Kaizen Programming

Vinícius Veloso de Melo
Institute of Science and Technology (ICT)
Federal University of São Paulo (UNIFESP)
São José dos Campos, SP, Brazil
vinicius.melo@unifesp.br

ABSTRACT

This paper presents Kaizen Programming, an evolutionary tool based on the concepts of Continuous Improvement from Kaizen Japanese methodology. One may see Kaizen Programming as a new paradigm since, as opposed to classical evolutionary algorithms where individuals are complete solutions, in Kaizen Programming each expert proposes an idea to solve part of the problem, thus a solution is composed of all ideas together. Consequently, evolution becomes a collaborative approach instead of an egocentric one. An idea's quality (analog to an individual's fitness) is not how good it fits the data, but a measurement of its contribution to the solution, which improves the knowledge about the problem. Differently from evolutionary algorithms that simply perform trial-and-error search, one can determine, exactly, parts of the solution that should be removed or improved. That property results in the reduction in bloat, number of function evaluations, and computing time. Even more important, the Kaizen Programming tool, proposed to solve symbolic regression problems, builds the solutions as linear regression models - not linear in the variables, but linear in the parameters, thus all properties and characteristics of such statistical tool are valid. Experiments on benchmark functions proposed in the literature show that Kaizen Programming easily outperforms Genetic Programming and other methods, providing high quality solutions for both training and testing sets while requiring a small number of function evaluations.

Categories and Subject Descriptors

I.2.2 [**Artificial Intelligence**]: Automatic Programming, Program Synthesis, Program Modification; D.1.2 [**Programming Techniques**]: Automatic Programming

Keywords

Symbolic regression, Linear regression, Evolutionary algorithm, Collaborative problem solving, Genetic programming

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.
Copyright 2014 ACM 978-1-4503-2662-9/14/07 ...$15.00.
http://dx.doi.org/10.1145/2576768.2598264.

1. INTRODUCTION

Genetic programming (GP) [8] is an evolutionary computation algorithm, inspired by the Genetic Algorithm (GA) [3], which evolves computer programs to solve a particular task. GP has been employed to automatically produce codes to solve (see [13]): curve-fitting, data modelling, symbolic regression, image and signal processing, financial trading, time series, industrial process control, bioinformatics, among others. Despite its successful application in several domains, there are some known issues with GP and related algorithms [11, 7], for instance, bloat, lack of heuristics, slow speed (large population size), and low success rate.

One of the main reasons for those issues is that the methods perform blind/random search. The only guide, or at least the most important, is pressure selection, which selects the best solutions to be mixed and to continue in the population. On the other hand, recombination and mutation are, in general, simple guesses: select two *random* sub-trees for swapping, generate a *random* sub-tree, generate a *random* ephemeral constant. The evolutionary process is done expecting that a new *random* solution presents a better fitness than the current solution. According to Korns [7], these characteristics make automated design algorithms hard to be accepted in the industry.

This paper presents a novel approach in order to overcome some of the presented drawbacks. The purpose of this paper is to improve the mechanism that guides the search, resulting in the reduction in bloat, population size, and number of function evaluations while increasing solution quality. The proposed method is called Kaizen Programming and is based on the Japanese methodology of Continuous Improvement of solutions in a collaborative problem solving approach.

2. RELATED WORKS

Some related works that try to improve tools for symbolic regression are presented next.

Moraglio et al. [10] presented the geometric crossover that searches directly the space of the underlying semantics of the programs. The operator transforms the syntax of an individual in order to perform a ball mutation on the semantic space. Consequently, any random improvement moves the current solution towards the optimum. Thus, as they pointed out in their paper, all random codes can be used, resulting in huge black-box solutions.

Korns [6] stated that all current approaches needed serious improvements. In the last years, he proposed several modifications in order to increase the quality of Genetic Programming and created an engine where numerical constants are

replaced by abstract constants, which are optimized by continuous meta-heuristics such as Particle Swarm Optimization and Differential Evolution.

Pennachin et. al [12] proposed the improvement of both performance and robustness of GP by the use of Affine Arithmetic to estimate the output range of expressions given their inputs over the training data. Estimated outputs with values too far from the desired output range are identified and the corresponding trees are discarded from the solutions. That strategy avoided extreme errors on the testing data.

The present work may be more related to ensembles of trees [14], a method in which several models (trees) are independently evolved, and then combined to provide a better forecast. The final fused prediction may be the average of the individual predictions, a weighted average, or another statistic. However, the technique proposed herein employs a distinct strategy, as will be presented in the next sections.

All these approaches improve solution quality, but bloat and the number of evaluations are still high, except for the works of Korns, who employs user specified goal expressions. With Kaizen Programming, this can be reduced because it can identify the contribution of parts of the solution. The Kaizen methodology is presented next.

3. THE KAIZEN METHODOLOGY

The Japanese word Kaizen means "Good Change," and is adopted as a philosophy of work [4] which means continuous improvement. Kaizen Event is the term given to an event consisting of a team (of workers and managers) working together for a brief period to find effective solutions to identified business problems. In an industry, for instance, it could be cycle time reduction, waste reduction, speed improvement, or any other problem.

Kaizen teams are usually small groups of about five individuals who spend their time for a few days until a particular issue is solved or a significant improvement is achieved according to the mission statement. The individuals are selected by their knowledge in the area where the issue is present or because they work elsewhere, but are also impacted by the issue. This way, different aspects of the problem may be analyzed in order to identify the effects of the proposed changes.

