Neurocomputing 246 (2017) 25–44

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Improving the prediction of material properties of concrete using Kaizen Programming with Simulated Annealing

Vinícius Veloso de Melo[a,*], Wolfgang Banzhaf[b]

[a] Institute of Science and Technology (ICT), Federal University of São Paulo (UNIFESP), 12231-280, Av. Cesare Monsueto Giulio Lattes, 1211 - Jardim Santa Ines I, São José dos Campos, SP, Brazil

[b] BEACON Center for the Study of Evolution in Action, Department of Computer Science and Engineering Michigan State University, East Lansing, MI, 48864, USA

ARTICLE INFO

Article history:
Received 16 March 2016
Revised 7 November 2016
Accepted 2 December 2016
Available online 22 February 2017

Keywords:
Automatic feature engineering
Kaizen Programming
Linear regression
High-performance concrete

ABSTRACT

Predicting the properties of materials like concrete has been proven a difficult task given the complex interactions among its components. Over the years, researchers have used Statistics, Machine Learning, and Evolutionary Computation to build models in an attempt to accurately predict such properties. High-quality models are often non-linear, justifying the study of nonlinear regression tools. In this paper, we employ a traditional multiple linear regression method by ordinary least squares to solve the task. However, the model is built upon nonlinear features automatically engineered by Kaizen Programming, a recently proposed hybrid method. Experimental results show that Kaizen Programming can find low-correlated features in an acceptable computational time. Such features build high-quality models with better predictive quality than results reported in the literature.

1. Introduction

The production of concrete is an important technique in Civil constructing, and High-Performance Concrete (HPC) is a material that has been widely used in structural applications such as bridges, high-rise buildings, and pavement construction. As HPC results in good workability, high-strength and low permeability, it has replaced high-strength concrete [1] in several applications. Such concrete characteristics are directly related to long-term durability, making HPC more reliable [2]. HPC's compressive strength is considered its most important quality [3,4], while the slump flow – the diameter of slumped fresh concrete[1] – can be used as a measure for its workability [5].

The production of HPC aims at reducing the porosity within the hydrated cement, which is the major difference to conventional concrete, achieving high compressive strength or a low water-to-binder ratio [3]. In order to achieve such characteristics, several chemical and mineral ingredients must be applied to the mixture during production, which requires industrial know-how because of the interactions/relationships among them [1,6].

Aiming at developing increasingly better materials, the concrete must be produced and tested after some time has elapsed. Naturally, this step is costly and time-consuming. Researchers, therefore, have investigated methods to model the material behavior [1], which can later be optimized to give the best mixture for a particular material property. Using the putatively optimized mixture, the material can be produced and analyzed, with the measurement results inserted into the dataset as a new example. Then a new model can be created and optimized, and the cycle repeats. This way, material engineers can obtain increasingly better mixtures.

Concrete compressive strength (CCS) has been predicted by linear or non-linear regression methods [7], but given its non-linear characteristics, Machine Learning (ML) techniques – mainly Artificial Neural Networks (ANNs [8]) – have been investigated [1,9]. Slump has been successfully predicted by ANNs [5,6] and Evolutionary Computation (EC) methods [3,10].

ML techniques such as ANNs and Support Vector Machines (SVMs [11]) produce models that are considered black-boxes, i.e., they are hard (if not impossible) to understand. Therefore, EC algorithms such as Genetic Programming (GP, [12,13]), have been employed to generate smaller models by means of Symbolic Regression (SR [14–16]) that are easier to interpret [3,10,17,18]

According to Schmidt and Lipson [19]: "*Symbolic regression is a method for searching the space of mathematical expressions*

* Corresponding author.
E-mail addresses: vinicius.melo@unifesp.br, dr.vmelo@gmail.com (V. Veloso de Melo), banzhaf@msu.edu (W. Banzhaf).

[1] The slump test is a means of assessing the consistency of fresh concrete. It is used, indirectly, as a means of checking that the correct amount of water has been added to the mix. The test is carried out in accordance with BS EN 12350-2, Testing fresh concrete. http://www.concrete.org.uk/fingertips-nuggets.asp?cmd=display&id=559.

http://dx.doi.org/10.1016/j.neucom.2016.12.077
0925-2312/

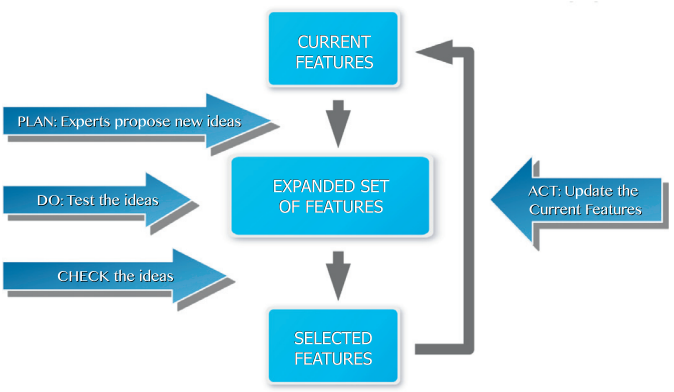

Fig. 1. Basic flowchart of KP for feature engineering [24].

while minimizing various error metrics. Unlike traditional linear and nonlinear regression methods that fit parameters to an equation of a given form, SR methods search both the parameters and the form of equations simultaneously. This process automatically forms mathematical equations that are amenable to human interpretation and help explicate observed phenomena." Therefore, the main reason for choosing SR over other regression methods such as SVMs or ANNs is interpretability, while a welcome side effect can be the automatic discovery of the exact equation that generates the response value. Such an equation may not be discovered by other methods because they only search for optimized coefficients. For instance, suppose that the desired equation is $y = \frac{x}{sin(3.1415+x)}$. SVMs and ANNs can generate a model that approximates the response surface; however, they are unable to find that ground-truth equation because their approach is not to find formulae, but to optimize coefficients.

Generally, EC algorithms try to mimic the evolution of organisms through improving solutions in a series of generations. A group of individuals (a population) generates offspring with the help of evolutionary operators called crossover (which recombines the genetic material of two parent individuals) and mutation (which adds variability in the genetic material of the offspring). Each individual is a complete solution to the problem being investigated, and its quality (fitness) is a score of how well it solves the problem (given by the objective function). Selection procedures are employed to select the parents and, later, the group of individuals that will pass to the next generation. In Genetic Programming (GP), the EC algorithms mostly used for SR, an individual is a group of instructions (such as a computer program) that must be executed to solve the problem. Such programs, as explained before, can be interpreted as mathematical equations composed of operators and operands. Operators can be arithmetic functions such as + and * or others, while operands/arguments are the constants and variables to be used in the formulae, e.g., -1.0, 3.14, x, and y. A widespread representation for GP operates on a tree-based structure, with operators as internal (non-terminal) nodes and operands as leaf (terminal) nodes of the tree. Another widespread representation (linear GP, [20]) uses a sequence of instructions as its genetic material. Generally, GP must optimize the individuals in order to obtain the best possible program.

In an attempt to develop an SR method that requires a smaller number of objective function evaluations and still produces high-quality, interpretable models, De Melo proposed a hybrid approach named Kaizen Programming (KP, [21]), based on the Kaizen [22] and the Plan-Do-Check-Act (PDCA [23]) methodologies. The basic idea is to use global search algorithms to perform automatic feature engineering and statistical methods to build models in a cycle of feature generation followed by feature selection (see Fig. 1). These steps will be explained in more detail later.

Because the features have to be used in the same model, they must complement each other; thus, the approach is truly collaborative instead of competitive. KP is guided by the importance of the partial solutions, not by the quality of the complete solution, which makes the search more efficient. KP relies on deterministic and efficient methods to build the model, while most similar methods rely on evolution.

A similar approach was proposed by Icke and Bongard [25] using a technique named Fast Function Extraction (FFX, [26]) to create features for a linear regression model. There, a hybrid algorithm uses features generated by an FFX run, which are passed onto a GP system for another step in model building. The authors hypothesized that such an approach would increase the chances of GP to succeed by letting FFX extract informative features while GP would build more complex models out of them. In this case, GP is the method supposed to solve the problem; this is exactly the opposite idea of our approach.

Given the promising results of KP shown in [21], here we extend the preliminary study of de Melo and Banzhaf [27] for feature engineering on real-world datasets of HPC concrete. The main contributions of this paper are as follows:

1. we present a novel KP implementation for feature generation applied to an SR problem. This implementation uses Simulated Annealing (SA [28]) to solve the combinatorial optimization problem;
2. we explicitly deal with multicollinearity, trying to assure that the resulting set of features, for the training set, has a maximum user-defined correlation value;
3. we use the Akaike Information Criterion (AIC, [29]) as cost (energy) function, which can be employed for model comparison and for reducing overfitting;
4. we apply a post-processing procedure to fix unrealistic prediction values;
5. we present several comparisons with EC, statistics, and ML methods on a series of experiments, including processing time.

The remainder of the paper is organized as follows. Section 2 presents many related works. Section 3 introduces Kaizen Programming for feature constructing. In Section 4, we describe the approach proposed in this work. Experimental results are shown in Section 5. Finally, Section 6 has a summary, conclusions, and future work.

2. Related work

In this section, we first present some work related to KP and then introduce related work from the literature that investigated feature construction to improve prediction of HPC compressive strength. Most work is based on EC algorithms such as GP. Unfortunately, the other authors did not use exactly the same dataset, but some important characteristics of the methods will be highlighted.

2.1. Symbolic regression methods with optimization of constants

Stinstra et al. [30] describe an SR method based on Pareto Simulated Annealing (PSA) to build meta-models. The solution is a set of transformations (new features) put together into a multiple linear regression model. The authors do not mention the constant optimization method, but it is probably the Ordinary Least Squares (OLS). In order to ensure the consistency of a model and to prevent overfitting, they use interval arithmetic and a complexity measure. The algorithm works with a tree data structure and randomly modifies a *single* feature at a time. The constants in the equations, called Ephemeral Random Constants (ERCs), are randomly generated by PSA during the optimization process. Experiments on two benchmark function problems showed that PSA was better than

Kriging (a popular statistical regression method, [31]) and GP in approximating the functions. The differences to our method will become clear in the next sections.

Fast Function Extraction (FFX, [26]) uses path-wise regularized linear learning techniques to create generalized linear models, i.e., a composition of nonlinear basis functions (formulas) with linearly-learned coefficients. The basis functions for a given complexity (tree depth) are all created and evaluated at once, and complexity increases in the course of iterations, resulting in an exponential growth in the amount of basis functions. Also, as all combinations are created at once, there is no learning to identify poor basis functions, which are greedily selected by a stepwise procedure. Many models are built with different levels of complexity and are filtered using a multi-objective Pareto domination procedure.

Arnaldo et al. [32] proposed the Multiple Regression Genetic Programming (MRGP) method, which decouples and linearly recombines a program's subexpressions via multiple linear regression on the target variable. MRGP uses Least-Angle Regression (LARS, [33]) in place of OLS to efficiently optimize the weights of *each* node in the tree. Instead of using the output of the original individual, the fitness is calculated by the output of this new multiple linear regression model. Nevertheless, the importance of the subexpressions in the new model is not heuristically exploited to generate better expressions in further generations. In released software, the authors replaced LARS by the Least Absolute Shrinkage and Selection Operator (LASSO, [34]).

GPTIPS [35] is a free open source MATLAB based software platform for symbolic data mining. GPTIPS uses a Multi-Gene Genetic Programming approach where an individual is a group of trees, instead of a single tree, which are used as features in a multiple linear regression model optimized via OLS. Multiple objectives may be treated via Pareto tournament selection, helping the search for high-quality yet low-complexity models. To avoid multicollinearity issues, it uses the Moore–Penrose pseudo-inverse instead of standard matrix inversion. Following the traditional EC approach, the multiple trees may eventually collaborate.

Castelli et al. [36] introduced the Geometric Semantic Genetic Programming with Local Search (GSGP-LS) to speed up the search process of GSGP by optimizing the values of the numerical constants present in the individual. They show that GSGP is slow but does not suffer from overfitting. GSGP-LS, on the other hand, is fast but overfits. Thus, they suggested a hybrid version (GSGP-LSH) that is fast and does not overfit. Such a hybrid runs the local search at the beginning of the optimization process (during the first g generations) for fast approximation of a high-quality solution, and then continues with the standard GSGP. A well-known issue regarding GSGP is that it grows expression trees with unlimited depth, thus ultimately generating black-boxes.

2.2. High-performance concrete prediction via symbolic regression

Regarding HPC prediction in particular, datasets of this domain are investigated in many contributions in the literature using methods from statistics, ANNs, fuzzy logic, and EC [1,4,6,7,9,37–41]. The objective of EC researchers is to not only demonstrate the effectiveness of their regression techniques but also to report the obtained formulae. For that reason, they usually focus on datasets of a single domain instead of trying multiple datasets from multiple domains.

Baykasoğlu et al. [3] applied Gene Expression Programming (GEP) to construct a single feature of a dataset with 104 instances and six input variables. It is the same regression problem investigated in this work, but with a different dataset. There is no indication of separation into folds nor training and test sets. GEP was compared with a Multilayer Perceptron neural network (MLP), a Generalized Feedforward Neural Network (GF), and Stepwise Linear Regression (LR). The authors report that the best results concerning the Mean Squared Error (MSE) were obtained by the MLP algorithm and that GEP was better than LR. While GEP achieved worse results than MLP, the authors argue that a short equation can be preferred over an ANN. Unfortunately, the authors provide no information concerning the processing time.

Tsai and Lin [10] studied a GP algorithm in which every term is weighted by a Genetic Algorithm (GA), naming it Weighted GP (WGP). This weighting results in a non-linear optimization, different from the linear optimization used in our work. When compared with the results in [3], WGP showed better prediction ability by reaching lower Root Mean Squared Error (RMSE). A relevant information is the processing time, which varied from 4018 s to 47,529 s.

Chen and Wang [17] modeled the strength of HPC with GEGA, a Grammatical Evolution combined with a Genetic Algorithm. Instead of using a binary or integer genome in GE, they used a real-coded GA arguing that it can better refine ERCs during the search. The results of RMSE on a dataset with 1140 samples show that GEGA outperformed GP, MLP, and Regression Analysis (RA). Unfortunately, there is no comparison with standard GE.

