+g(y)$	Max {compl($f(x)$), compl($g(y)$)}
$f(x)-g(y)$	Max {compl($f(x)$), compl($g(y)$)}
$f(x)*g(y)$	compl($f(x)$)+compl($g(y)$)
$f(x)/g(y)$	compl($f(x)$)+compl($1/g(y)$) (special case of $f(x)$ * $g(y)$)
$f(g(y))$	compl($f(x)$) * compl($1/g(y)$)
$f(x)^{\text{const}}$	$compl\,(f(x))*const$ *if* $const \geq 0 \wedge const \in \mathrm{N}$ (special case of $f(x)$ * $g(y)$) compl($f(x)$) * compl (y^{const}) otherwise, where y is a variable with the same range as $f(x)$
$\text{const}^{f}(x)$	compl($f(x)$) * compl ($const^{y}$) with y a variable with the same range as $f(x)$
$f(x)^{g(y)}$	Use the relation $f(x)^{g(y)}=\exp(g(y)*\ln(f(x)))$ in combination with the previous rules

polynomial with increasing degree is fitted in a fixed number of points until the approximation of the polynomial to the unary operator in these points is accurate enough. In our approach, we apply polynomial interpolation through an increasing number of training points until the accuracy in a certain test set is high enough. Other differences are that we use some different calculation rules and take the number of nodes in a function explicitly into account to make the measure more discriminative.

To determine the complexity of the terminal and binary function nodes, we use a set of calculation rules. The complexity of a constant node is zero, and the complexity of a variable node is one. The rules for binary and nested operators are derived from relations between polynomial interpolations and are listed in Table 3.

Take for example the formula $h(x) = x^6 + 6x$. This formula can be written as $f(x) + g(x)$ with $f(x) = x^6$, and $g(x) = 6x$. With the sixth rule, we can find that $f(x)$ has complexity 6 and $g(x)$ complexity 1. The first rule now tells us that $h(x)$ has complexity 6.

The complexity rules serve as an approximation to the complexity as defined by the minimal degree of the polynomial necessary to approximate the model. To explain the above calculation rules, let us first define $P_S(f(x))$ as a polynomial that interpolates $f(x)$ in a set of points S. The function $P_S(f(x)) + P_T(g(y))$ then forms a polynomial interpolation of $f(x) + g(y)$ in the set of points $S \times T$ because the sum of two polynomials is again a polynomial. The degree of this polynomial is at most the maximum of the degrees of $P_S(f(x))$ and $P_T(g(y))$. This explains the calculation rule for addition.

The calculation rule for nested functions follows from replacing every x in $P_S(f(x))$ by $P_T(g(y))$. This gives a polynomial with a degree equal to the product of the degrees of $P_S(f(x))$ and $P_T(g(y))$.

The other rules are obtained in a similar way and will therefore not be discussed.

The complexity of a unary function node is determined in the following way. First, we determine the minimal size of the training set such that the unique polynomial interpolation through the data in the training set approximates the unary operator good enough. The polynomial interpolation is only considered on the range of the input argument of the unary operator. This range is determined by the interval arithmetic (see Section 2.2, step 3). The approximation is considered good enough if the maximum approximation error on a test set is below a user-defined threshold ε. The test set consists of a number of data points located between the interpolation points.

The algorithm to determine the complexity of a unary function node is described in algorithm 2.

Algorithm 2. Method for determination of complexity for unary operators.

The reason for sampling Chebyshev points instead of equidistant points for the training set is avoiding Runge's phenomena. Runge's phenomena states that for some functions, the approximation by a polynomial interpolation through an increasing number of equidistant points on a fixed interval gets worse when the number of points increases. However, in algorithm 2, we assume that the approximation does get better when we use more points. Fortunately, this does hold for Chebyshev points because they do not suffer from Runge's phenomena (Mathews and Fink 2004).

The procedure to determine the complexity of binary operators might seem time consuming. However, when using it in our simulated-annealing-based algorithm, it can be calculated quite efficiently. After changing one node in a function, we only have to recalculate the complexity of the nodes between (and including) the changed node and the root node of the transformation function. This limits the number of times we have to use algorithm 2 per iteration of the simulated annealing algorithm. The computation time can also be limited by setting a maximum to the value of k in algorithm 2.