In a kaizen event, the team of experts meets for instructions. During the meeting, the experts discuss critical issues, do a brainstorming of ideas to provide solutions for the problem, and develop action plans. In general, the Plan-Do-Check-Act (PDCA [2]) methodology is employed to guide the continuous improvement process.

In PDCA, actions are planned, executed, checked, and new actions are taken based on the results. The cycle is repeated until the mission is complete. At each cycle, more knowledge on the problem is gained and every action can be evaluated according to its effectiveness in helping solving the issue. Therefore, at each cycle, the experts have more information to avoid bad actions and guide the search towards the solution. One of the interesting aspects is that complex actions that do not provide significant improvement may be discarded, reverting back to old simple yet not so efficient procedures.

4. KAIZEN PROGRAMMING

Kaizen Programming (KP), proposed herein, is a novel evolutionary tool based on the concepts of the Kaizen methodology. KP is a computational implementation of a Kaizen event with PDCA.

In usual evolutionary algorithms such as GP, an individual is a mapping of the problem's solution (a complete solution), and different individuals are different solutions for the same problem. That creates diversity for the search mechanism and gives the user the possibility of choice among the final best solutions.

When compared to GP, KP may be seen as a new paradigm for automated design of algorithms because KP individuals are not complete solutions, but part of it. Consequently, evolution becomes a collaborative approach instead of an egocentric one.

In KP, a team of experts is formed to propose ideas to solve a problem, then are joined to become a solution. The quality of the solution is how good it solves the problem, and the quality of an idea is a measurement of its contribution to the solution. Therefore, differently from general GP and other evolutionary algorithms that perform trial-and-error search, in KP one can determine, exactly, which parts of the solution should be removed or improved. Consequently, KP is a collaborative problem-solving approach and that the experts have to contribute by providing better ideas at each cycle. Such property results in a reduction in bloat, smaller population sizes, and lower number of function evaluations.

Not all experts may provide useful contributions all the time, but their knowledge is shared with the group to improve everyone's ideas. Algorithm 1 presents a high-level KP algorithm, and Figure 1 presents a flowchart.

Algorithm 1 The Kaizen Programming algorithm.

1. Create a team of experts;
2. Define target and set it as not achieved;
3. Do
   (a) PLAN: the team performs a brainstorming and each expert proposes an idea to solve part of the problem;
   (b) DO: the current ideas and the new ones (created from the second iteration) are applied to the problem and joined to become a complete solution;
   (c) CHECK: evaluate the solution then each single idea (old and new) is analyzed and its contribution to solve the problem is measured;
   (d) ACT: if the solution is improved, then the best ideas are selected and become the new standard, which is presented to the team along with each contribution, improving the knowledge of the problem. Create another kaizen event with a new team if the current one gets stuck in a local optimum;
4. Loop while target is not achieved;
5. Return the ideas with significant contribution and the final solution.

In this paper, KP is employed to solve symbolic regression problems. As mentioned before, KP will build solutions as linear regression models, where each variable of the model is called a *regressor*. The next subsections explain, in more detail, the PDCA cycle of KP.

4.1 PLAN

In this phase KP has to generate ideas, which are proposed by the team of experts to solve parts of the problem.

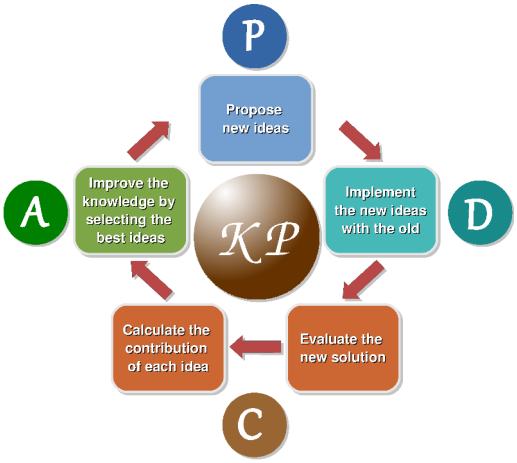


Figure 1: Basic Kaizen Programing flowchart.

Since the problem is new to all experts there are no initial standard and no initial knowledge. After the first cycle, a standard is established and the team acquires knowledge on the problem, improving it over the cycles.

For the symbolic regression problem, an idea from an expert is a mathematical expression that will be implemented and transformed into a *new* regressor (K_j, $j = 1, ..., t$) of the model. Therefore, the team size (t) corresponds to the *new* number of regressors. If the original problem has only one regressor (x) and KP is configured with a team of three experts, then one may have, for example, $K_1 = x^2$; $K_2 = sin(x)$; $K_3 = -x + 3/x$.