Castelli et al. [18] used a canonical GSGP implementation to model HPC CCS. Their main comparison was GSGP versus a standard GP. Results showed that GSGP was faster and more precise than GP, confirmed by a significance test. They also performed comparisons with well-known ML methods, such as LR, SVM, and MLP. The statistical analysis showed that the results obtained by GSGP were significantly better than those of the other methods.

Cheng et al. [42] proposed the Genetic Weighted Pyramid Operation Tree (GWPOT) for HPC CCS prediction. An Operation Tree (OT) is a hierarchical tree structure that represents the architecture of a mathematical formula; in fact, it is an array of integers, where each position corresponds to a specific location in the tree and each integer is mapped into a terminal or non-terminal element. The algorithm evolves multiple trees, one at a time, to complement the previous ones. Also, it works similarly to WGP [10] by adding weights to each tree node. Results were compared to those of an ANN, an SVM, and the Evolutionary Support Vector Machine Inference Model (ESIM). The authors report that GWPOT found similar or better solutions than the other methods, but provides an explicit formula.

Chen et al. [43] used a Parallel Hyper-cubic Gene Expression Programming (PHGEP) to predict the slump flow of high-performance concrete. The hyper-cube sides have subpopulations and are used to differentiate exploration from exploitation. Also, different corners have different degrees of exploration or exploitation. The results of PHGEP were compared to those of GEP, RA, and a Back-Propagation Neural Network (BPNN). The best solutions on the validation set, in terms of RMSE, were achieved by the methods in the following order: PHGEP, BPNN, GEP, and RA. These results indicate that PHGEP was better than GEP in exploring the search space and escaping from local optima.

The methods presented above evaluate hundreds of thousands of models because they use populations with hundreds of individuals running for hundreds or thousands of generations. The size of the final models, in number of nodes, is usually not reported. Also, only one of the authors reports processing time

In this paper, we propose a technique that can be efficient in solution quality, solution complexity, and processing time.

3. Kaizen Programming for feature construction

Kaizen Programming (KP), a hybrid approach based on a Kaizen event with PDCA methodology, is used to guide a continuous improvement process. Both Kaizen and PDCA have many tools to

solve problems. KP is proposed as an abstraction of these two methodologies, not a simulation. We start this section introducing KP terms, and later use traditional metaheuristics, statistics, and ML terms to simplify the reading.

A real Kaizen event has experts that propose and test *ideas* to solve a business issue. The current complete solution for the problem is known as the *standard*, which contains several ideas. By using PDCA, modifications in a business process are planned, executed, checked, and new actions are taken based on the results. The PDCA cycle is repeated until a goal, such as waste reduction, is achieved. Each action performed on the problem can be evaluated according to its effectiveness in helping to solve the issue; thus, at each cycle the team acquires more knowledge on the problem to avoid harmful actions and guide the search towards a better standard.

Three basic modules are necessary to solve problems using KP. For solving regression, KP performs (1) *Feature Generation* to expand the current feature set; (2) *Feature Selection*; and (3) *Model generation*. Algorithm 1 presents a high-level version of the KP algorithm applied here, and below we explain more details.

Algorithm 1 High-level algorithm of KP using ML terms.

1. **Generate** the initial set with nf *CurrentFeatures*
2. **Evaluate** *CurrentFeatures*
3. *BestFeatures* ← *CurrentFeatures*
4. *BestFeaturesQuality* ← *CurrentFeaturesQuality*
5. **Loop** while target is not achieved
   (a) **PLAN: Generate** nv new variations of *CurrentFeatures*
   (b) **DO: Create** *ExpandedFeatures* containing *CurrentFeatures* and their variations
   (c) **CHECK:**
   - **Build** a model on *ExpandedFeatures*, and calculate the importance of each feature
   - **Select** the nf most important features into *SelectedFeatures*
   - **Build** a new model on *SelectedFeatures* and calculate its quality

   (d) **ACT:**
   - **Update** *CurrentFeatures* if the new model is better
   - **Update** *BestFeatures* if the new model is better
   - **Restart** if necessary, generating new *CurrentFeatures*, and keeping *BestFeatures*
6. **Return** *BestFeatures*, *BestFeaturesQuality*

The first module contains the experts (data structure + procedures) that propose the ideas to solve the problem using, for instance, variations of the current best solution (the standard, which contains various ideas). For the problem investigated in this paper, an idea is an arbitrary mathematical expression used as a new feature, for instance, $feature = x^2 + (0.39 - x)$.

The second module calculates the new ideas (resulting in an actual partial solution) and joins these *trial* solutions with those from the current standard into a *single* structure of partial solutions. Afterward, it calculates the importance of each partial solution in helping to solve the problem. As the partial solutions must help each other, independent quality measures for partial solutions must not be used; clearly, a measure must consider the dependency among partial solutions. Traditional EC algorithms do not have this problem, because they work with individuals that each are complete solutions.

Finally, the third module is responsible for calculating the quality of the *complete* solution. A single complete solution is created after selecting the most important partial solutions from the set. Then, a scoring function returns the solution quality. The procedure or technique used to solve the problem must be able to use all partial solutions at once.

The objective of the PDCA cycle is that the experts provide ideas which make sense to the method that should solve the problem because it is the importance measurements that guide the search, not the quality of the complete solution. Hence, it is advisable that the methods used in the second and third modules are the same or at least related to each other (use the same criteria to choose the features to build the model), i.e., the method used to solve the problem is also employed to calculate the importance of the partial solutions. If those two modules were independent, then the ideas important to the procedure employed in the second module would not to be important to the method that actually solves the problem.

Because KP was proposed based on the Kaizen and PDCA methodologies, which focus on increasing knowledge of the system to guide the process to the goal, one must use techniques that can provide such information. For that reason, KP is a hybrid and must use efficient high-quality statistical and ML methods. Otherwise, KP would be no different from an EC algorithm, which relies only on the selection pressure to guide the search.

3.1. A toy example

We now present a toy example of how KP can be used for automatic feature engineering, with GP to provide the experts and OLS as the model building method. This example uses the algorithm presented in [27]. In the next section, we introduce the proposed modifications.

Suppose that $y = x^3 + x,\ x \in \mathbb{R}$ is our ground-truth response function. Clearly, the objective is to predict y from x. Let $\overrightarrow{x} = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7,\ 0.8, 0.9, 1.0]$ be a uniform sequence from 0.0 to 1.0. Solving a traditional linear model in the form $y = \beta_0 + \beta_1 x$, one obtains $y = -0.189 + 1.928x$, for which the calculated Root Mean Squared Error is $RMSE = 0.1345$.

Now, suppose that we are using GP for solving this regression problem and that the population has three individuals, named $ind1 = x^2$, $ind2 = sqrt(x)$, and $ind3 = log(x+1)$, while the generated offspring are $child1 = x^3$, $child2 = x$, and $child3 = sqrt(log(x+1))$. Using RMSE to calculate the fitness of these individuals, one gets: $fit(ind1) = 0.5188979$, $fit(ind2) = 0.4121862$, $fit(ind3) = 0.576697$, $fit(child1) = 0.591608$, $fit(child2) = 0.4240932$, $fit(child3) = 0.4862297$. The selection mechanism chooses individuals $ind2 = sqrt(x)$, $child2 = x$, and $child3 = sqrt(log(x+1))$ for the next population because they are the three best solutions among parents and children. Unfortunately, with such selection criterion, *child*1 is lost. Obviously, the formula present in *child*1 may be generated again in the future, or another selection criteria may choose to keep *child*1. However, the issue here is that an important part of the solution was neglected.

KP, on the other hand, works by combining partial solutions. Let us assume that KP is using GP as a global search method and with the same population (number of features, $nf = 3$).

In the PLAN step, the experts propose new ideas based on the current standard and generate the same offspring found by GP. The DO step parses these individuals, evaluates them, and creates a newly expanded dataset containing the parents and the offspring. In the CHECK step, OLS builds a model on this dataset for evaluating how each individual performs (see Table 1). The model shown in the table is $y = \beta_0 + \beta_1 ind1 + \beta_2 ind2 + \beta_3 ind3 + \beta_4 child1 + \beta_5 child2 + \beta_6 child3$, where $Intercept = \beta_0$, and the p-value for each variable regards the hypothesis test to verify whether the coefficient (β_i) is not statistically different from zero. If $p\text{-}value_i < \alpha$ (α is the significance level of the hypothesis test, traditionally 0.05), then we assume that the variable *is* important to the model.

As can be observed, the only important variables are the intercept (which is barely important), *child*1, and *child*2; thus, the reduced model after performing feature selection is $y = \beta_0 + \beta_1 child1 + \beta_2 child2$. After recalculating the coefficients, the final model is $y = 1.34e{-}16 + 1.0 * x + 1.0 * x^3$, which is the perfect

Table 1
Summary table of the multiple linear regression model for the toy problem. The Intercept is β_0 and the p-value indicates the importance of the variable to the model.

Feature	β	p-value
(Intercept)	1.18e−16	0.04
$ind1 = x^2$	2.89e−13	0.15
$ind2 = sqrt(x)$	1.34e−12	0.14
$ind3 = log(x+1)$	1.13e−12	0.15
$child1 = x^3$	1.00e+00	5.07e−54
$child2 = x$	1.00e+00	1.41e−48
$child3 = sqrt(log(x+1))$	−1.32e−12	0.14

model ($RMSE = 1.05e{-}16$), ignoring numerical issues. Thus, the optimum solution was found in a single iteration. If a termination criterion is not reached, the ACT step performs corrective actions to adjust the current standard. The new solution, built using only the most important features, replaces the *CurrentFeatures* provided it is better. A new cycle begins if the termination criterion is not reached. Also, it replaces *BestFeatures* if it is better. If a specific criterion is reached, then a restart occurs and new *CurrentFeatures* are randomly generated.

The algorithm described above (Algorithm 1) has some important characteristics. In the current implementation, the population size, which is the desired number of features (nf) has a fixed size to avoid excessive growth. If the number of important features is less than nf, then one must choose non-important features to complete the population. Also, because we are employing OLS, the number of variables in the model ($nf + nv$) must be fewer than the number of examples (rows) in the dataset. Finally, multicollinearity must be properly treated to avoid singularities when solving the linear system.

As can be noticed, the approach used in KP to combine GP and OLS takes a different path from that of other work: in KP, the main method builds the model (OLS, for local search), while GP's evolutionary operators and the data structure are used to search for features (the global search to provide good starting solutions for the local search method). On the other hand, in other approaches, the local search method is used to improve the solutions found by GP (or SA). In conclusion, while GP was used in [21,27], it is not essential for KP to work. In order to demonstrate this statement, here we use a combinatorial global optimization method not inspired by biology to search for high-quality features.

4. Kaizen Programming with Simulated Annealing

In this work, we replace tree-based GP by SA. SA works with a single solution and performs a global search by evaluating neighboring solutions stochastically. If a neighbor is better than the current solution, it replaces the latter. On the other hand, a worse solution can be accepted according to an acceptance function. Such behavior allows SA to escape from non-promising regions and explore the search space by moving to worse neighborhoods. The acceptance probability function considers a parameter T, called temperature, which decreases through SA iterations. At high temperatures, the acceptance probability tends also to be high, while low temperatures usually result in a low acceptance probability, turning SA into a hill-climbing local search approach. The *cooling factor* must be tuned so that SA has enough iterations to explore the search space. Pseudo-code for SA is shown in Algorithm 2.

The candidate generating neighbor function takes the current solution and returns a neighbor, generally a slightly modified solution. A procedure that generates a neighbor is usually called a *move*, and SA can have several of them. Using the KP methodology for feature engineering, each move is an expert and such neighbor function comprises the PLAN and DO steps, returning the *ExpandedFeatures* set. Therefore, a single SA solution is a set of features, and a neighbor has variations of the *CurrentFeatures*.

Algorithm 2 Simulated Annealing algorithm.

```
1: function SA(s0, e0, T0, Tmin, maxiter, cooling_factor)
2:     s ← s0; e ← E(s)                          ▷ Initial state, energy.
3:     sbest ← s; ebest ← e                      ▷ Initial "best" solution.
4:     T ← T0                                    ▷ Initial temperature.
5:     iter ← 0                                  ▷ Iteration count.
6:     while iter < maxiter ^AND^ T > Tmin do    ▷ Loop while permitted.
7:         snew ← neighbor(s)                    ▷ Generate a neighbor.
8:         enew ← E(snew)                        ▷ Compute its energy.
9:         Δ ← e − enew                          ▷ Compute the difference.
10:        if Δ < 0 ^OR^ random(0, 1) < exp(−Δ/T) then   ▷ Accept the
   neighbor solution.
11:            s ← snew
12:            e ← enew
13:            if enew < ebest then              ▷ Update the best solution.
14:                sbest ← snew
15:                ebest ← enew
16:            end if
17:        end if
18:        T ← T × cooling_factor                ▷ Update the temperature.
19:        iter ← iter + 1
20:    end while
21:    return sbest, ebest                       ▷ Return the best solution found.
22: end function
```

The SA energy function is the cost (objective) function to be optimized. It corresponds to the CHECK step of KP, which gives the quality of the new solution. Finally, KP's ACT step is the decision step of SA, when it decides whether to replace the current solution.

In order to have SA replace GP applied in our previous work, we modified the algorithm in the way explained in the next section.

4.1. Representation and initialization

A KP expert is a data structure and the procedures to operate on it. In [27], KP used a traditional tree-based GP to provide the experts (crossover and mutation). Here, SA works on a linear data structure instead of a tree. This replacement facilitates not only the application of SA but will be useful for other combinatorial optimization algorithms applied in the future.

Here, the programs consist of a *fixed-size* flat linear sequences of operators and operands. Initial features are random sequences of symbols from the operands and operators sets chosen for the problem. As it is not encoding a correct tree, any feature is valid, for instance:

$$feature = [x, +, x, x, x, sqrt, -].$$

In order to start with features that have a certain balance between terminals and non-terminals, and also to allow for intermediate neutral code, we are employing the following heuristic initialization:

$$\begin{aligned} feature = concatenate(& \\ & choose(operands, 2 * size), \\ & choose(concatenate(operands, operators), size), \\ & choose(operators, 2 * size)) \end{aligned}$$

where $size$ is a random integer number between 1 and max_size, $operands$ is the set of data inputs, $operators$ is the set of functions, and $choose(x, n)$ is a function that randomly chooses n elements from x allowing repetition. Different initialization methods may provide better seeds for the search. This is an important aspect to be investigated in the future.