3.3 Pareto simulated annealing

Now that we have determined a complexity measure, we still need a method for finding a function with a desired quality and complexity. The main problem is that functions with better fit generally have a higher complexity (overfitting). The trade-off decision between fit and complexity is difficult to make before we have any results. We regard maximizing metamodel quality and minimizing complexity as two separate objectives and use a multiobjective combinatorial optimization (MOCO) method to find multiple good solutions.

```
Initialize: Approximate the range of the input values with interval arithmetic
            Set k = 1
Repeat
  Increase k by 1
  Create a training set consisting of k Chebyshev points
  Find the interpolating polynomial of degree at most k-1
  Create a test set
Until the maximum error on the test set is below

Complexity of the unary operator is k-1
```

The use of simulated annealing in the basic algorithm makes it a natural choice to use the multiobjective extension called Pareto simulated annealing (see Czyżak and Jaszkiewicz 1998). Pareto simulated annealing does not try to find a solution that is optimal according to one objective, but instead, to find an approximation of the Pareto set. The Pareto set is the set of Pareto optimal solutions. In our situation, Pareto optimal solutions are defined by the fact that their metamodel quality cannot improve without deteriorating the complexity and vice versa.

The two main differences between "normal" simulated annealing and Pareto simulated annealing are the method for comparing two solutions and the method for determining the amount of difference in performance of two solutions.

In Pareto simulated annealing, comparing two solutions $f_1(x)$ and $f_2(x)$ leads to four possible scenarios:

- $f_1(x)$ and $f_2(x)$ are equally good: $f_1(x)$ and $f_2(x)$ are equally good on all objectives.
- $f_1(x)$ dominates $f_2(x)$: $f_1(x)$ is at least equally good as $f_2(x)$ on all objectives and better on at least one.
- $f_1(x)$ is dominated by $f_2(x)$: $f_1(x)$ is at most equally good as $f_2(x)$ on all objectives and worse on at least one.
- $f_1(x)$ and $f_2(x)$ are mutually nondominating: $f_1(x)$ is worse than $f_2(x)$ on at least one objective and better on at least one other objective.

We choose to always accept $f_2(x)$ if it dominates $f_1(x)$ or if it is equally good as $f_1(x)$. We only accept $f_2(x)$ with a certain probability if it is dominated by $f_1(x)$ or if $f_1(x)$ and $f_2(x)$ are mutually nondominating.

The acceptance probability depends among others on the amount of difference in performance of the two solutions. To determine this difference, we need to convert the performances on multiple objectives into one single measure. For this, we choose to use the dominance-based performance measure. This measure is introduced in Smith et al. (2004) for general Pareto simulated annealing and solves some drawbacks of more traditional measures like the weighted sum. The dominance-based performance measure is defined by:

$$\Delta E(f_1(x),f_2(x)) = \frac{1}{\left|A\widetilde{P}F\right|}\left(\left|A\widetilde{P}F_{f_1(x)}\right| - \left|A\widetilde{P}F_{f_2(x)}\right|\right), \tag{10}$$

with:

APF	set of solutions that approximate the Pareto front
$A\widetilde{P}F$	$APF \cup \{f_1(x),f_2(x)\}$
$A\widetilde{P}F_{f(x)}$	set of solutions in $A\widetilde{P}F$ that dominate $f(x)$.

An example of an APF is depicted in Fig. 5 by the circles. When we let the triangle depict $f_1(x)$ and the star depict $f_2(x)$, then $\left|A\widetilde{P}F\right| = 11$, $\left|A\widetilde{P}F_{f_1(x)}\right| = 5$ and $\left|A\widetilde{P}F_{f_2(x)}\right| = 2$. The value of the dominance-based performance measure thus becomes 3/11.

Therefore, the performance of a solution is measured by the percentage of solutions in $A\widetilde{P}F$ that it dominates. An advantage of this measure is that solutions that are not dominated by a solution in the current APF are always accepted, which is not always the case with other measures. Furthermore, solutions in sparsely populated areas of the APF generally have a better performance than solutions in densely populated areas. This stimulates the search of a set of solutions that is evenly spread over the APF.