4.2 DO

After all experts present their ideas these are individually applied to the problem (calculated). Therefore, the result is a matrix $TRIAL_{n,w}$, where n is the number of observations in the sample dataset used for training and w is the new number of regressors. The new ideas are joined with the current standard $STD_{n,t}$ and all of them will be used to build a new model. The solution quality and the contributions of the current standard can be changed for better or worse. Below there is an example of regressors matrix $K_{n,(t+w)}$, where t and w may have different values as explained later.

$$K = \begin{bmatrix} STD_{11} & \dots & STD_{1t} & TRIAL_{11} & \dots & TRIAL_{1w} \\ \vdots & \dots & \vdots & \vdots & \dots & \vdots \\ STD_{n1} & \dots & STD_{nt} & TRIAL_{n1} & \dots & TRIAL_{nw} \end{bmatrix}.$$

4.3 CHECK

In this phase KP creates a linear model using K and the set of expected outputs for the problem (y), and optimizes it using Ordinary Least-Squares (OLS). OLS is a statistical tool that minimizes the sum of the squared residuals, which are the squared vertical distances between the observed responses in the sample dataset and the responses predicted by the linear model. Using the previous example with K_1, K_2, and K_3, the model is created in the form:

$$\hat{y_i} = \hat{\beta}_1 K_{i,1} + \hat{\beta}_2 K_{i,2} + \hat{\beta}_3 K_{i,3}, \quad (1)$$

where $\hat{y}_i, i = 1, ..., n$ is the calculated output for an and $\hat{\beta}_1$, $\hat{\beta}_2$, and $\hat{\beta}_3$ are the coefficients estimated by OLS using

$$\hat{\beta} = (K^T K)^{-1} K^T y. \quad (2)$$

Therefore, it is easy to notice that all models generated by KP are linear *in the parameters* since $\hat{\beta}_i$ are *never* within the regressors (K_i) generated by the experts. Also, one can notice that Eq. 1 does not explicitly contain the intercept. In fact, one of the regressors can be a constant generated by KP.

The linear model is used not only to perform the linear scaling and evaluate the solution, but also to guide the search for better ideas. Hence, it is important to be clear that the estimated parameters are *not* included in the solution during the search, *only* in the final solution.

In order to guide the search one must check the contribution of each regressor to the model, that is, the contribution of each idea to the solution. Hence, the quality of an idea (*fitness*) is not the sum of residuals.

The contribution is measured as the $p-value$ of the regressor, calculated using $p = 2 * (1 - T(df, |t|))$, where T is the cumulative distribution of the studentâĂŹs t-distribution, df is the residual degrees of freedom, and $|t|$ is the absolute value of the observed t-statistic. If $p-value$ is not significant ($p-value > \alpha$), then the idea was not useful to solve the problem and may be discarded in the next cycle. Furthermore, if the $p-value$ of a regressor K_j is significant, but the absolute value of the corresponding coefficient $\hat{\beta}_j$ is lower than a predefined threshold θ, then K_j is considered not significant because $\hat{\beta}_j K_j \approx 0$. Thus, K_j is penalized and set to 1.0. The penalty also applies to duplicated ideas, for which OLS calculates the coefficient as NA.

The use of the contribution is a very important characteristic of the method to automatically act as bloat control. On the other hand, if an idea is improved it can automatically replace the old one.

$$contrib(K_j) = \begin{cases} 1.0, & if\ p-value(K_j) = NA \\ 1.0, & if\ p-value(K_j) > \alpha \\ 1.0, & if\ |\hat{\beta}_j| < \theta \\ p-value(K_j), & otherwise \end{cases}. \quad (3)$$

The solution (model) quality is given by a goodness of fit measure by comparing how much the initial variation in the sample dataset can be reduced by regressing onto K. In statistics, the coefficient of determination R^2 is the proportion of variability in the dataset that is accounted for by a statistical model. While R^2 is a goodness of fit measure, adjusted R^2 is a comparative measure of suitability of models with distinct sets of regressors. R^2 may increase just because the number of regressors increased. On the other hand, adjusted R^2 increases only if the new regressor improves the model more than would be expected by chance. Since KP generates distinct models with different number of regressors (selected according to their contribution) each cycle, adjusted R^2 can be used to compare the models and select the one that presents higher value without being mistaken by the number of regressors. Adjusted R^2 is calculated as follows, where p is the number of regressors:

$$Adj.R^2 = 1 - (1 - R^2)\frac{n-1}{n-p-1} = R^2 - (1 - R^2)\frac{p}{n-p-1}, \quad (4)$$

Adjusted R^2, which was used to stop the search, can be easier to interpret than RMSE, for instance, because it is scale-free, w.r.t. the response values of the problem, and the result of this statistic is in the range [0, 1] where 1 means a perfect fit. Actually, negative values may occur but are not expected and correspond to a very low-quality model. The corresponding algorithm of this phase is presented in Algorithm 2.

Algorithm 2 The Check phase of KP.

Input: K, y, α, θ, $MAXVALUE$ //*the penalty*
Output: Fit, $Contrib$ //*the Contributions*

1. Build the model
2. Calculate the contribution of the regressors
3. If there are no significant regressors
   (a) Set Fit as MAXVALUE and the $Contrib$ as in Eq. 3
4. Else
   (a) Build a new model using only the significant regressors
   (b) Calculate the contribution of the regressors
   (c) Calculate Fit as the goodness-of-fit (Adjusted R^2)

4.4 Act

In this stage, the experts will propose new ideas to solve the problem, based on the knowledge acquired in the Check phase due to the contributions of the current ideas in the standard. The new solution replaces the standard *only* if it is better.

To generate a new idea, the expert may combine his current idea with the idea proposed by another expert, and then improve the combined solution with another new information. However, not all experts have to present new ideas every cycle. That procedure works in a similar fashion as crossover and mutation of a GP algorithm.