It is important to be clear that a solution for KP is a group of features, not a single feature. Nevertheless, it is a *single* complete solution.

4.2. Parsing and evaluation

As one may have noticed, the above representation is treated in a postfix manner. Operands are pushed onto the stack, operations are applied to the elements popped from the stack, and the *resulting expression* is pushed back on the stack. If the number of elements in the stack is fewer than the operator's arity, then the operation is skipped. Finally, an empty stack returns a penalty constant to mark the feature for removal. Similar postfix stack-based representations have been used in related works such as Stack-based GP [44], PushGP [45], and more recently ellenGP [46]. Oltean et al. [47] provide a survey on such representation. Also, a related prefix version using a binary GA has been proposed in [48].

An interesting aspect of the linear representation is the occurrence of neutral code that is not executed in the current solution, but that may be enabled in a variant. Also, one may have independent blocks of code in a single feature. Thus, we first parse the feature to extract the valid code in order to avoid unnecessary evaluation and return the stack top. For instance, considering the feature in the previous example, the final stack would be:

$$stack = [[x],\ [x],\ [(sqrt(x) - x)]],$$

because the first x is not used by posterior operators, the operator '+' is skipped, the second x is not used either, and the remaining elements are used together. The stack top, which is the parsed expression ready to be evaluated, is $(sqrt(x) - x)$. Clearly, if the top is a single operand, then it is the final expression. On the other hand, the parser returns $+Infinity$ if the stack is empty (the feature has only operators).

4.3. Acceptance criterion

Canonical SA explores the search space while the temperature is high. However, when it starts the hill-climbing stage (at very low temperature, meaning that only better neighbors are accepted), it continues from the last solution, not from the best one. In fact, it does not save the best solution found (something known in EC as an "elitist" approach). We modified the traditional version to save the best solution found so far. Once the minimum temperature allowed (T_{min}) is reached, T is set to 0.0 and the algorithm recovers its best solution found so far and starts a hill-climbing (exploitation) procedure from there.

4.4. Moves (KP experts)

To generate neighbors, SA uses the concept of moves we apply three different ones:

- *Move 1:* it randomly chooses a feature from the *CurrentFeatures* set, makes a copy, and modifies several of its elements respecting their arity. Operands can only be replaced by operands, while operators can only be replaced by operators of the same arity. The number of modifications is a parameter that decreases with the temperature. It starts with a random integer from 1% to 20% of the length of the feature. When the hill-climbing stage starts, the number of modifications is set to 1.
- *Move 2:* it randomly chooses a solution from the *CurrentFeatures* set, makes a copy, and modifies several of its elements *ignoring* their arity. Therefore, an operand can be replaced by an operator with arity 2. This replacement makes sense because the inserted operator may be bypassed, modifying the data flow within the feature. The number of modifications is a parameter that decreases with the temperature. It starts with a random integer from 1% to 20% of the length of the feature. When the hill-climbing stage starts, the number of modifications is set to 1 for refining.
- *Move 3:* this expert proposes a new random feature. This move is employed because a feature has a fixed size; thus, one may have only small features in the group, but longer features may be necessary to achieve better quality solutions.

It is worth mentioning that all features have the same probability of being chosen, as all of them are important. Therefore, there is no selection pressure. The same is valid for the moves.

4.5. Dealing with multicollinearity

In [27], the DO step creates *ExpandedFeatures* which contain *CurrentFeatures* and the offspring generated by GP. Pearson correlation is calculated among all pairs of variables and those highly correlated (absolute value above *max_correlation*) are removed, keeping only the first, from left to right. As the offspring are concatenated after *CurrentFeatures*, they are usually removed when high collinearity is detected. This behavior makes sense as one intends to prioritize current features as long as they are the best found so far. However, if offspring are better, they will not be used.

In order to have a better guarantee for choosing the best features, we employ a double-check here: *ExpandedFeatures_1 = CurrentFeatures + SA neighbors* and *ExpandedFeatures_2 = SA neighbors + CurrentFeatures*. One builds two expanded models and subsequently two reduced models that contain only the selected features. Finally, the ACT step compares both reduced models with the *CurrentFeatures* model and updates the latter with the best result. This way, one not only reduces the odds of a failing OLS, but also increases the chance of finding a high-quality set of low-correlated features. This aspect of KP is very important and differentiates it from other work. While this procedure tries to guarantee low-correlated features in the training set, correlations in the test set cannot be avoided and are therefore likely higher.

4.6. Reducing prediction issues

In this section, we present the two methods employed to reduce prediction issues.

4.6.1. Optimization criterion

In [27], there is no procedure to avoid overfitting. Here, instead of minimizing the RMSE, we minimize the well-known Akaike Information Criterion (AIC) defined as:

$$\mathrm{AIC} = -2\log\mathcal{L} + 2D, \tag{1}$$

where $\mathcal{L}$ is the maximized likelihood using all available data for estimation and D is the number of variables in the model, equivalent to *nf* in our implementation. Asymptotically, minimizing AIC is equivalent to minimizing the leave-one-out cross-validation error [49]. Thus, calculating AIC just once, for each model, should be enough to compare them and select the best one.

4.6.2. Safe prediction

When dealing with a real-world dataset in which the desired output values are expected to lie in a certain range, this information can be used to prohibit the model from outputting unrealistic predictions.

Such safe predictions are bounded by considering background information on the dataset or simply information extracted from the training set output. For instance, if negative output values are not expected, the lower bound should be zero, and similarly for the upper bound. Here, the safe prediction is calculated as:

$$LB = min(training\_output) - std, \tag{2}$$

$$UB = max(training\_output) - std, \tag{3}$$

Table 2
Concrete strength dataset description (D1).

Attribute	Unit	Minimum	Maximum	Average
Cement	kg/m^3	102.00	540.00	281.17
Blast-furnace slag	kg/m^3	0.00	359.40	73.90
Fly ash	kg/m^3	0.00	200.10	54.19
Water	kg/m^3	121.80	247.00	181.57
Super-plasticizer	kg/m^3	0.00	32.20	6.20
Coarse aggregate	kg/m^3	801.00	1145.00	972.92
Fine aggregate	kg/m^3	594.00	992.60	773.58
Age of testing	Day	1	365	45.66
Concrete compressive strength	MPa	2.33	82.6	35.82

Table 3
Flow, strength, and slump dataset description (D2). All data have age=28.

Attribute	Unit	Minimum	Maximum	Average
Cement	kg/m^3	137.00	374.00	229.89
Fly ash	kg/m^3	0.00	193.00	77.97
Slag	kg/m^3	0.00	260.00	149.01
Water	kg/m^3	160.00	240.00	197.17
Superplasticizer	kg/m^3	4.40	19.00	8.54
Coarse aggregate	kg/m^3	708.00	1049.90	883.98
Fine aggregate	kg/m^3	640.60	902.00	739.60
SLUMP	cm	0.00	29.00	18.05
FLOW	cm	20.00	78.00	49.61
28-day compressive strength	MPa	17.19	58.53	36.04

$$safe\_prediction = min(UB,\ max(LB,\ prediction)), \tag{4}$$

where *std* is the standard deviation of the training set outputs, *LB* and *UB* are the lower and upper bounds, respectively, and *prediction* is the value returned by the model. Although simplistic, this safe prediction serves our purpose.

5. Experimental results

This section presents experimental results of KP using SA coupled with OLS to perform feature engineering and model fitting for a regression problem.

5.1. The datasets

Here we investigate the technique on the HPC data available from the UCI ML repository [50], first presented by Yeh [1,6]. A brief descriptive analysis of the datasets is shown in Tables 2[2] and 3. The first dataset (D1)[3] contains eight numerical attributes and 1030 examples. There is no indication of missing values.

The second dataset (D2)[4] contains seven predictive numerical attributes, three response variables, and 103 examples. There is also no indication of missing values. Besides coming from a different sample that considers only measures taken within 28 days, some researchers (for instance [17]) append this dataset to D1, using only the MPa response. Here, we decided to perform a separate analysis because many researchers investigate only D1. Moreover, one can observe later that the MPa predictions for D2 are substantially better those for D1, which could inflate the prediction quality. As D2 has three output variables, we investigate independent models to predict the slump, the flow, and the compressive strength.

[2] This descriptive table has the correct values, while the corresponding table in [27] has typos. However, the datasets are the same and did not interfere with the results.

[3] D1 was downloaded from https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength.

[4] D2 was downloaded from https://archive.ics.uci.edu/ml/datasets/Concrete+Slump+Test.

Table 4
Execution and SA parameters of KP.

Parameter	Value
Number of features (*nf*)	10
Number of variations (*nv*)	$3 \times nf$
max_size (for feature creation)	5
Cycles	1000 (D1), 1300 (D2)
max_correlation	0.4, 0.8
cooling_factor	0.95
$T_{initial}$	100.000
T_{min}	1.0
Solution quality	AIC

5.2. Implementation and configuration of the algorithms

We implemented KP in Python, with the statistical parts of KP (the OLS method, *p*-values, among others) employing the *statsmodels* package. All non-treated issues that occur during the model building (singularity, overflow, etc.) are caught as exceptions and regarded as poor quality models/features. Experiments were executed on a system environment with an Intel(R) Xeon(R) CPU E5-2620@2.00 GHz, Ubuntu Linux 14.04, Kernel 3.13.0-30-generic x86_64, gcc (Ubuntu 4.8.2-19ubuntu1) 4.8.2, Python 2.7.6, numpy 1.8.2, statsmodels 0.5.0, Java(TM) SE Runtime Environment (build 1.7.0_67-b01), Weka 3.6.11 [51].

KP is compared to the following methods from the literature that perform symbolic regression and use linear regression to optimize constants and were configured according to the original publications:

- Geometric Semantic Genetic Programming with Local Search (GSGP-LSH) [36]. Source code was generously provided by Mauro Castelli;
- Multiple Regression Genetic Programming (MRGP) [32]. Software was downloaded from the FlexGP Project website.[5]

Here we compare different methods using run time instead of the number of evaluations/models as termination criterion. This is preferable as the user can decide how much time he/she has available to wait for the method to return results. We executed KP for a specific number of cycles and used execution time to configure the number of generations of the other two methods. For the larger dataset (D1), KP took approximately 1 min to perform 1000 cycles and achieved results similar to those found in the literature. As a result, we set 1 min as the time limit and adjusted the number of cycles/generations to terminate at *approximately* that time. We are ignoring the fact that GSGP-LSH is implemented in C++, MRGP is implemented in Java, and KP is implemented in Python.

The configurations of KP, GSGP-LSH, and MRGP are shown in Tables 4, 5, and 6 respectively. For KP, we are using ten features because this gave the best results in our earlier study [27]. One can notice that we test two *max_correlation* values during optimization. With this analysis, we intend to show the effect of that parameter. Other parameters were chosen empirically, where $nf + nv <$ *number of observations* (OLS issue), the choice of cooling factor and the temperatures are common values from the literature. Also, only the four basic arithmetic functions ($+, \times, -, /(safe)$[6]) are used, as suggested in [18,36]. As GSGP-LSH encodes only these four functions, KP and MRGP were configured in the same way for a fair comparison. Furthermore, many researchers choose this function set to investigate how the algorithm performs with limited tools. One is interested in solving problems for which an exact solution is not available in the search space.

[5] http://flexgp.github.io/gp-learners/mrgp.html.

[6] *safeDiv* returns 0 if the denominator's module is less than $1e{-}10$.

Table 5
Execution and evolutionary parameters of GSGP-LSH. Zero depth indicates if the initial population can have single-node individuals.

Parameter	Value
Population size	200
Max number of generations	410 (D1), 3250 (D2)
Max linear regression generations	10
Init type	2
Crossover prob.	0.7
Mutation prob.	0.3
Max depth creation	6
Tournament size	8
Zero depth	0
Mutation step	1
Number of random constants	100 from −100.0 to 100.0
Solution quality	RMSE

Table 6
Execution and evolutionary parameters of MRGP.

Parameter	Value
Population size	200
Time limit	1 min
Crossover prob.	0.7
Mutation prob.	0.3
Tree initial max depth	6
Max depth creation	17
Tournament size	8
Solution quality	RMSE
Prediction model	Most accurate

The set of terminals (operands) is composed of the names of the datasets' features. GSGP-LSH and MRGP are allowed to evolve ERCs, but our KP implementation has no such characteristic available. Finally, for MRGP we chose the most accurate model for predicting the test data.

5.3. Evaluation

We performed experiments on three different training and test set splits: 50%/50%, 70%/30%, and 90%/10%. For each split, we executed 50 independent runs on three comparisons. The prediction quality is the RMSE on the test set. We followed the methodology used in related works.

The first experiment compares the results of KP, GSGP-LS, and MRGP. We present a summarized table containing median RMSE (test data) and the two-sided Wilcoxon's Rank Sum test comparing all methods against KP. There are also charts with the dot and violin (distribution) plots of the runs for visual comparison and a more detailed descriptive analysis of the test RMSE (see Appendix).

Later we compare the results of KP with many other methods from the literature. These methods are evolutionary, statistical, or ML, performing a hypothesis test. Here, we did not investigate whether the KP constructed features could be useful to other regression techniques. Such experiments were performed in [27], where we showed that most methods did benefit from the new features.

5.4. Results and discussion

In this section, we present and discuss our experimental results and finish with the best feature sets that were constructed for each dataset.

5.4.1. Assessing the behavior of the max_correlation parameter

In this first experiment, KP was executed with $max\_correlation = 0.4$ and $max\_correlation = 0.8$. Using this parameter value, KP guarantees that the features generated during the training have a maximum correlation. In Table 7, we show the mean of the maximum correlations obtained on the test set for the 50 independent runs.

Table 7
Mean of the maximum correlations obtained on the test set for 50 independent runs. Pct is the training percentage.