The main drawback of the measure is that performance differences can be relatively high when the APF contains few points. This can result in a coarse acceptance probability distribution meaning that a slightly worse fit or complexity can result in a large reduction in acceptance probability. Smith et al. (2004) present a number of methods to alleviate this problem. The solution we choose to prevent this is to create extra points evenly spread on the attainment surface of the APF as described in Smith et al. (2004). The attainment surface of the APF is the boundary of the area containing all points dominated by the points in the APF. The points in the APF are thus on the attainment surface. In Fig. 5, the attainment surface is depicted by the black line. The extra points are created on the part of the attainment surface that lies between the solution with the highest complexity and the solution with the lowest complexity. This part of the attainment surface is depicted in Fig. 5 by the thick part. Half of the extra points are now evenly spread over the horizontal parts and the other half over the vertical parts. An important advantage of this solution is that it maintains both above described advantages of the dominance-based performance measure.

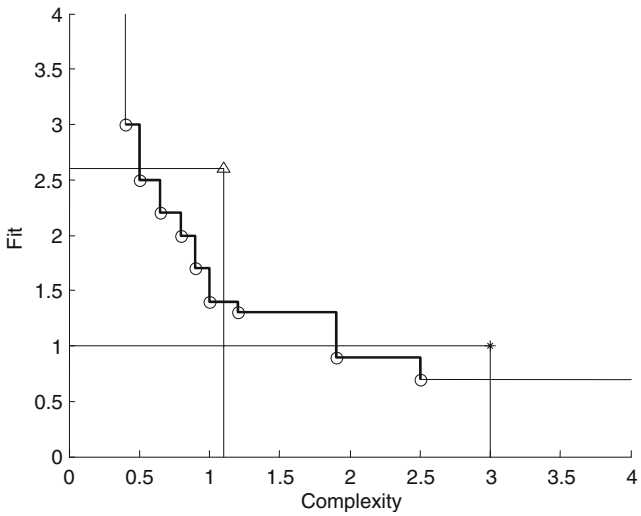

Fig. 5 Example of the APF and the attainment surface

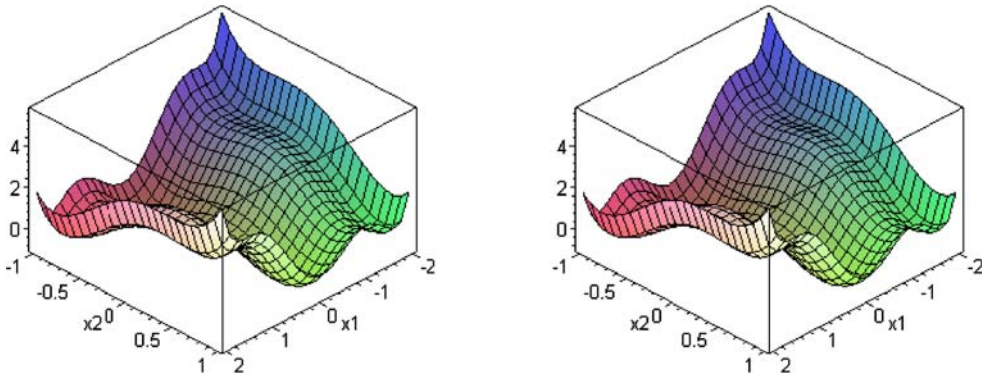

Fig. 6 The six-hump camel back function (*left*) and the meta-model (*right*)

The Pareto simulated annealing version of algorithm 1 is described in algorithm 3. Note that step 6 and step 7 are adapted and that step 9 is added. To efficiently store and update the APF, we use two vectors containing, respectively, the complexities and fits of the functions in the APF. In these vectors, the functions are ordered according to increasing complexity. For functions in the APF, this is equivalent to sorting them according to decreasing fit. With these two vectors, we can easily determine if the current solution dominates or is dominated by solutions in the APF. We only change the APF in two ways. Firstly, if some solutions in the APF are dominated by the current solution, we remove these solutions. Secondly, if no solution in the APF dominates our current solution, the current solution is added to the APF.

Algorithm 3. The Pareto simulated annealing version of the symbolic regression algorithm. The differences to algorithm 2 are noted in italics.

4 Numerical comparison to other metamodel types

In this section, we present a comparison between our suggested approach and two other metamodels approaches. First of all, we compared the results of our algorithm to the Kriging model (for details on fitting Kriging models, see Sacks et al. 1989). Next, we compared the results to symbolic regression based on genetic programming. We used the implementation of Smits and Kotanchek (2004) to compare our results.