An additional characteristic of KP is that it uses a small number of experts and works as a hill-climbing method. Thus, it is necessary to use a restarting procedure to escape from a local optimum if the standard is stagnated (remains the same) for a period defined by the user. That means that the investigated issue was underestimated and it is necessary to create a new Kaizen event with a different and larger team. Hence, the restart procedure saves the current standard, increases the number of experts by EF (expansion factor), and generates new random ideas ignoring the old ones. The method restarts the PDCA cycle until some stopping criterion is achieved. The best standard discovered out of all restarts is returned as the final solution.

4.5 Resulting solution

The resulting solution (standard) is an additive function that approximates the regressors on the response variable. It is not expected that the exact function which generated the response will be found (because KP generates a composite function). For instance, take the following example function:

$$f(x_0, x_1, x_2, x_3, x_4) = -5.41 + 4.9\frac{x_3 - x_0 + \frac{x_1}{x_4}}{3x_4}, \qquad (5)$$

where $x_i = U(-50, 50)$. The best solution found by a KP is shown in Figure 2.

$f_{found}(x_0, x_1, x_2, x_3, x_4) =$
(0.261141430577
∗(−1 ∗ ((1/((−1 ∗ (((1/(−1 ∗ (x0)))
∗(1/ − 6.2545928837239835))))
∗(−1 ∗ (x4)))))))
+(−1.63333333333
∗(−1 ∗ ((1/((−1 ∗ (((1/(x4 ∗ x3)) ∗ x4)))
∗(−1 ∗ (x4)))))))
+(−1.63333333333
∗(−1 ∗ ((1/((−1 ∗ (((1/x1) ∗ x4)))
∗(−1 ∗ (x4)))))))
+(0.6446225282
∗(−1 ∗ ((1/(1/8.3925084267666)))))

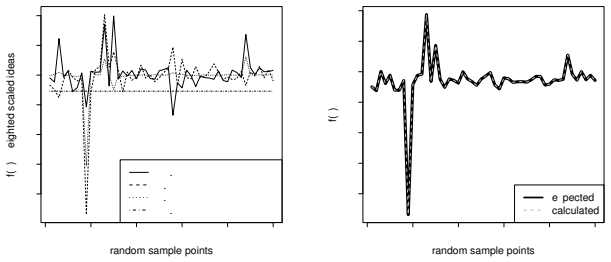
(a)

$f_{simplified}(x_0, x_1, x_2, x_3, x_4) =$
$-1.6333333333324 * \frac{x_0}{x_4}$
$+1.63333333333 * \frac{x_1}{x_2}$
$+1.63333333333 * \frac{x_3}{x_4}$
-5.41000000000209

(b)

Figure 2: (a) the best solution found by a KP run; (b) the simplified expression.

That set of ideas can be plot with random x_i values as in Figure 3(a), whereas the final calculated outputs $\hat{y}$ and the expected real outputs are shown in Figure 3(b). The calculated output (dashed line) is exactly inside the expected output (solid line), meaning that the exact function was found besides in a different form.



(a) (b)

Figure 3: (a) Plot of significant ideas; (b) Expected (Eq. 5) and calculated (Fig 2(b)).

Next section presents experimental results of Kaizen Programming to solve symbolic regression problems.

5. EXPERIMENTAL RESULTS

This next section presents a comparison of Kaizen Programming against related methods in solving a set of well-known symbolic regression benchmark functions.

5.1 Benchmark functions

The functions tested in this work, whose definitions one can see in Table 1, were taken from [9]. The description of the table, as shown in [9] is: variable names are, in order, x, y, z. Some benchmarks intentionally omit variables from the function. U[a,b,c] is c uniform random samples drawn from a to b, inclusive, for the variable. E[a,b,c] is a grid of points evenly spaced (for this variable) with an interval of c, from a to b inclusive. Testing and training sets are independent.

In order to compare to other methods, tests were also performed in the well-known Nguyen benchmark functions shown in Table 2

5.2 Implementation and configuration of the algorithms

GP was implemented using DEAP (Distributed Evolutionary Algorithms in Python, [1]), an evolutionary computation framework for rapid prototyping and testing of ideas.

Table 1: Symbolic regression functions (Keijzer functions) and the datasets as proposed in [9].

Function	Training data	Testing data
$keijzer1(x) = 0.3xsin(2\pi x)$	E[-1, 1, 0.1]	E[-1, 1, 0.001]
$keijzer2(x) = 0.3xsin(2\pi x)$	E[-2, 2, 0.1]	E[-2, 2, 0.001]
$keijzer3(x) = 0.3xsin(2\pi x)$	E[-3, 3, 0.1]	E[-3, 3, 0.001]
$keijzer4(x) = x^3e^{-x}cos(x)sin(x)(sin^2(x)cos(x) - 1)$	E[0, 10, 0.05]	E[0.05, 10.05, 0.05]
$keijzer5(x) = (30xz)/((x - 10)y^2)$	x,y: U[-1, 1, 1000] z: U[1, 2, 1000]	x,y: U[-1, 1, 10000] z: U[1, 2, 10000]
$keijzer6(x) = \sum_i^x 1/i$	E[1, 50, 1]	E[1, 120, 1]
$keijzer7(x) = lnx$	E[1, 100, 1]	E[1, 100, 0.1]
$keijzer8(x) = \sqrt{x}$	E[1, 100, 1]	E[1, 100, 0.1]
$keijzer9(x) = arcsinh(x)$	E[1, 100, 1]	E[1, 100, 0.1]
$keijzer10(x) = x^y$	U[0, 1, 100]	E[0, 1, 0.1]
$keijzer11(x) = xy + sin((x - 1)(y - 1))$	U[-3, 3, 20]	E[-3, 3, 0.01]
$keijzer12(x) = x^4 - x^3 + y^2/2 - y$	U[-3, 3, 20]	E[-3, 3, 0.01]
$keijzer13(x) = 6sin(x)cos(y)$	U[-3, 3, 20]	E[-3, 3, 0.01]
$keijzer14(x) = 8/(2 + x^2 + y^2)$	U[-3, 3, 20]	E[-3, 3, 0.01]
$keijzer15(x) = x^3/5 + y^3/2 - y - x$	U[-3, 3, 20]	E[-3, 3, 0.01]

Table 2: Symbolic regression functions (Nguyen functions) and the datasets as proposed in [9].