Dataset	Pct	max_correlation=0.4		max_correlation=0.8	
		Mean	Std. dev.	Mean	Std. dev.
D1:MPa	**0.5**	0.46	0.09	0.71	0.08
D1:MPa	**0.7**	0.48	0.08	0.72	0.07
D1:MPa	**0.9**	0.54	0.12	0.75	0.09
D2:Flow	**0.5**	0.59	0.18	0.77	0.07
D2:Flow	**0.7**	0.70	0.18	0.79	0.08
D2:Flow	**0.9**	0.88	0.09	0.87	0.07
D2:MPa	**0.5**	0.60	0.15	0.79	0.06
D2:MPa	**0.7**	0.67	0.13	0.82	0.05
D2:MPa	**0.9**	0.84	0.12	0.88	0.06
D2:Slump	**0.5**	0.62	0.18	0.79	0.08
D2:Slump	**0.7**	0.70	0.15	0.79	0.08
D2:Slump	**0.9**	0.89	0.09	0.89	0.06

Table 8
The median RMSE on the test set and two-sided Wilcoxon Rank Sum test against KP 0.8. The marks are for significance levels $\alpha = 0.001$ (3 bullets), 0.01 (2 bullets), and 0.05 (one bullet). *W*/*T*/*L* means number of Wins/Ties/Losses.

Dataset	Pct	*N*	KP 0.8	GSGP-LSH		KP 0.4		MRGP	
D1:MPa	**0.5**	**50**	6.94	12.02	•••	7.58	•••	6.95	
D1:MPa	**0.7**	**50**	6.70	11.89	•••	7.38	•••	6.42	○
D1:MPa	**0.9**	**50**	6.65	11.81	•••	7.34	•••	6.37	
D2:Flow	**0.5**	**45**	15.62	14.41	○	18.17	•	17.04	•
D2:Flow	**0.7**	**50**	13.46	13.15		16.10	•••	15.63	•
D2:Flow	**0.9**	**50**	12.55	11.90		14.31	•	13.90	••
D2:MPa	**0.5**	**50**	1.37	3.27	•••	3.82	•••	2.32	•••
D2:MPa	**0.7**	**50**	1.24	2.72	•••	3.55	•••	1.97	•••
D2:MPa	**0.9**	**50**	1.06	2.58	•••	3.14	•••	1.69	•••
D2:Slump	**0.5**	**49**	9.11	8.43		9.79		9.30	
D2:Slump	**0.7**	**50**	7.79	8.14		8.75	•••	8.69	•••
D2:Slump	**0.9**	**50**	7.42	6.92		8.79	•••	7.80	
W*/*T*/*L				1/5/6		0/1/12		1/4/7	

○, • Statistically significant improvement or degradation.

One can observe that the maximum correlation increased along with the percentage of the training data; we are still investigating this issue to understand when and why it happens. However, even in the test data, the collinearity is not considerably higher than *max_correlation*, except for $Pct = 0.9$. We conclude that KP can find sets of low-correlated features. Next, we evaluate if these sets provide high-quality models.

5.4.2. Comparison with EC methods

In this section, we compare KP with GSGP-LSH and MRGP. Table 8 presents a summary of the median RMSE on the test set and the two-sided Wilcoxon Rank Sum test, where the baseline is KP with $max\_correlation = 0.8$. *N* is the number of runs that worked. All failed runs were GSGP-LSH's, and we did run the other methods the same number of times for a fair comparison. A failed run means that the resulting *RMSE* was Inf, NaN, or NA.

As expected, for all methods, an increase in Pct means a decrease in RMSE because larger training data simplifies the learning process and allows for the generation of more accurate models. For D1:MPa, the biggest dataset, GSGP-LSH was unable to find good models within the 1 min timeframe. The best results were obtained by MRGP, which was better than KP 0.8 for $Pct = 0.7$ and not different for $Pct = 0.5$ and $Pct = 0.9$. The low-correlation feature sets found by KP 0.4 provided better prediction models than those of GSGP-LSH, but worse than those of KP 0.8. Thus, a less correlated feature set showed less predictive power than a more correlated set.

Considering D2:Flow, the best performing method was GSGP-LSH, but the models were only better than KP for Pct=0.5. KP was better than MRGP on all splits. Regarding D2:MPa, KP 0.8 ex-

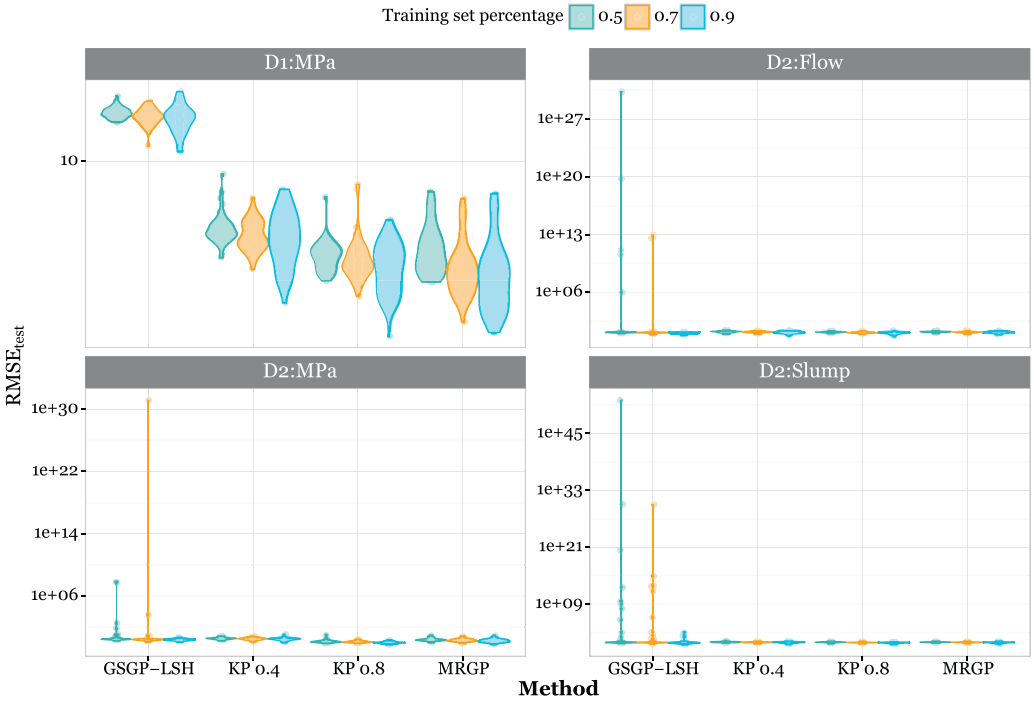

Fig. 2. Violin and dot plots of the datasets, methods, and configurations. This plot is in log-scale.

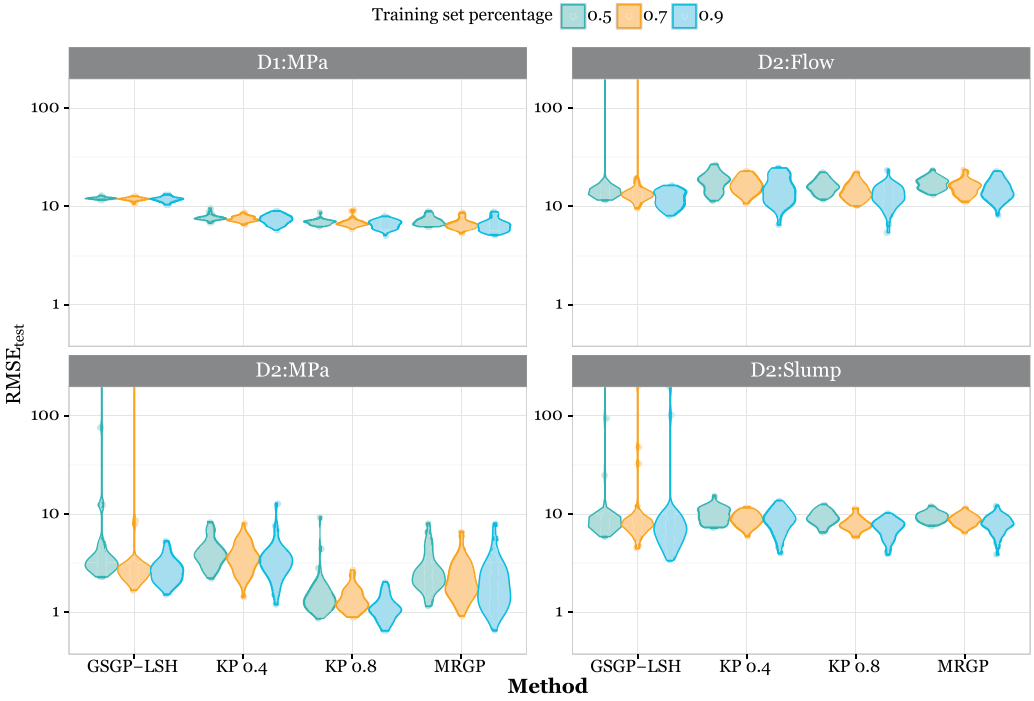

Fig. 3. Violin and dot plots of the datasets, methods, and configurations. The chart is zoomed between 0.5 and 150 for better visualization. This plot is in log-scale.

celled for all values of Pct, achieving considerably better results than those of GSGP-LSH and MRGP. Finally, for D2:Slump, GSGP-LSH was the best method for Pct=0.5, but the hypothesis test did not detect significant differences; except that KP achieved lower RMSE than MRGP. In the last row, the number of wins, ties, and losses shows that KP 0.8 was the best performing method and that KP 0.4 was worse than the former.

5.4.2.1. Visual comparison. Next, we present two charts, the first one (see Fig. 2) shows all solutions, while the second (see Fig. 3) is zoomed to a region of interest between 0.5 and 150. The dot plots are the real values for the runs; the dots are transparent and slightly jittered to show overlaps. The violin plot is a box plot with a rotated kernel density plot on each side which shows the probability density of the data at different values. Thus, the higher the concentration of dots (density), the larger the area in such region.

In Fig. 2 one can observe that GSGP-LSH had many outliers. Those outliers are, in fact, overfit runs. Such behavior was not detected in [36], probably because they tested other datasets. In that paper, GSGP-LSH overfit when LS was used in all generations; then, the authors tested a hybrid version in which LS is run for the first g=10 generations only and had no more issues. We are using the

Table 9
Average number of models and processing time.

Dataset	KP		GSGP-LSH		MRGP	
	Models	Time (s)	Models	Time (s)	Models	Time (s)
D1	1001	64.12	82,200	60.04	614.67	94.35
D2	1301	62.75	650,200	58.79	912.00	85.69

same approach here, as explained in Section 5.2, but it overfitted. We provide Fig. 3 for a better visualization.

GSGP-LSH was the most robust method on D1, showing the smallest variation, but got stuck in local optima. It was the best method on D2:Flow, and the second-best on D2:Slump. For KP, there is a visible improvement when *max_correlation* was increased from 0.4 to 0.8. This increase also resulted in more robustness and less variation. Thus, the chosen quality measure (AIC) and the safe prediction procedure were adequate at reducing overfitting. Finally, MRGP had no overfitting problems, suggesting that LASSO is more effective than OLS; thus, LASSO is a promising alternative method for a future investigation. However, next we show what the advantages are with OLS why we intend to stay with it for now.

5.4.2.2. Processing time. The number of models created by each method and the average processing time in seconds are shown in Table 9. For KP, it is the number of cycles because in each cycle there is one complete (reduced) model we disconsider the evaluation of the two expanded models used to evaluate the importance of the features. For GSGP-LSH and MRGP it is *Population size + Population size* $\times$ *Number of generations* because the offspring and population sizes are the same. The statistics for D2 considers all three datasets because they were executed for the same number of cycles/generations.

We configured the algorithms to run for *approximately* 60 s. As explained before, for KP and GSGP-LSH we estimated the total time in number of cycles/generations. MRGP has a time limit parameter; however, it did run for a longer period because it finishes only *after* a generation is completed. For this reason, the number of models evaluated by MRGP is averaged because the algorithm stopped after the time limit.

Even though we set the processing time as termination criterion, it is interesting to observe that GSGP-LSH is many times faster than KP and MRGP, and that KP (in Python) was substantially faster than MRGP (in Java). KP built approximately 38% (for D1) and 29% (for D2) more models than MRGP in approximately 66% of the time[7] LASSO gave MRGP a better predictive accuracy than OLS in KP, but it is slower than OLS because of the nature of the problem being optimized (and possibly because of some configuration issues).

We also ran GSGP-LSH allowing LS for all generations, instead of limiting it to 10 generations, to compare the processing time. The increase in time was not significant, approximately 7 s, probably because it is optimizing only three coefficients per generation, while KP optimized 51 for each of the two expanded models and 11 for the complete solution.

5.4.2.3. Interpretability. In Table 10, we present the mean and standard deviation of the number of nodes in the final solution created for each dataset. A node is either an operator or an operand. For KP, one must add the intercept and multiply each feature by a constant. For MRGP, one must add the intercept and each constant is multiplied by a node in the individual. GSGP-LSH does not provide the final solution for comparison. Besides, interpretability is a known issue in GSGP as it exponentially grows the expression tree without a maximum size limit (the geometric crossover adds two individuals).

[7] If we consider the expanded models created by KP to calculate the importance of the variables, then the number is two times bigger (3003 and 3903).

Table 10
Number of nodes in the complete solution.

Dataset		KP		MRGP	
	Pct	Mean	Std. dev.	Mean	Std. dev.
D1:MPa	**0.5**	123.92	18.48	397.88	58.30
D1:MPa	**0.7**	132.64	23.86	411.32	70.16
D1:MPa	**0.9**	117.84	22.86	404.84	65.74
D2:Flow	**0.5**	128.36	19.58	126.80	22.10
D2:Flow	**0.7**	131.16	24.12	195.68	14.66
D2:Flow	**0.9**	129.92	17.30	210.92	25.78
D2:MPa	**0.5**	113.36	20.42	131.12	15.12
D2:MPa	**0.7**	117.48	17.16	181.52	33.02
D2:MPa	**0.9**	109.08	20.80	211.88	30.91
D2:Slump	**0.5**	127.04	24.35	128.84	18.45
D2:Slump	**0.7**	132.56	24.56	193.40	13.66
D2:Slump	**0.9**	132.72	20.59	207.80	25.94

One may see in the table that, in average, MRGP needed more than three times the number of nodes needed by KP for D1 and more than 40% the number needed by KP for D2.