From Sections 2 and 3, it can be noted that there are many parameters in our algorithm that may influence the numerical results. However, the reannealing concept takes care of the simulated annealing parameters. Most other parameters that we varied turned out to be insensitive with respect to the quality of the model, except the tree depth and the number of trees. We varied these parameters in the first case.

4.1 The six-hump-camel-back function

The first case is called the six-hump-camel-back function Fig. 6. This is an explicit formula, which has the advantage that we can accurately assess how the approximations compare to the real function. For the training set, we created a two-dimensional space-filling latin hypercube design (LHD) containing 30 experiments on the range $[-2, 2]\times[-1, 1]$. Using 30 experiments, it should be possible to build an accurate model. A space-filling (Maximin) LHD is often used for the approximation of deterministic simula-

```
0: Initialize: find a good initial set of transformation functions
   Repeat
1:   Adapt annealing temperature
2:   Select a transformation function
3:   Change the selected transformation function
4:   Check the integrity of the model
     If the integrity-check is OK then
5:     Calculate the coefficients of the model
6:     Evaluate the quality and complexity of the approximating model
7:     Always accept the change if the model dominates its predecessor or if it is
         equally good.
       Otherwise, accept the change with a probability based on the performance
         difference and the annealing temperature.
       If the model is accepted
8:       Simplify the selected transformation function
9:       Compare the model with the current APF and update the APF if necessary
       Endif
     Endif
10: Until stopping criterion is reached
```

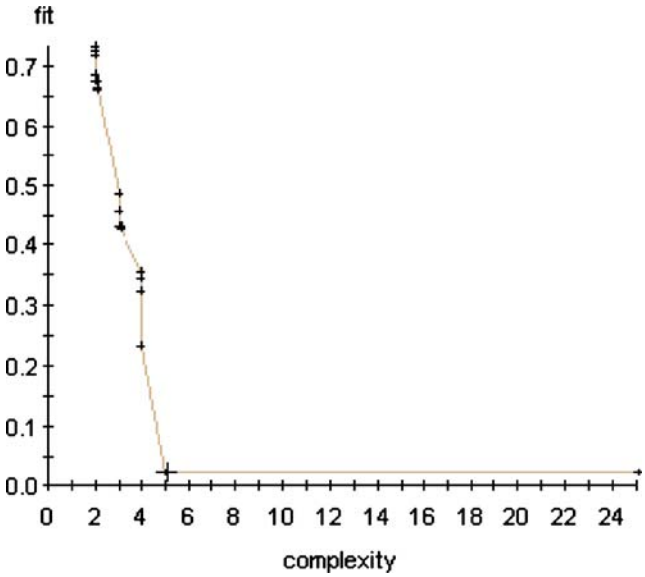

Fig. 7 A linear interpolation of the points in the APF for the six-hump camel back case after 200,000 iterations, in which the crosses represent the Pareto optimal solutions. Note that this is not the attainment surface of the APF (The fit on the vertical axis is a linear combination of CV-RMSE and RMSE)

tion models. For details on the construction of such LHDs, we refer to Morris and Mitchell (1995).

The explicit formula is given by:

$$f(x_1,x_2) = x_1^2\left(4 - 2.1x_1^2 + \frac{x_1^4}{3}\right) + x_1x_2 + x_2^2\left(-4 + 4x_2^2\right) \tag{11}$$

The best symbolic regression function that was found on this data set is selected by choosing the function with the lowest RMSE with an acceptable complexity from the Pareto front. This selection process depends on the priorities of the user. If a simple model is requested, then a worse model fit would be accepted. The chosen metamodel is given by:

$$f(x_1,x_2) = c_1 + c_2\cos(x_1) + c_3\cos(2x_1) + c_4(x_1x_2) + c_5\cos(x_2) + c_6\sin\left(c_7x_1^2\right) + c_8x_1^2, \tag{12}$$

where c_i is a constant. Note that all constants can be fit using linear regression except c_7, which was randomly entered by our algorithm. Further note that unfortunately, the formula (12) has very little resemblance to formula (11). If we would have carried out enough iterations, and the original function would fit in the model trees that we would have used (i.e., enough terms and tree depth), then the algorithm would have come up with the original function. With less complex test functions, we have rediscovered the original functional format.