Function	Training/Testing data
$nguyen1(x) = x^3 + x^2 + x$	x: U[-1, 1, 20]
$nguyen2(x) = x^4 + x^3 + x^2 + x$	x: U[-1, 1, 20]
$nguyen3(x) = x^5 + x^4 + x^3 + x^2 + x$	x: U[-1, 1, 20]
$nguyen4(x) = x^6 + x^5 + x^4 + x^3 + x^2 + x$	x: U[-1, 1, 20]
$nguyen5(x) = sin(x^2)cos(x) - 1$	x: U[-1, 1, 20]
$nguyen6(x) = sin(x) + sin(x + x^2)$	x: U[-1, 1, 20]
$nguyen7(x) = log(x + 1) + log(x^2 + 1)$	x: U[0, 2, 20]
$nguyen8(x) = \sqrt{x}$	x: U[0, 4, 20]
$nguyen9(x) = sin(x) + sin(y^2)$	x, y: U[-1, 1, 100]
$nguyen10(x) = 2sin(x)cos(y)$	x, y: U[-1, 1, 100]

DEAP provides a complete GP implementation, with several operators and flexibility for improvements.

KP was also implemented using GP modules from DEAP. Therefore, the same modules used in the GP implementation were used in KP, which means that KP was not privileged by a better implementation of the genetic operators. The statistical parts of KP (the OLS method, Adjusted R^2, $p - values$, among others) used the *lm* (linear model) function from R programming language. The connection between Python and R was performed by the Rpy2 Python package.

For the tests using *Keijzer* functions, the methods were configured as shown in Table 3 with the function set proposed in [9].

Table 3: Run and evolutionary parameter values for *Keijzer* functions.

Parameter	Value (KP/GP50/GP500)
Initial Experts/Population size	8/50/500
Ideas/Population generator	GP Ramped
Generations	2000/500/50
Stagnation	25% of the generations
Factor (EF) to increase experts/population	ceiling (10% of popsize)
Tournament Selection	1/3/3
Crossover probability	1.0/0.9/0.9
Idea combinator/crossover operator	One-point
Mutation probability	1.0/0.05/0.05
Idea improver/mutation operator	90% GP Uniform Mutation 10% GP ERC Mutation
Max. depth	2/15/15
Non-terminals	$+$, $\times$, $\frac{1}{n}$, $-n$, $\sqrt{n}$,
Terminals	x, Constants (ERC) are random values from $N(\mu = 0, \sigma = 5)$.
Solution quality/fitness	$Adjusted\, R^2/R^2/R^2$
Value-to- reach (VTR)	$1 - fitness < 1e - 5$
Trials	50
θ (for KP Check)	$1e - 4$
α (for KP Check)	0.05

For the test using *Nguyen* functions, KP was configured as shown in Table 4 with the function set proposed in [5].

Table 4: Modified run and evolutionary parameter values for *Nguyen* functions. The configuration of the other parameters are in Table 3.

Parameter	Value (KP)
Maximum number of node evaluations	100,000
Stagnation	25% of the generations
Max. depth	8
Non-terminals	$+$, $\times$, $-$, $/(protected)$, sin, cos, exp, $log(protected)$
Terminals	x, Constant 1 (*nguyen1* to *nguyen8*), y (*nguyen9* and *nguyen10*)
Hit	when the solution has an absolute error < 1e-2 on a fitness case
Successful run	when the solution scores hits on all fitness cases
Trials	100

5.3 Evaluation

A descriptive analysis of the results obtained in the experiments is presented with the median best values for each benchmark function and statistical comparison between KP and GP50 (GP with popsize of 50 individuals), and KP and GP500 (GP with popsize of 500 individuals). The Wilcoxon Mann-Whitney test at a significance level $\alpha = 0.01\%$ was performed using the RMSE Testing results of 50 trials. No statistical comparison was performed for the Training results.

5.4 Results

Regarding the *Keijzer* benchmark functions, Table 5 presents the results of the training set whereas Table 6 presents the results of the testing set.

The *Nguyen* benchmark functions use the same interval for training and testing, but the sets of points are distinct. For these tests, it was employed the same methodology presented by Karaboga et al. in [5]. The descriptive analysis is shown in Table 7 and the comparison with other methods is shown in Table 8.

5.5 Discussion

One can notice in Table 3 that, as suggested in [9], not all operators employed in the benchmark functions (see Table 1) are available in the function set. Therefore, the exact solution can be hard or impossible to find. Thus, it is expected that KP fails for some runs.

In this work, the generation of ideas is performed using the same well-known evolutionary operators employed by GP: population initialization, crossover, mutation, and selection. KP guides the search by identifying the partial solutions to be improved, but the search is still random as in GP.