MRGP generates weights for each node in the expression tree, meaning that there are many constants to interpret. As an example, the most accurate solution found by MRGP for D1, $Pct = 0.5$, $run = 10$, is presented in Fig. 5; it is an average size solution. For KP, the solution for the same dataset and run is shown in Fig. 4. As can be seen, KP returns features without weights because they can be found by another method. Also, the new features can be investigated separately, and posterior feature selection methods can be employed. Nevertheless, KP may also be configured to return the complete linear model.

Figs. 4 and 5 are examples of average size solutions. As the algorithms are stochastic, the complexity of the solutions depends on the run. The important aspect is that, for KP, the resulting features are usually small and make the interpretation easier than having a single and long feature. Nevertheless, in order to achieve a higher-quality model, we understand that MRGP's approach can be considered.

5.4.3. Comparison with statistical and Machine Learning methods

We believe that it is important to compare KP with traditional statistical and ML methods in order to verify if our approach is useful. To this end, the results obtained by KP and other methods from Weka are shown in Table 11.

The most important points of this comparison are as follows. KP was substantially better than Gaussian Processes on D1 and D2:MPa, achieving similar or worse quality on the remaining datasets. KP should achieve better results than LR in all runs because it should be able to discover intrinsic useful relationships among variables; however, this was not the case as LR showed better performance on two cases and a similar performance in another four cases. Notwithstanding, one may also observe that LR was better than GSGP-LSH and MRGP for the same datasets (see Table 8). The conclusion is that one should start with LR and try another method whether the model is not satisfactory.

MLP was by far the best method on D2:MPa, but lost on all D1 cases and achieved similar, but higher, medians on the remaining datasets. Lastly, the two SVM methods showed poor performance, while SVM-RBF was the worst. Tuning the parameters of all these methods, including KP, could lead to even better results, although this is out of scope in the current paper.

In general, KP found models with prediction qualities that are as good as, or better than more complex and robust methods. Nonetheless, KP is solving a more complicated problem by search-

F_1 = safeDiv (mul (safeDiv (X6, add (X3, X5)), X6), X2)
F_2 = add (mul (sub (safeDiv (add (safeDiv (mul (safeDiv (X6, mul (X7, X2)), X6), X7), X5), X6), X1), X1), X5)
F_3 = sub (mul (add (mul (add (add (X2, X7), sub (X7, X6)), X1), X7), X7), X1)
F_4 = add (mul (sub (safeDiv (X4, safeDiv (mul (safeDiv (X6, mul (X7, X2)), X6), X7)), X5), X3), X1)
F_5 = sub (safeDiv (mul (add (X6, X7), X2), X6), X2)
F_6 = safeDiv (safeDiv (sub (mul (X7, X3), X6), X7), X7)
F_7 = safeDiv (mul (safeDiv (X6, add (X3, X5)), X6), X6)
F_8 = safeDiv (add (X1, X0), X7)
F_9 = add (safeDiv (safeDiv (safeDiv (X7, X3), X1), X7), X7)
F_{10} = sub (sub (sub (sub (X7, X0), X1), X6), X3)

Fig. 4. The solution found by KP 0.8 for D1, *Pct* = 0.5, *run* = 10. Each row is a feature, in descending order by their measured importance (*p*-value).

0.5642741735607776 1.489685080837832 -0.001940843632311741 -0.013311641727225898 -0.028815673997481823 0.0 0.0 0.05388766222933474 0.0 0.0 -9.208653209960547E-6 -0.18153324290955924 0.0 0.007568988520008509 0.0 -0.03042287071652561 -0.005958789646933299 0.06805478478034446 2.0823848395948817E-6 0.21877548191511753 2.277364869589371E-5 -4.209167848061266E-9 0.0 -3.959428189455679E-4 -1.484352922687251E-4 -3.450055503322242E-8 -0.5123686508009959 0.0 5.42888875553093E-5 -50.231630265302776 1.582023474755423E-6 -0.07222139172422774 0.0 0.0 0.0 0.24914842778389898 -0.20112813987794376 0.0311408581419013 0.0031230491045083557 0.0 0.0 0.0 0.0 -0.009212673076972171 0.008774997388744732 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0012468437722671071 0.0026958785658863366 0.0035518413566837247 3.271352563416159E-5 0.0 -0.00860244296876585 0.0 3.0371885408686233E-5 6.55445880593691E-4 -2.18672839734938 24E-5 0.0 0.005018793606184378 0.0 -3.5407567449223374E-5 -0.09424590204186901 0.0 -0.01538581197844892 0.01300065835259202 -3.337222500327086E-7 -0.04461388369039028 -1.9816945531931775E-7 1.8719048712613342E-6 -0.02360080792959842 -3.051631865783058E-6 0.0 0.0 -4.380093002009564E-8 -7.779884520165063E-8 0.0 -1.580920901783054E-10 0.0 0.1254359007765962 0.06808962376952692 -2.6057849593481802E-11 -1.0125069737335788E-5 0.0 0.16085732102212782 -1.8461270683617894E-7 0.11320953181411784 0.02772306174665268 0.0 0.0 -0.003178408385814222 0.0 0.0016597231056268253 0.0 0.026540513900150765 -0.012380578972349647 0.0010681960923677387 0.0028424461110986986 9.537365427215783E-7 -0.050614170280415816 -6.929412578504819E-6 -5.592138474917022E-7 0.0 -0.003665408433632131 1.6688182048245275E-8 -1.2227786480968513E-5 -5.6065208645300 75E-6 -1.1872296525812083E-4 0.0 -0.018618571852996084 5.051888894295086E-6 0.0 0.0 -1.427281179146826E-5 -4.7453943785223567E-7 0.06016617715959356 -9.202048324347654E-7 -0.18009539061511107 0.0 -9.085937359179258 -3.675651120936964E-7 -5.8198657686568485 1.6087736431755727E-6 -354.3547472142399 -387.69212504794064,-397.90236227814336,(- (mydivide (+ (- (* (+ X3 X1) (mydivide X2 X3)) (* (+ X6 X2) (mydivide X2 X2))) (* (mydivide (- X5 X6) (+ X7 X2)) (+ (+ X6 X3) (* X2 X8)))) (- (* (+ (* X6 X4) (mydivide X3 X3)) (* (- X5 X4) (mydivide X5 X3))) (mydivide (* (mydivide X4 X8) (+ X6 X5)) (mydivide (* X6 X3) (mydivide X3 X4))))) (* (mydivide (mydivide (- (- X5 X8) (+ X1 X1)) (- (* X6 X6) (+ X8 X8))) (mydivide (mydivide (- X6 X2) (mydivide X6 (mydivide X1 X8))) (mydivide (- X2 X8) (* X7 X6)))) (* (* (- (- X1 X5) (* X1 X5)) (mydivide (+ X7 X7) (mydivide X6 X2))) (* (* (- X8 X6) (- X8 X6)) (mydivide (mydivide X6 X4) (* X6 X7))))))

Fig. 5. The most accurate solution found by MRGP for D1, Pct=0.5, run=10.

Table 11
The median RMSE on the test set and two-sided Wilcoxon Rank Sum test agains KP 0.8. The marks are for significance levels $\alpha = 0.001$ (3 bullets), 0.01 (2 bullets), and 0.05 (one bullet). *W*/*T*/*L* means number of Wins/Ties/Losses. Some KP results are different from those in Table 8 because here $N = 50$.

Dataset	Pct	KP 0.8	GaussP		LR		MLP		SVM-Poly		SVM-RBF	
D1:MPa	**0.5**	6.94	8.39	•••	10.49	•••	7.53	•••	10.87	•••	11.30	•••
D1:MPa	**0.7**	6.70	8.06	•••	10.44	•••	7.73	•••	10.85	•••	11.01	•••
D1:MPa	**0.9**	6.65	7.76	•••	10.21	•••	7.49	•••	10.67	•••	10.58	•••
D2:Flow	**0.5**	15.65	13.81	○○○	13.72	○○○	17.25	••	14.55	○○	17.31	•
D2:Flow	**0.7**	13.46	13.35		13.51		15.79	•	14.35		17.08	•••
D2:Flow	**0.9**	12.55	12.96		12.84		14.37	•	13.86		16.44	•••
D2:MPa	**0.5**	1.37	3.84	•••	2.88	•••	1.27		2.88	•••	6.64	•••
D2:MPa	**0.7**	1.24	3.12	•••	2.91	•••	1.01	○○	2.86	•••	5.66	•••
D2:MPa	**0.9**	1.06	2.54	•••	2.56	•••	0.53	○○○	2.55	•••	4.91	•••
D2:Slump	**0.5**	9.05	7.56	○○○	7.99	○○○	8.89		8.63		9.42	
D2:Slump	**0.7**	7.79	7.51	○	7.89		8.54		8.46	•	9.42	••
D2:Slump	**0.9**	7.43	6.91		7.67	•	8.66	•	8.31	•	9.26	•••
W*/*T*/*L			3/3/6		2/3/7		2/3/7		1/3/8		0/1/11	

○, • Statistically significant improvement or degradation.

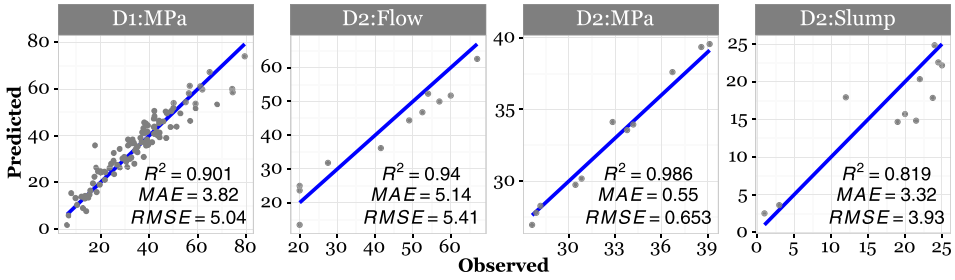

Fig. 6. Observed *versus* predicted plots for the best constructed features. The diagonal blue line is the reference. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 12
Comparison of Mean and Median RMSE (test set, rounded to two decimal places) with results from the literature for dataset D1. Symbol '–' means not available.

Method	Models	Split	Mean	Median
KP 0.8	1001	50%/50%	6.90	6.94
KP 0.8	1001	70%/30%	6.82	6.70
KP 0.8	1001	90%/10%	6.58	6.65
MLP [4]	–	10-fold cv	6.33	–
CART [4]	–	10-fold cv	9.70	–
CHAID [4]	–	10-fold cv	8.98	–
SVM [4]	–	10-fold cv	6.91	–
LR [4]	–	10-fold cv	11.24	–
Square regression [18]	–	70%/30%	–	15.91
Isotonic regression [18]	–	70%/30%	–	13.39
Radial basis function network [18]	–	70%/30%	–	16.09
SVM polynomial kernel (degree 5) [18]	–	70%/30%	–	6.79
KP + LR [27]	101	10-fold cv	6.54	6.51
KP + Gaussian processes [27]	101	10-fold cv	5.98	–
Standard GP [18]	360,000	70%/30%	–	8.67
Geometric semantic GP [18]	280,000	70%/30%	–	5.93*
GOT [42]	160,000	5-fold cv	7.12	–
WOS [42]	160,000	5-fold cv	6.89	–
GWPOT [42]	640,000	5-fold cv	6.38	–
GEGA [17]	200,000	66%/34%	9.95	–
GP [17]	200,000	66%/34%	10.87	–

ing for both the structure of the model (the formula) and the parameters, while the other methods have a fixed structure and must only optimize the parameters. Also, one must remember that KP is creating features with only the four basic arithmetic operators, while the other methods can build smoother surfaces (kernels, activation functions).

Regarding interpretability, the best models are those built by LR, which is the simplest method. However, the models created by KP are also linear in the parameters and likely easier to interpret than an MLP and a set of SVM weights. Moreover, SR methods are used to search for hidden knowledge" in the data and express it in formulae, which could be useful to researchers, even if it is less precise than a black-box method.

5.4.4. Comparison with the literature

Finally, in Table 12 we compare our results with those of the literature. The number of models of the EC methods was estimated as: $population\,size + population\,size \times generations \times crossover\,rate$. Unfortunately, we could not find reports in the literature using SR methods on dataset D 2.

The results of KP are better than several others reported in the literature, even though KP was configured with only the four basic arithmetic operators as suggested in [18], while other methods used more operators that are potentially more effective in approximating the data. KP outperformed the EC methods (from Standard GP to below) but required fewer models (there is no information on processing times in the literature). For GSGP (marked in the table), we tried to reproduce the results using the library provided by the authors, but the average RMSE was more than twice the one reported in their paper [18][8]

The other KP results in the table (KP+LR and KP+Gaussian Processes) are from [27], in which we used Genetic Programming to provide the experts and performed 10-fold cross-validation (cv) each generation to reduce overfitting. Given the high computational cost of using cv, in this paper we employed AIC expecting to achieve a similar predictive quality, but with a lower computational cost. As the results are not considerably different, although a direct comparison is 50 × 90%/10% vs 10-fold cv, we consider that the results achieved by KP in this work are satisfactory, suggesting that AIC was useful to reduce overfitting.

Related EC work from the literature does not report computational time for comparison, but some is implemented in MATLAB, which is not a high-performance programming language. Hence, we are unable to compare such processing cost.

5.5. Constructed features

Table 13 has the constructed features that presented the lowest test RMSE for KP 0.8 in each dataset with a training set percentage of 90%. Fig. 6 shows the observed versus predicted plots for those features.

As the purpose of KP is to create and improve features, they have no weights as can be seen in Table 13. These weights must be optimized by OLS (or another method) for a specific training set. However, the final features must be adequate to allow the creation of high-quality models that present low RMSE on the test set. In the Appendix, we provide an *R* source code to evaluate those features.