Table 4 Results for the metamodels on the six-hump-camel-back case

	RMSE on the training set	RMSE on the test set
Kriging model	0	0.1119
Symbolic regression model/ SA	0.0367	0.0386
Symbolic regression model/ GP	0.114	0.141

The APF after 200,000 iterations was calculated in a few minutes on a modern PC (2.00 GHz Intel Pentium M processor). The time intensive part of each iteration is the calculation of the least squares coefficients and the cross validation statistic. This is a $o(n^3 + m^2)$ operation, where n is equal to the number of transformation functions and m is equal to the number of experiments. The APF is depicted in Fig. 7. To compare the actual quality of the metamodel, we created another test set of 30 experiments, by extending the original LHD to a new space-filling LHD consisting of 60 experiments. The results are printed in Table 4.

These results were found with a maximal tree depth of 4. Increasing the tree depth to 6 led to considerably more complex metamodels, but also even better results on the test and training set.

The second test case is also based on an exact formula. This example originates from Smits and Kotanchek (2004) of Dow Core R&D. The data consists of five input variables $(x_1,\ldots,x_5) \in [0,4]^5$, of which only the first two (x_1 and x_2) are significant. The output variable (y) consists of two parts. The first part is an explicitly known formula;

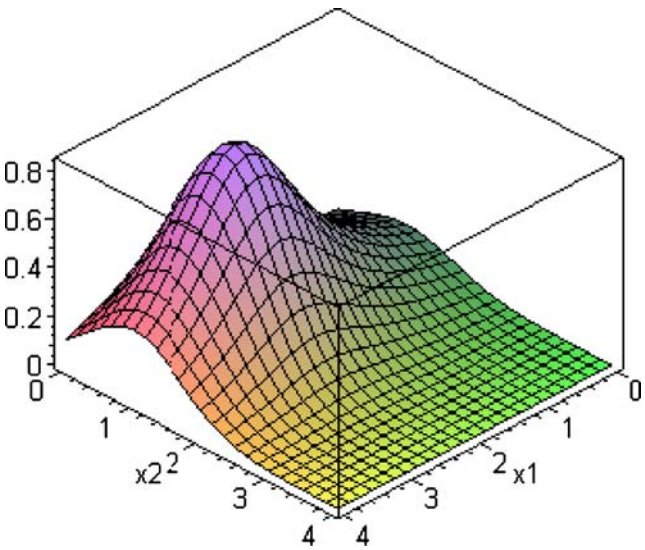

Fig. 8 The second example is based on an explicit function in x_1 and x_2, with added noise

Table 5 The training set and test set results for the four symbolic regression models and the kriging model

Metamodel	No. terms	Tree depth	Training set RMSE	Test set RMSE
Symbolic regression model/SA 1	2	4	0.111	0.090
Symbolic regression model/SA 2	2	7	0.104	0.088
Symbolic regression model/SA 3	6	4	0.056	0.052
Symbolic regression model/SA 4	6	7	0.041	0.037
Kriging			0	0.039
Symbolic regression model/GP			0.033	0.048

the second part is noise. The explicitly known formula is given by

$$y(x_1, x_2, x_3, x_4, x_5) = \frac{e^{-(x_2-1)^2}}{1.2 + (x_1 - 2.5)^2}, \tag{13}$$

to which we will refer as the Kotanchek formula. In Fig. 8, a plot of the Kotanchek formula is depicted.

We assume the noise to be uniformly distributed in the interval [−0.05, 0.05]. On 100 data points (again a space-filling LHD), four Pareto fronts were created in 200,000 iterations using different settings of the algorithm: two or six terms and a tree depth of four or seven. For each of the settings, a good function was chosen from the Pareto front by selecting a function with a low RMSE and an acceptable complexity, and for which the RMSE cannot be improved much by selecting a function with higher complexity. Then, a test set of 50 data points is generated by extending the LHD. Table 5 contains the results for this experiment. The results are compared with the Kriging model. On the test set, the result of the Kriging model is comparable to the best symbolic regression model. It is far less interpretable though, since it consists of 100 terms in which all 5 dimensions are present.

The four metamodels are depicted in Fig. 9.