Given that both methods can restart when stagnation is detected, i.e the best solution does not change during a period, the number of function evaluations (NFEs) can vary. For KP, at each cycle the current ideas have to be reevaluated because they will be part of a new solution with the new ideas. Therefore, at each generation there are 16 evaluations instead of 8. However, the number of generations was set to provide similar NFEs as presented in Table 5. GP was configured with populations 50 and 500 as commonly used in related works in the literature.

Table 5: Training Results using *Keijzer* benchmark functions (descriptive analysis).

		KP			GP50			GP500		
Function	**Statistic**	**Adj.R^2**	**RMSE**	**NFEs**	**R^2**	**RMSE**	**NFEs**	**R^2**	**RMSE**	**NFEs**
	Min	9.996e-01	1.926e-06	328	1.404e-01	4.833e-02	25050	4.123e-01	8.001e-02	26700
*keijzer*1	Median	**1.000e+00**	**2.210e-04**	9492	4.144e-01	8.969e-02	25050	4.144e-01	8.969e-02	30450
	Max	1.000e+00	2.279e-02	32260	8.300e-01	1.087e-01	25050	5.340e-01	8.985e-02	32170
	Min	1.252e-01	5.419e-04	2880	3.439e-02	1.031e-01	25050	5.502e-02	2.174e-01	28460
*keijzer*2	Median	**8.994e-01**	**6.747e-02**	32190	1.957e-01	2.159e-01	25050	1.392e-01	2.234e-01	30550
	Max	1.000e+00	2.194e-01	32380	8.165e-01	2.366e-01	25050	1.844e-01	2.341e-01	32310
	Min	1.259e-01	9.983e-02	32100	0.000e+00	2.469e-01	25050	3.263e-02	3.346e-01	28300
*keijzer*3	Median	**3.830e-01**	**2.654e-01**	32170	1.064e-01	3.437e-01	25050	7.050e-02	3.506e-01	30300
	Max	9.132e-01	3.359e-01	32260	5.389e-01	3.636e-01	25050	1.530e-01	3.576e-01	31950
	Min	5.120e-01	3.533e-02	32180	0.000e+00	2.517e-01	25050	0.000e+00	3.155e-01	25500
*keijzer*4	Median	**7.408e-01**	**1.631e-01**	32330	7.186e-03	3.164e-01	25050	1.489e-03	3.174e-01	25500
	Max	9.887e-01	2.658e-01	32540	3.725e-01	3.186e-01	25050	1.383e-02	4.051e-01	26550
	Min	1.000e+00	8.122e-04	368	5.197e-02	1.918e-02	25050	8.050e-01	8.048e-02	28760
*keijzer*5	Median	**1.000e+00**	**1.524e-03**	2180	8.988e-01	1.756e-01	25050	8.582e-01	2.036e-01	32020
	Max	1.000e+00	9.060e-03	27180	9.987e-01	5.428e-01	25050	9.797e-01	2.417e-01	32740
	Min	1.000e+00	6.985e-16	56	1.000e+00	9.899e-01	51	1.000e+00	9.899e-01	501
*keijzer*6	Median	1.000e+00	**9.216e-16**	56	1.000e+00	9.899e-01	51	1.000e+00	9.899e-01	501
	Max	1.000e+00	1.179e-14	14510	1.000e+00	9.899e-01	51	1.000e+00	9.899e-01	501
	Min	1.000e+00	2.964e-06	56	9.391e-01	6.074e-02	25050	9.955e-01	1.049e-01	28590
*keijzer*7	Median	**1.000e+00**	**4.613e-04**	80	9.978e-01	1.747e-01	25050	9.977e-01	1.813e-01	30660
	Max	1.000e+00	1.776e-02	224	9.997e-01	9.258e-01	25050	9.992e-01	2.512e-01	32480
	Min	1.000e+00	7.911e-16	56	1.000e+00	0.000e+00	51	1.000e+00	0.000e+00	501
*keijzer*8	Median	1.000e+00	3.499e-15	56	1.000e+00	**0.000e+00**	51	1.000e+00	**0.000e+00**	501
	Max	1.000e+00	1.340e-02	160	1.000e+00	0.000e+00	101	1.000e+00	0.000e+00	501
	Min	1.000e+00	9.773e-06	56	9.479e-01	8.466e-02	25050	9.694e-01	1.118e-01	25500
*keijzer*9	Median	**1.000e+00**	**1.587e-03**	88	9.985e-01	1.714e-01	25050	9.991e-01	1.354e-01	30300
	Max	1.000e+00	2.912e-02	248	9.996e-01	1.006e+00	25050	9.994e-01	7.704e-01	32050
	Min	1.000e+00	2.057e-03	2616	9.361e-01	4.198e-02	25050	9.297e-01	9.039e-02	30760
*keijzer*10	Median	**1.000e+00**	**2.998e-03**	32210	9.565e-01	1.562e-01	25050	9.575e-01	1.526e-01	32870
	Max	1.000e+00	1.946e-02	32330	9.968e-01	1.887e-01	25050	9.863e-01	1.917e-01	32870
	Min	9.709e-01	8.244e-02	32090	8.980e-01	3.121e-01	25050	8.862e-01	4.060e-01	31930
*keijzer*11	Median	**9.936e-01**	**1.934e-01**	32160	9.577e-01	6.452e-01	25050	9.597e-01	6.359e-01	32680
	Max	9.985e-01	4.542e-01	32230	9.906e-01	7.936e-01	25050	9.899e-01	7.766e-01	32870
	Min	9.999e-01	3.960e-15	328	8.275e-01	1.028e+00	25050	8.650e-01	1.373e+00	30220
*keijzer*12	Median	**1.000e+00**	**2.628e-02**	2692	9.751e-01	3.527e+00	25050	9.802e-01	3.565e+00	31190
	Max	1.000e+00	7.772e-01	32220	9.985e-01	1.204e+01	25050	9.981e-01	9.722e+00	32300
	Min	8.947e-01	8.033e-02	32120	1.926e-01	6.009e-01	25050	1.368e-01	1.645e+00	28340
*keijzer*13	Median	**9.765e-01**	**3.482e-01**	32190	4.064e-01	2.217e+00	25050	3.895e-01	2.261e+00	31190
	Max	9.990e-01	8.525e-01	32310	9.626e-01	3.090e+00	25050	6.765e-01	2.889e+00	32870
	Min	9.890e-01	2.354e-03	7976	6.821e-01	8.574e-02	25050	6.858e-01	2.988e-01	28860
*keijzer*14	Median	**9.984e-01**	**5.319e-02**	32180	8.358e-01	5.821e-01	25050	7.935e-01	6.968e-01	30890
	Max	1.000e+00	1.294e-01	32310	9.955e-01	1.031e+00	25050	9.333e-01	9.405e-01	32740
	Min	1.000e+00	8.585e-16	272	4.192e-01	4.882e-01	25050	4.722e-01	9.975e-01	29910
*keijzer*15	Median	**1.000e+00**	**3.923e-15**	2732	7.752e-01	1.520e+00	25050	7.609e-01	1.705e+00	32200
	Max	1.000e+00	3.308e-02	11000	9.804e-01	2.387e+00	25050	9.398e-01	2.542e+00	32870