Regarding the plots, the closer the points are to the reference line, the better the prediction. It is easy to notice that the highest quality model was D2:MPa, and the lowest quality model was D2:Slump. D1:MPa has an outlier near to position 20 × 40 and a few poor predictions around position 70 × 60. Nevertheless, the models present an acceptable overall quality, with high R^2 values.

As explained before, a complete model using a set of ten features could be a black-box. However, it is plausible to assume that most features are simple enough to be understood and can likely be interpreted by material engineers. Thus, one might say that the models are gray-boxes. This is one of the reasons for using SR methods [15,19]. We are not aware of an algorithm that can assure full interpretability of models, and we do not guarantee it either.

6. Summary, conclusions, and future works

Kaizen Programming is a hybrid algorithm that uses a collaborative problem solving approach where partial solutions compose

[8] In fact, the reported value seems to be the MAE, not the RMSE, and our MAE was better than that as shown in the Appendix Table 14.

Table 13
Best constructed features. Function *div* is *safeDiv*.

Dataset	Features
D1:MPa	1: mul(sub(div(sub(mul(mul(add(X3, add(X3, mul(X3, X5))), X6), X6), X7), X2), X7), X6) 2: sub(add(add(sub(div(mul(mul(mul(X1, X0), X4), X5), X4), X7), X0), X3), X1) 3: mul(X3, add(sub(add(X4, X4), X3), X2)) 4: sub(mul(add(sub(mul(X7, mul(X7, X6)), X0), X6), X5), X2) 5: mul(mul(sub(add(add(mul(X3, X4), X7), X5), X6), X7), X0) 6: sub(mul(X5, add(X7, X2)), add(X4, div(X2, X3))) 7: div(mul(add(add(mul(mul(X0, X3), X4), X0), X1), X7), X0) 8: div(sub(X0, div(add(add(div(X1, X0), X2), mul(X3, add(X3, X4))), X0)), X3) 9: sub(mul(mul(sub(mul(X3, mul(X1, X6)), X0), X6), X5), X2) 10: mul(div(sub(add(sub(add(X1, X2), sub(X7, X3)), X1), X4), X7), X0)
D2:Flow	1: mul(mul(sub(X6, X0), sub(X3, mul(sub(X0, X3), add(X5, X4)))), add(X1, X4)) 2: div(mul(mul(X2, sub(X6, add(div(X6, X4), X5))), X4), add(X6, X4)) 3: div(mul(mul(X2, sub(X6, X2)), div(X6, X4)), X5) 4: mul(mul(sub(X6, div(sub(X3, mul(sub(X0, X3), add(X5, X4))), add(X1, X4))), X3), X2) 5: div(add(div(div(add(div(sub(X1, X4), X3), X4), X0), X3), X1), X1) 6: div(div(add(add(add(X3, div(X4, X5)), X1), X6), X5), X3) 7: div(mul(mul(X2, sub(X4, X0)), div(X6, X4)), X5) 8: sub(sub(mul(mul(div(sub(mul(mul(X4, div(X4, X2)), X6), X1), X5), X1), X0), X4), X3) 9: mul(mul(X4, X2), X1) 10: add(mul(div(sub(X3, X2), X6), X2), X5)
D2:MPa	1: mul(sub(add(sub(sub(div(mul(sub(X3, X3), X1), add(X3, X6)), X3), X3), X2), X5), X3) 2: sub(add(X2, X3), X4) 3: sub(add(sub(sub(div(mul(sub(X3, X3), add(add(X3, X6), X3)), X3), X0), X5), X3), X2) 4: mul(X6, X6) 5: sub(mul(div(sub(sub(div(add(sub(X5, X2), X6), X3), X4), X6), X3), X4), X3) 6: mul(X6, X0) 7: div(X6, sub(X4, X3)) 8: sub(X4, add(sub(X4, X4), X5)) 9: mul(sub(add(sub(sub(div(mul(sub(X3, X3), X1), add(X3, X6)), X3), X3), X0), X5), X3) 10: mul(X6, X3)
D2:Slump	1: mul(add(mul(mul(mul(mul(X6, X5), div(X1, X3)), X2), X0), X0), X1) 2: add(sub(X0, X1), X6) 3: mul(add(X6, X2), X5) 4: sub(mul(mul(div(add(mul(sub(div(X1, div(X0, X1)), X4), X1), X5), X4), X2), X2), X3) 5: mul(add(add(mul(X1, mul(X6, X5)), X0), X1), X3) 6: mul(mul(X5, X4), X5) 7: div(X1, div(sub(X0, add(X2, X2)), X6)) 8: div(X1, sub(mul(add(X2, X2), X6), X3)) 9: div(add(sub(X4, div(X4, X5)), X5), X5) 10: div(mul(div(X5, X3), X6), X5)

a complete solution. KP can use global optimization algorithms, statistics, and ML. The partial solutions are created by the experts that generate ideas (partial solutions) based on the current standard (best set of partial solutions), which is improved over the cycles. Ideas are evaluated by their contribution to help to solve the problem, and the most important ones are selected for the next improvement cycle. Ideas providing similar partial solutions (not similar contributions) are automatically dropped to maintain diversity.

In this work, KP was employed to perform automatic feature engineering to build regression models for predicting properties of concrete. KP was tested on two datasets from the literature and the results were compared to those of EC, statistics, and ML methods. It was shown that KP is competitive in terms of predictive quality (RMSE), outperforming several methods from the literature. KP can be a useful alternative to related methods as it provides a set of features instead of a single one.

KP, being implemented in Python, was faster than MRGP (in Java), and both were substantially slower than GSGP-LSH (in C++). However, GSGP-LSH runs LS only in the first ten generations to optimize only three coefficients to reduce the chance of overfitting, while KP and MRGP optimize many coefficients every iteration.

In the future, we intend to apply KP as a feature constructing approach on harder problems, for instance, larger datasets containing more features, noise, and missing values. If OLS does not show acceptable performance, regularization methods such as LASSO might be useful, despite its higher computational cost. We also plan to compare our approach to other methods, such as FFX.

Further aspects: (i) ifferent initialization methods may provide better seeds for the search. This is another important issue to be investigated in the future, as has been already demonstrated in GP. (ii) Instead of Simulated Annealing, one may investigate other combinatorial optimization methods. Also, the inclusion of ERCs should employ efficient numerical optimization methods, not SA. (iii) We intend to study different approaches to reduce overfitting. While we used 10-fold cross-validation during training in previous work, it was time-consuming, though it showed better robustness against overfitting than achieved here with AIC.

The results obtained in this work show that KP can find high-quality features in short time and that a traditional linear regression model can achieve similar or better results than black-box models. The future research avenues presented above are opportunities to enhance KP to provide a useful tool to the ML community.

Acknowledgments

This paper was supported by the Brazilian Government CNPq (Universal) grant (486950/2013-1) and CAPES (Science without Borders) grant (12180-13-0) to V.V.M., and Canada's NSERC Discovery grant RGPIN 283304-2012 to W.B.

7. Appendix

This section presents the *R* source code to test the best features shown in Table 13, Section 5.5, and tables with more detailed statistics regarding the experiments.

7.1. R source code

R code to validate the solutions presented in the constructed features section.

```
# Clean the enviroment
rm (list=ls())

# Define safe division
safediv = function(a,b) if (abs(b)<1e-10) 0 else a/b
safeDiv = Vectorize(safediv)

mul = function(a,b) a*b
add = function(a,b) a+b
sub = function(a,b) a-b

# Read the dataset
# Change the file path, the header, and the field separator character if
    necessary
tab = read.table("/home/user/Research/Data/concrete_data.csv", header = TRUE,
     sep=",")

# Rename the columns to X0 to Xn
colnames(tab) = paste("X", 0:(ncol(tab)-1), sep="")

# Add the original columns to the
# enviroment so they can be called directly
attach(tab)

# Create the features
df = data.frame(
  f1=mul (sub (safeDiv (sub (mul (mul (add (X3, add (X3, mul (X3, X5))), X6),
      X6), X7), X2), X7), X6),
  f2=sub (add (add (sub (safeDiv (mul (mul (mul (X1, X0), X4), X5), X4), X7),
      X0), X3), X1),
  f3=mul (X3, add (sub (add (X4, X4), X3), X2)),
  f4=sub (mul (add (sub (mul (X7, mul (X7, X6)), X0), X6), X5), X2),
  f5=mul (mul (sub (add (add (mul (X3, X4), X7), X5), X6), X7), X0),
  f6=sub (mul (X5, add (X7, X2)), add (X4, safeDiv (X2, X3))),
  f7=safeDiv (mul (add (add (mul (mul (X0, X3), X4), X0), X1), X7), X0),
  f8=safeDiv (sub (X0, safeDiv (add (add (safeDiv (X1, X0), X2), mul (X3, add
      (X3, X4))), X0)), X3),
  f9=sub (mul (mul (sub (mul (X3, mul (X1, X6)), X0), X6), X5), X2),
  f10=mul (safeDiv (sub (add (sub (add (X1, X2), sub (X7, X3)), X1), X4), X7)
     , X0),
  resp=tab[, ncol(tab)]  # Original response variable
)


# Print the correlation summary
cat("\n\nPrint the correlation summary on df:")
correlations = cor(df)
correlations[abs(correlations)>0.999999]=0.0  # Remove self-correlation
print(summary(as.numeric(correlations)))


# Define the split point
pct = 0.9
qtd = trunc(nrow(df)*pct)
```

```

# Create empty arrays to save RMSE
vet_RMSE_train = NULL
vet_RMSE_test = NULL

# For posterior plotting and analysis
best_train = NULL
best_test = NULL
best_model = NULL
best_model_RMSE = Inf

# Generate many splits and calculate RMSE
for (i in 1:50){
  # Shuffle the dataset
  pos = sample(nrow(df), replace=F)

  # Split the dataset
  train = df[ pos[1:qtd], ]
  test = df[ pos[-(1:qtd)], ]

  # Build a model on the training set
  model = lm(resp ~ ., data=train)
  # print (summary(model))

  # Calculate and save RMSE of the test set
  pred = predict(model, newdata=test)
  RMSE = sqrt(mean((test$resp - pred)^2))
  vet_RMSE_test = c(vet_RMSE_test, RMSE)

  # Calculate and save RMSE of the training set
  RMSE = sqrt(mean((train$resp - predict(model))^2))
  vet_RMSE_train = c(vet_RMSE_train, RMSE)


  if (RMSE < best_model_RMSE){
    best_train = train
    best_test = test
    best_model = model
    best_model_RMSE = RMSE
  }
}

################################
# USING THE BEST MODEL
################################

# Print descriptive statistics
cat("\n\nPrint descriptive statistics (train RMSE):\n")
print (summary(vet_RMSE_train))


cat("\n\nPrint descriptive statistics (test RMSE).")
cat("\nThe median value should be close to 6.647 (see Table 10, D1:MPa, Pct
    =0.9):\n")
print (summary(vet_RMSE_test))


# Print summary of the best model (the one with the lowest training RMSE)
cat("\n\nPrint summary of the best model  (the one with the lowest training
    RMSE):\n")
print (summary(best_model))

# Plot expected vs predicted
reference_line_train = sort(best_train$resp)
pred_train = predict(best_model)
RMSE_train = best_model_RMSE
mae_train = mean(abs(best_train$resp - pred_train))

reference_line_test = sort(best_test$resp)
pred_test = predict(best_model, newdata=best_test)
RMSE_test = sqrt(mean((best_test$resp - pred_test)^2))
mae_test = mean(abs(best_test$resp - pred_test))

par(las=1, mfrow=c(2,1))
plot(best_train$resp, pred_train, ylim=range(c(best_train$resp, pred_train)),
     xlab="Observed", ylab="Predicted", main=paste("RMSE_train =", round(
    RMSE_train, 4), " MAE_train=", round(mae_train, 4)), col="darkgray", pch
    =20, cex=1.3)
points(best_train$resp, pred_train, col="darkblue", cex=1.2, lwd=2)
lines(reference_line_train, reference_line_train, col="darkred", lwd=3, lty
    =2)
grid()


plot(best_test$resp, pred_test, ylim=range(c(best_test$resp, pred_test)),
    xlab="Observed", ylab="Predicted", main=paste("RMSE_test =", round(RMSE_
    test, 4), " MAE_test=", round(mae_test, 4)), col="darkgray", pch=20, cex
    =1.3)
points(best_test$resp, pred_test, col="darkblue", cex=1.2, lwd=2)
lines(reference_line_test, reference_line_test, col="darkred", lwd=3, lty=2)
grid()
```

7.2. More detailed descriptive statistical analysis

In Table 14, we present other statistics regarding KP 0.8 prediction quality, which was better than KP 0.4. Table 15 shows an extended descriptive analysis comparing KP, GSGP-LSH, and MRGP, and Tables 16 and 17 shows an extended descriptive analysis compared KP with methods in WEKA.

Table 14
Other predictive quality statistics for KP 0.8.

	Dataset	D1:Mpa			D2:Flow			D2:MPa			D2:Slump		
	Pct	**0.5**	**0.7**	**0.9**	**0.5**	**0.7**	**0.9**	**0.5**	**0.7**	**0.9**	**0.5**	**0.7**	**0.9**
	Median	5.201	5.079	4.957	12.359	1.028	6.640	10.744	0.978	5.863	10.002	0.863	5.585
MAE	**Mean**	5.184	5.085	5.000	12.438	1.133	6.824	10.919	1.047	6.047	10.066	0.884	5.653
	Std. dev.	0.241	0.289	0.469	1.868	0.444	1.056	1.740	0.253	0.882	2.412	0.207	1.119
	Median	48.184	44.933	44.189	244.798	1.890	81.834	181.225	1.542	60.642	157.464	1.124	55.154
MSE	**Mean**	47.794	46.855	43.783	264.940	4.186	87.115	212.997	2.077	65.007	174.674	1.343	54.602
	Std. dev.	5.906	8.873	8.571	86.591	11.979	27.210	87.270	1.355	21.160	88.418	0.816	21.078
	Median	6.941	6.703	6.647	15.646	1.375	9.046	13.461	1.242	7.787	12.548	1.060	7.427
RMSE	**Mean**	6.901	6.820	6.585	16.074	1.643	9.223	14.324	1.385	7.968	12.820	1.115	7.239
	Std. dev.	0.409	0.596	0.652	2.587	1.232	1.450	2.823	0.401	1.245	3.245	0.319	1.496
	Median	0.912	0.915	0.923	0.604	0.985	0.477	0.697	0.989	0.516	0.686	0.990	0.634
R	**Mean**	0.912	0.913	0.921	0.589	0.973	0.455	0.652	0.985	0.515	0.669	0.984	0.599
	Std. dev.	0.012	0.016	0.017	0.126	0.059	0.167	0.157	0.012	0.164	0.188	0.020	0.245
	Median	0.832	0.837	0.851	0.365	0.971	0.228	0.486	0.978	0.267	0.470	0.980	0.403
R^2	**Mean**	0.832	0.833	0.849	0.363	0.951	0.234	0.449	0.970	0.291	0.482	0.968	0.418
	Std. dev.	0.022	0.028	0.031	0.134	0.093	0.141	0.174	0.024	0.149	0.229	0.039	0.241

Table 15
Comparison of RMSE on the test set with GSGP-LSH, MRGP, and KP with two *max_correlation* values. Pct is the training set percentage. *N* is the number of successful runs. The results of a two-sided Wilcoxon Rank Sum test with levels $\alpha = 0.001$, 0.01 and 0.05 are also indicated.