We conclude that for this experiment, even the complexity used by the model with six terms and a tree depth of seven, overfitting has not occurred. Metamodels one, two and four consist of only variables x_1 and x_2. Metamodel three consists of variables x_1, x_2 and x_3.

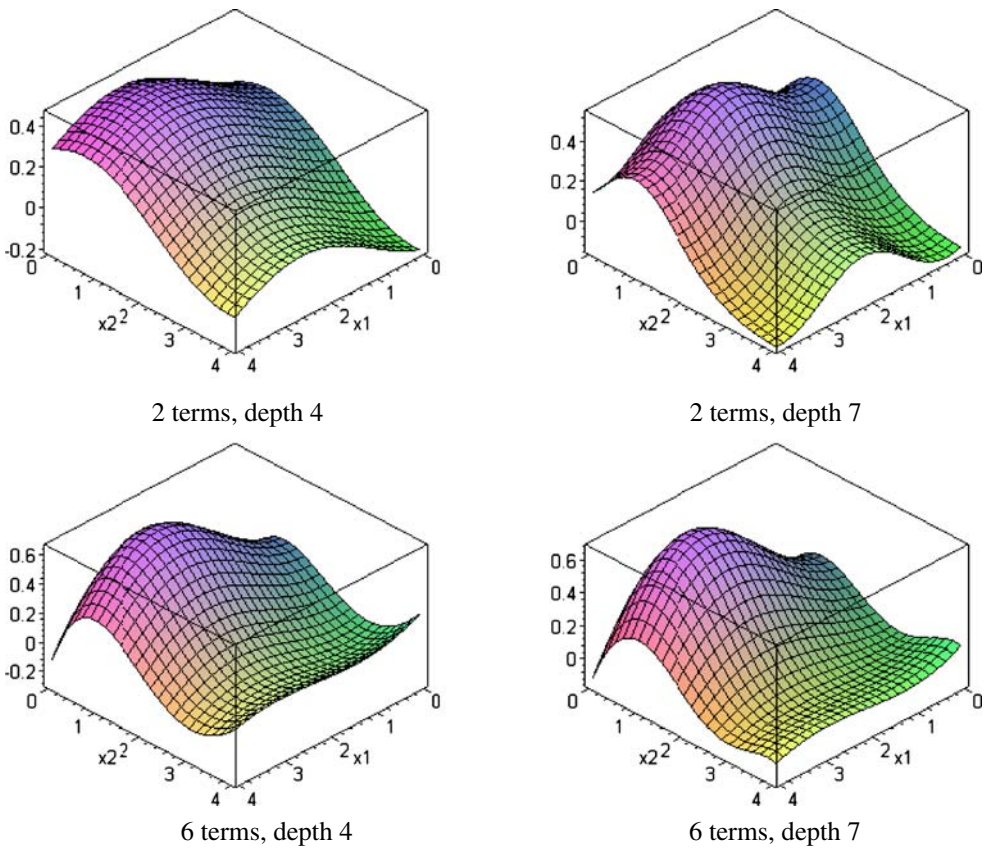

Fig. 9 The four meta-models for the Kotanchek set. The variable x_3 is fixed to 0

5 Conclusion

In this paper, we have described a simulated-annealing-based approach to symbolic regression. We have elaborated on the algorithm and data structures, and have presented the results based on two cases. We conclude that although it requires some effort to find the best symbolic regression model, the quality of the metamodels that can be found are very promising. A large advantage of symbolic regression compared to Kriging is interpretability. The complexity measure as described in this paper is a quantification of the interpretability. Although the expressions used by the symbolic regression metamodels are less complex, the fit results are comparable or better than Kriging models. Compared to genetic programming approaches, our method offers a better way of dealing with constants in the model via linear regression. In genetic programming, estimating constants is a big issue. Further, the amount of models that need to be evaluated in a simulated annealing method will on average be less compared to the tens of thousands of models that need to be evaluated in each generation of Genetic Programming. Usually, many generations are needed to come up with an acceptable model. However, further testing on a wider variety of test functions will need to be performed to validate these claims. The Pareto simulated annealing algorithm produces a list of solutions with different fit/complexity trade-offs. This gives the user flexibility to choose the metamodel that best fits his or her demands.