The stopping criteria are the maximum number of generations or the value-to-reach (VTR). VTR was set as a high-quality model ($R^2 > 0.99999$) instead of a small RMSE because R^2 is scale-free with a maximum value of 1.0. KP uses Adj.R^2 because the number of regressors can change during the process because only the significant ideas are used to build the model. Also, Adj.R^2 tends to be lower than R^2, thus a high Adj.R^2 can be better than a high R^2.

For the 15 functions, the training results show that KP achieved the highest values of R^2 for 12 functions, with two ties (*keijzer*6 and 8), but the RMSE value of KP in *keijzer*6 is largely better. KP found very high-quality models (median Adj.$R^2 > 0.99$) for 11 functions, whereas GP50 and GP500 achieved such quality for only four functions.

For several functions KP required a much lower NFEs (median) than GP (see *keijzer*1, 5, 9, 12, 15). In fact, KP is much faster in finding good models, but the final adjustments can take time because it uses the same random evolutionary procedures employed by GP. Lower quality models (Adj.$R^2 > 0.95$) were found by KP using just hundreds of evaluations.

In the first experiment, the goal was to verify if KP could outperform GP, thus the statistic tests were carried out as KPxGP50, and KPxGP500. With respect to the testing results (see Table 6), one can see that KP showed statistically inferior quality results for functions *keijzer*3, 8, 11, 13, and 14. For the functions with very large values (say $> 1e5$) it was detected that the solution found by KP included the operator $\frac{1}{n}$. When n approaches zero, the result increases exponentially. If the training dataset does not include small values for n, then KP can not treat this extreme behavior, thus resulting in an over-fitted model.

For functions *keijzer*1, 2, 5, 6, 7, 10, and 15, KP errors were smaller by one order of magnitude or more. GP50 and GP500 presented similar results to each other, with GP500 having smaller variance.

The second experiment, using Nguyen benchmark functions, provides the comparison of KP against state-of-the-art techniques recently presented in the literature [5]. For these problems, the method has to perform curve-fitting where the maximum absolute error for any of the points (fitness cases) is lower than 0.01, therefore, resulting in a successful run.

Table 6: Testing Results using *Keijzer* benchmark functions. KP x GP50 and KP x GP500. Signal "=" means that the results of KP and GP are statistically equal, "<" that the results of GP are statistically better, and ">" that are statistically worse.