Method	Dataset	Pct.	*N*	Min.	1st Qu.	Median		Mean	3rd Qu.	Max.	Std. dev.
GSGP-LSH	**D1:MPa**	**0.5**	**50**	11.63	11.92	12.02	•••	12.08	12.24	12.87	0.27
KP 0.4	**D1:MPa**	**0.5**	**50**	6.86	7.41	7.58	•••	7.68	7.83	9.52	0.46
KP 0.8	**D1:MPa**	**0.5**	**50**	6.25	6.59	6.94		6.90	7.13	8.68	0.41
MRGP	**D1:MPa**	**0.5**	**50**	6.22	6.59	6.95		7.10	7.44	8.87	0.71
GSGP-LSH	**D1:MPa**	**0.7**	**50**	10.62	11.66	11.89	•••	11.89	12.15	12.62	0.39
KP 0.4	**D1:MPa**	**0.7**	**50**	6.56	7.18	7.38	•••	7.48	7.83	8.65	0.44
KP 0.8	**D1:MPa**	**0.7**	**50**	5.89	6.49	6.70		6.82	7.04	9.13	0.60
MRGP	**D1:MPa**	**0.7**	**50**	5.33	6.10	6.42	○	6.60	6.84	8.63	0.73
GSGP-LSH	**D1:MPa**	**0.9**	**50**	10.38	11.34	11.81	•••	11.79	12.13	13.12	0.64
KP 0.4	**D1:MPa**	**0.9**	**50**	5.74	6.80	7.34	•••	7.39	7.95	8.95	0.79
KP 0.8	**D1:MPa**	**0.9**	**50**	5.04	6.07	6.65		6.59	7.07	7.95	0.65
MRGP	**D1:MPa**	**0.9**	**50**	5.11	5.64	6.37		6.42	6.83	8.80	0.92
GSGP-LSH	**D2:Flow**	**0.5**	**45**	11.61	13.13	14.41	○	4.47E+28	16.04	2.01E+30	3.00E+29
KP 0.4	**D2:Flow**	**0.5**	**45**	11.42	14.84	18.17	•	17.87	19.95	25.64	3.69
KP 0.8	**D2:Flow**	**0.5**	**45**	11.78	13.50	15.62		15.87	17.18	22.06	2.56
MRGP	**D2:Flow**	**0.5**	**45**	13.16	15.29	17.04	•	17.12	18.61	23.19	2.33
GSGP-LSH	**D2:Flow**	**0.7**	**50**	9.54	12.31	13.15		3.67E+11	14.50	8.62E+12	1.55E+12
KP 0.4	**D2:Flow**	**0.7**	**50**	10.71	14.05	16.10	•••	16.52	18.59	22.77	3.12
KP 0.8	**D2:Flow**	**0.7**	**50**	10.19	12.36	13.46		14.32	16.22	22.01	2.82
MRGP	**D2:Flow**	**0.7**	**50**	11.26	13.64	15.63	•	15.54	16.93	23.47	2.67
GSGP-LSH	**D2:Flow**	**0.9**	**50**	8.03	10.18	11.90		12.04	14.11	16.37	2.27
KP 0.4	**D2:Flow**	**0.9**	**50**	6.52	11.22	14.31	•	15.08	17.77	24.68	4.60
KP 0.8	**D2:Flow**	**0.9**	**50**	5.41	11.00	12.55		12.82	14.68	23.21	3.25
MRGP	**D2:Flow**	**0.9**	**50**	8.14	12.30	13.90	••	14.82	16.90	22.67	3.41
GSGP-LSH	**D2:MPa**	**0.5**	**50**	2.30	2.84	3.27	•••	2.28E+06	4.02	5.89E+07	1.13E+07
KP 0.4	**D2:MPa**	**0.5**	**50**	2.22	3.18	3.82	•••	4.13	4.64	8.21	1.38
KP 0.8	**D2:MPa**	**0.5**	**50**	0.87	1.13	1.38		1.64	1.73	9.22	1.23
MRGP	**D2:MPa**	**0.5**	**50**	1.16	1.92	2.32	•••	2.74	2.90	8.00	1.38
GSGP-LSH	**D2:MPa**	**0.7**	**50**	1.68	2.23	2.72	•••	2.73E+29	2.94	1.37E+31	1.93E+30
KP 0.4	**D2:MPa**	**0.7**	**50**	1.44	2.84	3.55	•••	3.71	4.41	7.99	1.29
KP 0.8	**D2:MPa**	**0.7**	**50**	0.89	1.12	1.24		1.39	1.63	2.70	0.40
MRGP	**D2:MPa**	**0.7**	**50**	0.93	1.60	1.97	•••	2.30	2.70	6.56	1.13
GSGP-LSH	**D2:MPa**	**0.9**	**50**	1.52	2.14	2.58	•••	2.73	3.11	5.29	0.81
KP 0.4	**D2:MPa**	**0.9**	**50**	1.21	2.50	3.14	•••	3.49	3.91	12.73	1.81
KP 0.8	**D2:MPa**	**0.9**	**50**	0.65	0.88	1.06		1.12	1.26	2.01	0.32
MRGP	**D2:MPa**	**0.9**	**50**	0.66	1.23	1.69	•••	2.09	2.58	7.96	1.30
GSGP-LSH	**D2:Slump**	**0.5**	**49**	5.91	7.73	8.43		2.07E+50	9.89	1.02E+52	1.45E+51
KP 0.4	**D2:Slump**	**0.5**	**49**	7.28	8.35	9.79		9.86	11.28	15.16	1.78
KP 0.8	**D2:Slump**	**0.5**	**49**	6.54	8.21	9.11		9.25	10.20	12.36	1.46
MRGP	**D2:Slump**	**0.5**	**49**	7.59	8.51	9.30		9.37	10.18	11.94	1.10
GSGP-LSH	**D2:Slump**	**0.7**	**50**	4.54	7.36	8.14		1.97E+28	9.36	9.85E+29	1.39E+29
KP 0.4	**D2:Slump**	**0.7**	**50**	5.96	7.83	8.75	•••	8.88	9.92	11.53	1.39
KP 0.8	**D2:Slump**	**0.7**	**50**	5.89	7.15	7.79		7.97	8.61	11.34	1.25
MRGP	**D2:Slump**	**0.7**	**50**	6.52	7.82	8.69	•••	8.72	9.47	11.50	1.11
GSGP-LSH	**D2:Slump**	**0.9**	**50**	3.41	5.39	6.92		30.33	8.87	891.60	127.94
KP 0.4	**D2:Slump**	**0.9**	**50**	4.01	7.08	8.79	•••	8.69	10.49	13.56	2.25
KP 0.8	**D2:Slump**	**0.9**	**50**	3.93	6.30	7.43		7.24	8.27	10.17	1.50
MRGP	**D2:Slump**	**0.9**	**50**	3.90	7.03	8.00		7.91	8.91	12.07	1.63

○, • Statistically significant improvement or degradation

Table 16
Part I: comparison of RMSE on the test set using Statistical and ML methods. All runs succeeded. The results of a two-sided Wilcoxon Rank Sum test with levels $\alpha = 0.001$, 0.01 and 0.05 are also indicated.

Method	Dataset	Pct.	Min.	1st Qu.	Median		Mean	3rd Qu.	Max.	Std. dev.
GaussP	**D1:MPa**	**0.5**	8.00	8.22	8.39	•••	8.39	8.55	8.87	0.23
KP 0.8	**D1:MPa**	**0.5**	6.25	6.59	6.94		6.90	7.13	8.68	0.41
LR	**D1:MPa**	**0.5**	10.04	10.32	10.49	•••	10.52	10.67	11.20	0.26
MLP	**D1:MPa**	**0.5**	6.50	6.99	7.53	•••	7.89	8.58	12.89	1.21
SVM-Poly	**D1:MPa**	**0.5**	9.95	10.68	10.87	•••	10.98	11.21	12.97	0.57
SVM-RBF	**D1:MPa**	**0.5**	10.69	11.20	11.30	•••	11.32	11.43	12.03	0.25
GaussP	**D1:MPa**	**0.7**	7.28	7.91	8.06	•••	8.07	8.24	8.66	0.29
KP 0.8	**D1:MPa**	**0.7**	5.89	6.49	6.70		6.82	7.04	9.13	0.60
LR	**D1:MPa**	**0.7**	9.76	10.27	10.44	•••	10.48	10.69	11.33	0.35
MLP	**D1:MPa**	**0.7**	6.53	7.23	7.73	•••	7.82	8.31	11.21	0.87
SVM-Poly	**D1:MPa**	**0.7**	9.55	10.48	10.85	•••	11.02	11.49	13.52	0.77
SVM-RBF	**D1:MPa**	**0.7**	10.28	10.77	11.01	•••	10.99	11.21	11.60	0.31
GaussP	**D1:MPa**	**0.9**	6.56	7.19	7.76	•••	7.72	8.25	9.17	0.66
KP 0.8	**D1:MPa**	**0.9**	5.04	6.07	6.65		6.59	7.07	7.95	0.65
LR	**D1:MPa**	**0.9**	9.19	9.81	10.21	•••	10.32	10.77	12.32	0.61
MLP	**D1:MPa**	**0.9**	5.78	7.01	7.49	•••	7.91	8.32	12.43	1.57
SVM-Poly	**D1:MPa**	**0.9**	9.22	10.39	10.67	•••	10.78	11.21	13.50	0.97
SVM-RBF	**D1:MPa**	**0.9**	9.68	10.25	10.58	•••	10.72	11.16	11.95	0.57
GaussP	**D2:Flow**	**0.5**	11.38	13.27	13.81	○○○	13.83	14.45	16.33	1.08
KP 0.8	**D2:Flow**	**0.5**	11.78	14.14	15.65		16.07	17.56	22.06	2.59
LR	**D2:Flow**	**0.5**	11.45	12.93	13.72	○○○	13.71	14.57	15.81	1.07
MLP	**D2:Flow**	**0.5**	12.74	15.77	17.25	••	18.17	19.98	26.75	3.60
SVM-Poly	**D2:Flow**	**0.5**	11.96	13.66	14.55	○○	14.80	15.55	19.53	1.67
SVM-RBF	**D2:Flow**	**0.5**	13.34	16.00	17.31	•	17.12	18.15	20.88	1.72
GaussP	**D2:Flow**	**0.7**	9.53	12.73	13.35		13.26	14.31	17.41	1.56
KP 0.8	**D2:Flow**	**0.7**	10.19	12.36	13.46		14.32	16.22	22.01	2.82
LR	**D2:Flow**	**0.7**	10.09	12.63	13.51		13.47	14.47	17.44	1.47
MLP	**D2:Flow**	**0.7**	10.59	13.52	15.79	•	15.90	17.81	22.07	3.23
SVM-Poly	**D2:Flow**	**0.7**	11.00	13.19	14.35		14.45	15.43	18.96	1.80
SVM-RBF	**D2:Flow**	**0.7**	11.77	15.26	17.08	•••	16.82	18.45	22.53	2.39
GaussP	**D2:Flow**	**0.9**	6.45	11.13	12.96		12.90	14.75	19.92	2.64
KP 0.8	**D2:Flow**	**0.9**	5.41	11.00	12.55		12.82	14.68	23.21	3.25
LR	**D2:Flow**	**0.9**	8.32	11.29	12.84		13.02	14.85	19.95	2.52
MLP	**D2:Flow**	**0.9**	4.84	11.76	14.37	•	14.97	18.09	35.69	5.15
SVM-Poly	**D2:Flow**	**0.9**	9.45	11.95	13.86		14.03	15.70	19.92	2.69
SVM-RBF	**D2:Flow**	**0.9**	7.57	14.00	16.44	•••	16.17	18.46	24.62	3.70
GaussP	**D2:MPa**	**0.5**	2.79	3.52	3.84	•••	3.83	4.08	5.18	0.60
KP 0.8	**D2:MPa**	**0.5**	0.87	1.13	1.38		1.64	1.73	9.22	1.23
LR	**D2:MPa**	**0.5**	2.32	2.76	2.89	•••	2.91	3.05	3.53	0.27
MLP	**D2:MPa**	**0.5**	0.57	0.91	1.27		1.44	1.88	2.73	0.65
SVM-Poly	**D2:MPa**	**0.5**	2.22	2.73	2.88	•••	2.88	3.04	3.60	0.28
SVM-RBF	**D2:MPa**	**0.5**	4.84	6.22	6.64	•••	6.57	6.97	7.75	0.61
GaussP	**D2:MPa**	**0.7**	2.20	2.85	3.13	•••	3.20	3.57	4.52	0.55
KP 0.8	**D2:MPa**	**0.7**	0.89	1.12	1.24		1.39	1.63	2.70	0.40
LR	**D2:MPa**	**0.7**	1.77	2.67	2.91	•••	2.85	3.11	3.61	0.40
MLP	**D2:MPa**	**0.7**	0.48	0.74	1.02	○○	1.25	1.50	3.82	0.74
SVM-Poly	**D2:MPa**	**0.7**	1.79	2.63	2.86	•••	2.80	2.99	3.62	0.39
SVM-RBF	**D2:MPa**	**0.7**	3.95	5.13	5.66	•••	5.76	6.54	7.60	0.88
GaussP	**D2:MPa**	**0.9**	1.16	2.04	2.55	•••	2.69	3.10	4.92	0.87
KP 0.8	**D2:MPa**	**0.9**	0.65	0.88	1.06		1.12	1.26	2.01	0.32
LR	**D2:MPa**	**0.9**	1.23	2.16	2.56	•••	2.69	3.11	4.97	0.76
MLP	**D2:MPa**	**0.9**	0.27	0.43	0.53	○○○	0.57	0.63	1.10	0.20
SVM-Poly	**D2:MPa**	**0.9**	1.10	2.07	2.55	•••	2.65	3.21	5.16	0.82
SVM-RBF	**D2:MPa**	**0.9**	2.26	3.95	4.91	•••	5.01	5.85	7.98	1.37

○, • Statistically significant improvement or degradation

Table 17
Part II: comparison of RMSE on the test set (rounded to two decimal places) using Statistical and ML methods. All runs succeeded. The results of a two-sided Wilcoxon Rank Sum test with levels $\alpha = 0.001$, 0.01 and 0.05 are also indicated.