There are some open issues though. One significant improvement of the algorithm might be found by first applying a number of transformations of the response data. Next, the search procedure can check the metamodel on all transformation without much computational effort and select not only the best transformation functions for the input parameters, but also the best transformation for the output parameter. Another interesting extension to the algorithm could be fitting rational functions of transformation functions. This would increase the number of parameters in the metamodel that can be efficiently calculated, and thus probably increase the quality of the model. Finally, it would be beneficial to be able to dynamically alter the number of terms and the depth of the trees during the search. That way, the user would not need to make these decisions.

Acknowledgement The authors would like to acknowledge Guido Smits and Katya Vladislavleva of DOW Chemical for sharing the comparison data and the valuable discussions on symbolic regression. Further, we would like to thank the anonymous referees for their valuable comments.

References

Aarts E, Korst J (1989) Simulated annealing and boltzmann machines: a stochastic approach to combinatorial optimization and neural computing. Wiley, Chichester

Alexandrov NM, Dennis JE, Lewis RM, Torczon V (1998) A trust region framework for managing use of approximation models in optimization. J Struct Optim 15(1):16–23

Barthelemy JFM, Haftka RT (1993) Approximation concepts for optimum structural design—a review. J Struct Optim 5:129–144

Cuyt A, Verdonk B (1992) Multivariate rational data fitting: general data structure, maximal accuracy and object orientation. Numer Algorithms 3:159–172

Czyżak P, Jaszkiewicz A (1998) Pareto simulated annealing—a metaheuristic technique for multiple-objective combinatorial optimization. J Multi-Criteria Decis Anal 7(1):34–47

Gambling M, Jones RD, Toropov VV, Alvarez LF (2001) Application of optimization strategies to problems with highly non-linear response. In engineering design optimisation, proceedings of the 3rd ASMO UK/ISSMO conference on engineering design optimization. Harrogate, UK, pp 249–256 (9–10 July)

Garishina NV, Vladislavleva CJ (2004) On development of a complexity measure for symbolic regression via genetic programming. Technical report, Mathematics for industry program of the Stan Ackermans Institute, Eindhoven

Jin R, Chen W, Simpson T (2001) Comparative studies of metamodeling techniques under multiple modeling criteria. Struct Multidiscipl Optim 23(1):1–13

Jin R, Du X, Chen W (2003) The use of metamodeling techniques for optimization under uncertainty. Struct Multidiscipl Optim 25:99–116

Jones DR, Schonlau M, Welch WJ (1998) Efficient global optimization of expensive black-box functions. J Glob Optim 13:455–492

Keijzer M (2003) Improving symbolic regression with interval arithmetic and linear scaling. In genetic programming, proceedings of EuroGP'2003, vol. 2610 of LNCS. Essex, UK, pp 71–83

Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT, Cambridge, MA, USA

Mathews JH, Fink KD (2004) Numerical methods using matlab, 4th edn. Prentice Hall Pub, Upper Saddle River, NJ, USA

Montgomery DC (1984) Design and analysis of experiments. Wiley, New York

Morris MD, Mitchell TJ (1995) Exploratory designs for computer experiments. J Stat Plan Inference 43:381–402

Powell MJD (1987) Radial basis functions for multivariable interpolation: a review. In: Algorithms for approximation of functions and data. Oxford University Press, Oxford, UK, pp 143–167

Sacks J, Welch WJ, Mitchell TJ, Wynn HP (1989) Design and analysis of computer experiments. Stat Sci 4:409–435

Simpson TW, Booker AJ, Ghosh D, Giunta AA, Koch PN, Yang RJ (2004) Approximation methods in multidisciplinary analysis and optimization: a panel discussion. Struct Multidiscipl Optim 27 (5):302–313

Smith KI, Everson RM, Fieldsend JE (2004) Dominance measures for multi-objective simulated annealing. In: Proceedings of the 2004 IEEE Congress on Evolutionary Computation. IEEE Press, New Jersey, pp 23–30

Smits G, Kotanchek M (2004) Pareto-front exploitation in symbolic regression. In: Genetic programming theory and practice II. Springer, Ann Arbor, USA, pp 283–299

Vapnik V, Golowich SE, Smola A (1997) Support vector method for function approximation, regression estimation, and signal processing. In: Advances in NeuralIAProcessings Systems. MIT, Cambridge, MA, USA, pp 281–287