Method	Dataset	Pct.	Min.	1st Qu.	Median		Mean	3rd Qu.	Max.	Std. dev.
GaussP	**D2:Slump**	**0.5**	6.05	7.11	7.56	○○○	7.55	8.03	9.08	0.61
KP 0.8	**D2:Slump**	**0.5**	6.54	8.20	9.05		9.22	10.19	12.36	1.45
LR	**D2:Slump**	**0.5**	6.96	7.80	7.99	○○○	8.04	8.32	9.26	0.45
MLP	**D2:Slump**	**0.5**	6.96	7.86	8.89		9.28	10.17	13.92	1.75
SVM-Poly	**D2:Slump**	**0.5**	6.75	8.20	8.63		8.83	9.24	12.13	1.18
SVM-RBF	**D2:Slump**	**0.5**	6.88	8.55	9.42		9.30	10.15	11.15	1.05
GaussP	**D2:Slump**	**0.7**	5.13	6.75	7.51	○	7.31	7.85	9.17	0.96
KP 0.8	**D2:Slump**	**0.7**	5.89	7.15	7.79		7.97	8.61	11.34	1.25
LR	**D2:Slump**	**0.7**	6.16	7.45	7.89		7.96	8.57	9.55	0.79
MLP	**D2:Slump**	**0.7**	4.51	7.20	8.54		8.74	10.06	13.99	2.23
SVM-Poly	**D2:Slump**	**0.7**	5.60	7.70	8.46	•	8.46	9.22	12.15	1.30
SVM-RBF	**D2:Slump**	**0.7**	4.94	8.24	9.42	•••	9.25	10.43	12.10	1.65
GaussP	**D2:Slump**	**0.9**	4.16	6.23	6.91		7.18	8.48	10.46	1.54
KP 0.8	**D2:Slump**	**0.9**	3.93	6.30	7.43		7.24	8.27	10.17	1.50
LR	**D2:Slump**	**0.9**	5.24	7.03	7.67	•	7.94	8.99	10.85	1.32
MLP	**D2:Slump**	**0.9**	2.82	6.52	8.66	•	8.23	10.01	14.75	2.77
SVM-Poly	**D2:Slump**	**0.9**	4.23	6.79	8.31	•	8.17	9.60	12.34	1.83
SVM-RBF	**D2:Slump**	**0.9**	2.84	7.24	9.26	•••	9.01	11.08	14.14	2.58

○, • Statistically significant improvement or degradation.

References

[1] I.-C. Yeh, Modeling of strength of high-performance concrete using artificial neural networks, Cem. Concr. Res. 28 (12) (1998) 1797–1808.
[2] J.-S. Chou, C.-F. Tsai, Concrete compressive strength analysis using a combined classification and regression technique, Autom. Constr. 24 (2012) 52–60.
[3] A. Baykasoğlu, A. Öztaş, E. Özbay, Prediction and multi-objective optimization of high-strength concrete parameters via soft computing approaches, Expert Syst. Appl. 36 (3) (2009) 6145–6155.
[4] J.-S. Chou, A.-D. Pham, Enhanced artificial intelligence for ensemble approach to predicting high performance concrete compressive strength, Constr. Build. Mater. 49 (2013) 554–563.
[5] M. Moini, A. Lakizadeh, M. Mohaqeqi, Effect of mixture temperature on slump flow prediction of conventional concretes using artificial neural networks, Aust. J. Civil Eng. 10 (1) (2012) 87–98.
[6] I.-C. Yeh, Modeling slump flow of concrete using second-order regressions and artificial neural networks, Cem. Concr. Compos. 29 (6) (2007) 474–480.
[7] S. Bhanja, B. Sengupta, Investigations on the compressive strength of silica fume concrete using statistical methods, Cem. Concr. Res. 32 (9) (2002) 1391–1394.
[8] S. Haykin, Neural Networks – A Comprehensive Foundation, Prentice Hall, 2004.
[9] I.B. Topcu, M. Sarıdemir, Prediction of compressive strength of concrete containing fly ash using artificial neural networks and fuzzy logic, Comput. Mater. Sci. 41 (3) (2008) 305–311.
[10] H.-C. Tsai, Y.-H. Lin, Predicting high-strength concrete parameters using weighted genetic programming, Eng. Comput. 27 (4) (2011) 347–355.
[11] M.A. Hearst, S.T. Dumais, E. Osman, J. Platt, B. Scholkopf, Support vector machines, IEEE Intell. Syst. Appl. 13 (4) (1998) 18–28.
[12] J.R. Koza, Genetic Programming – On the Programming of Computers by Means of Natural Selection, Complex Adaptive Systems, MIT Press, 1993.
[13] W. Banzhaf, F.D. Francone, R.E. Keller, P. Nordin, Genetic Programming: An Introduction on the Automatic Evolution of Computer Programs and Its Applications, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.
[14] Y. Peng, C. Yuan, X. Qin, J. Huang, Y. Shi, An improved gene expression programming approach for symbolic regression problems, Neurocomputing 137 (2014) 293–301. Advanced Intelligent Computing Theories and Methodologies. Selected papers from the 2012 Eighth International Conference on Intelligent Computing (ICIC 2012)
[15] F. Tang, S. Chen, X. Tan, T. Hu, G. Lin, Z. Kang, Discovery scientific laws by hybrid evolutionary model, Neurocomputing 148 (2015) 143–149.
[16] B.L. Villarreal, G. Olague, J. Gordillo, Synthesis of odor tracking algorithms with genetic programming, Neurocomputing 175 (2016) 1019–1032. Part B
[17] L. Chen, T.-S. Wang, Modeling strength of high-performance concrete using an improved grammatical evolution combined with macrogenetic algorithm, J. Comput. Civil Eng. 24 (3) (2009) 281–288.
[18] M. Castelli, L. Vanneschi, S. Silva, Prediction of high performance concrete strength using genetic programming with geometric semantic genetic operators, Expert Syst. Appl. 40 (17) (2013) 6856–6862.
[19] M. Schmidt, H. Lipson, Symbolic regression of implicit equations, in: Genetic Programming Theory and Practice VII, Springer, 2010, pp. 73–85.
[20] M.F. Brameier, W. Banzhaf, Linear Genetic Programming, Springer Science & Business Media, 2007.
[21] V.V. De Melo, Kaizen programming, in: Proceedings of the 2014 Conference on Genetic and Evolutionary Computation (GECCO '14), ACM, New York, NY, USA, 2014, pp. 895–902.
[22] M. Imai, Kaizen (Ky'zen), the Key to Japan's Competitive Success, McGraw-Hill, 1986.
[23] H. Gitlow, S. Gitlow, A. Oppenheim, R. Oppenheim, Tools and Methods for the Improvement of Quality, Irwin Series in Quantitative Analysis for Business, Taylor & Francis, 1989.
[24] V.V. de Melo, Breast cancer detection with logistic regression improved by features constructed by Kaizen programming in a hybrid approach, in: Evolutionary Computation (CEC), 2016 IEEE Congress, IEEE, 2016, pp. 16–23.
[25] I. Icke, J.C. Bongard, Improving genetic programming based symbolic regression using deterministic machine learning, in: Proceedings of the 2013 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2013, pp. 1763–1770.
[26] T. McConaghy, Ffx: Fast, scalable, deterministic symbolic regression technology, in: Genetic Programming Theory and Practice IX, Springer, 2011, pp. 235–260.
[27] V.V. de Melo, W. Banzhaf, Predicting high-performance concrete compressive strength using features constructed by Kaizen programming, in: Proceedings of the 2015 Brazilian Conference on Intelligent Systems (BRACIS), 2015, pp. 80–85.
[28] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, et al., Optimization by simulated annealing, Science 220 (4598) (1983) 671–680.
[29] H. Bozdogan, Model selection and Akaike's information criterion (AIC): the general theory and its analytical extensions, Psychometrika 52 (3) (1987) 345–370.
[30] E. Stinstra, G. Rennen, G. Teeuwen, Metamodeling by symbolic regression and pareto simulated annealing, Struct. Multidiscip. Optim. 35 (4) (2008) 315–326.
[31] T. Hengl, G.B. Heuvelink, D.G. Rossiter, About regression-kriging: from equations to case studies, Comput. Geosci. 33 (10) (2007) 1301–1315.
[32] I. Arnaldo, K. Krawiec, U.-M. O'Reilly, Multiple regression genetic programming, in: Proceedings of the 2014 Conference on Genetic and Evolutionary Computation (GECCO '14), ACM, New York, NY, USA, 2014, pp. 879–886.
[33] B. Efron, T. Hastie, I. Johnstone, R. Tibshirani, et al., Least angle regression, Ann. Stat. 32 (2) (2004) 407–499.
[34] R. Tibshirani, Regression shrinkage and selection via the lasso, J. R. Stat. Soc. Ser. B (Methodol.) 58 (1) (1996) 267–288.
[35] D.P. Searson, Gptips 2: an open-source software platform for symbolic data mining, in: Handbook of Genetic Programming Applications, Springer, 2015, pp. 551–573.
[36] M. Castelli, L. Trujillo, L. Vanneschi, S. Silva, et al., Geometric semantic genetic programming with local search, in: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, ACM, 2015, pp. 999–1006.
[37] H.-G. Ni, J.-Z. Wang, Prediction of compressive strength of concrete by neural networks, Cem. Concr. Res. 30 (8) (2000) 1245–1250.
[38] A. Öztaş, M. Pala, E. Özbay, E. Kanca, N. Çaglar, M.A. Bhatti, Predicting the compressive strength and slump of high strength concrete using neural network, Constr. Build. Mater. 20 (9) (2006) 769–775.
[39] I.-C. Yeh, L.-C. Lien, Knowledge discovery of concrete material using genetic operation trees, Expert Syst. Appl. 36 (3) (2009) 5807–5812.
[40] S.M. Mousavi, P. Aminian, A.H. Gandomi, A.H. Alavi, H. Bolandi, A new predictive model for compressive strength of HPC using gene expression programming, Adv. Eng. Softw. 45 (1) (2012) 105–114.
[41] V. Chandwani, V. Agrawal, R. Nagar, Modeling slump of ready mix concrete using genetic algorithms assisted training of artificial neural networks, Expert Syst. Appl. 42 (2) (2015) 885–893.
[42] M.-Y. Cheng, P.M. Firdausi, D. Prayogo, High-performance concrete compressive

strength prediction using genetic weighted pyramid operation tree (GWPOT), Eng. Appl. Artif. Intell. 29 (2014) 104–113.
[43] L. Chen, C.-H. Kou, S.-W. Ma, Prediction of slump flow of high-performance concrete via parallel hyper-cubic gene-expression programming, Eng. Appl. Artif. Intell. 34 (2014) 66–74.
[44] T. Perkis, Stack-based genetic programming, in: Proceedings of the First IEEE Conference on Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence, IEEE, 1994, pp. 148–153.
[45] L. Spector, Autoconstructive evolution: push, pushgp, and pushpop, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), vol. 137, 2001.
[46] W. La Cava, T. Helmuth, L. Spector, K. Danai, Genetic programming with epigenetic local search, in: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, ACM, 2015, pp. 1055–1062.
[47] M. Oltean, C. Groşan, L. Dioşan, C. Mihăilă, Genetic programming with linear representation: a survey, Int. J. Artif. Intell. Tools 18 (02) (2009) 197–238.
[48] W. Banzhaf, Genetic programming for pedestrians, in: Proceedings of the 1993 International Conference on Genetic Algorithms (ICGA), 1993, p. 628.
[49] M. Stone, An asymptotic equivalence of choice of model by cross-validation and Akaike's criterion, J. R. Stat. Soc. Ser. B (Methodol.) 39 (1) (1977) 44–47.
[50] M. Lichman, UCI machine learning repository, 2013, http://archive.ics.uci.edu/ml.
[51] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, The weka data mining software: an update, SIGKDD Explor. Newsl. 11 (1) (2009) 10–18.

Vinícius Veloso de Melo is an associate professor at the Institute of Science and Technology-UNIFESP, São Paulo, Brazil. He obtained his B.Sc. in Computer Science from PUC-MG, Brazil, in 2002; his M.Sc. and Ph.D. in Computer Science from University of São Paulo, Brazil, in 2005 and 2009, respectively. He has published papers in international peer-reviewed journals and conferences. His current research interests are evolutionary computation, metaheuristics, and machine learning.

Wolfgang Banzhaf is the John R. Koza Chair in Genetic Programming at Michigan State University, USA. Previously, he was University Research Professor in the Department of Computer Science of Memorial University of Newfoundland, Canada, where he served as head of department 2003–2009 and 2012–2016. His research interests are in the field of bio-inspired computing, notably evolutionary computation and complex adaptive systems. Studies of self-organization and the field of Artificial Life are also of very much interest to him. Recently he has become more involved with network research as it applies to natural and man-made systems.