Function	Statistic	KP	GP50	GP500
	Min	1.067e-05	4.528e-02	7.876e-02
*keijzer*1	Median	**4.360e-04**	8.260e-02	8.199e-02
	Max	5.737e+06	2.423e+03	1.304e+00
	P-value	-	>	>
	Min	6.858e-04	1.729e-01	2.089e-01
*keijzer*2	Median	**6.849e-02**	2.391e-01	2.217e-01
	Max	8.229e+00	2.345e+01	1.002e+00
	P-value	-	>	>
	Min	9.673e-02	2.950e-01	3.412e-01
*keijzer*3	Median	2.065e+00	3.889e-01	**3.526e-01**
	Max	9.984e+01	9.999e+08	2.710e+00
	P-value	-	=	<
	Min	3.541e-02	2.516e-01	3.155e-01
*keijzer*4	Median	**1.612e-01**	3.165e-01	3.174e-01
	Max	2.656e-01	9.736e+00	5.835e+00
	P-value	-	>	>
	Min	8.298e-04	1.946e-02	7.827e-02
*keijzer*5	Median	**1.604e-03**	1.928e-01	2.076e-01
	Max	1.042e-02	2.236e+08	2.327e-01
	P-value	-	>	>
	Min	5.490e-16	9.958e-01	9.958e-01
*keijzer*6	Median	**8.808e-16**	9.958e-01	9.958e-01
	Max	8.030e-14	9.958e-01	9.958e-01
	P-value	-	>	>
	Min	3.888e-04	7.352e-02	1.063e-01
*keijzer*7	Median	**1.272e-02**	2.040e-01	1.711e-01
	Max	4.491e+02	9.995e+09	5.502e+08
	P-value	-	>	>
	Min	1.232e-15	0.0	0.0
*keijzer*8	Median	6.077e-15	**0.0**	**0.0**
	Max	1.628e+00	0.0	0.0
	P-value	-	<	<
	Min	4.739e-03	1.452e-01	1.393e-01
*keijzer*9	Median	**1.026e-01**	3.650e-01	1.513e-01
	Max	3.823e+09	9.995e+08	Inf
	P-value	-	>	>
	Min	2.000e-03	3.179e-02	2.375e-01
*keijzer*10	Median	**5.642e-02**	3.492e-01	3.480e-01
	Max	1.496e+01	3.570e-01	3.574e-01
	P-value	-	>	>
	Min	8.528e-01	6.091e-01	5.943e-01
*keijzer*11	Median	1.223e+13	**6.425e-01**	6.749e-01
	Max	8.597e+41	4.624e+26	1.016e+13
	P-value	-	<	<
	Min	7.485e-15	1.024e+00	1.123e+00
*keijzer*12	Median	**1.274e-01**	1.342e+01	4.010e+00
	Max	2.037e+13	7.184e+42	1.895e+01
	P-value	-	>	>
	Min	9.516e-01	1.985e+00	1.834e+00
*keijzer*13	Median	4.634e+12	8.184e+00	**4.693e+00**
	Max	1.319e+43	5.658e+42	1.572e+42
	P-value	-	<	<
	Min	1.269e-01	9.317e-01	1.064e+00
*keijzer*14	Median	1.455e+13	2.094e+01	**1.242e+00**
	Max	2.578e+41	1.803e+28	4.805e+13
	P-value	-	<	<
	Min	1.084e-15	8.889e-01	1.409e+00
*keijzer*15	Median	**1.481e-02**	3.559e+00	3.098e+00
	Max	5.055e+13	5.868e+13	9.610e+13
	P-value	-	>	>

The results of KP, shown in Table 7, are averaged over 100 trials. One can observe that the maximum average error is lower than 0.01 for all problems, the raw fitness (sum of the absolute errors for all fitness cases) is low considering the number of points (see Table 4), RMSE for the training is also very low, and RMSE for testing is not too high. Therefore, one can conclude that the fitted curve is close to the real curve.

The other important analyses are the number of objective function and node evaluations. For functions *nguyen*1-8, the average number of function evaluations was lower than 100. That number corresponds to less than 12 generations (starting with 8 experts, but not all ideas may be included in the final solution) to find a model with the desired accuracy. The number of node evaluations is also small, which means that the ideas generated to compose the final solutions used few functions and, more importantly, the mechanism the KP uses to guide the search was extremely effective.

Just for the sake of comparison, the results shown in [5] used the limit of 15×10^6 node evaluations as stopping criterion to terminate the methods. That number is four to five orders of magnitude higher than the average required by KP to achieve the expected solution quality. In fact, this number is ever higher when one analyses the number of successful runs shown in Table 8. Not only KP required a substantially smaller number of function evaluations to discover high-quality models, but also did this for all runs and all problems, easily outperforming all methods used in the comparison.

Another important issue is the comparison against ensembles of trees [14]. Ensembles usually rely on the model performance, therefore, the best models are selected to generate a fused prediction. That means that all models must have a high-quality prediction and thus similar behavior. That also means that one can not expect that the models complement each other, that is, that a model is evolved to fill the gaps left by the other models. That behavior is exactly the opposite of what KP does. One of KP's strengths is that it fuses high-quality models (for example, $f(x) = x^4$ and $f(x) = x^3 + x^2$) with poor quality models (for instance, $f(x) = x$) to compose the correct model ($f(x) = x^4 + x^3 + x^2 + x$).

6. SUMMARY AND CONCLUSIONS

Kaizen Programming (KP) proposed in this work is an evolutionary algorithm that uses a collaborative problem solving approach in which partial solutions are joined to result in a complete solution. The partial solutions are created by the experts, that generate ideas based on knowledge obtained from the problem during the improvement process. The knowledge increases because the ideas have their contribution to the problem measured when a complete solution is evaluated. Thus, one can know, exactly, which ideas are useful for the next improvement cycle. With this approach, the guessing, and consequently the bloat, are decreased when compared to Genetic Programming (GP), making KP less art and more science.

The search used by KP showed effective, but the creation of new ideas is still random, which limits KP's performance. Besides that, KP was significantly superior in the tested benchmark functions when compared to the results of GP and other methods.

Given that KP returns a solution in the form of a linear regression model, it can be easier to interpret than a solution found by GP. One can separately investigate the behavior of each component of the solution because it is an additive function. Moreover, the model is based on strong mathematical and statistical foundations, it is largely described and employed in the literature and also to solve real world problems. The linear model allows the posterior use of several other statistical tools such as model selection (AIC, BIC, among other), calculate prediction and confidence intervals, perform residual analysis, among others. Therefore, KP is a promising tool to be used in computer science, statistics, engineering, and any other field of study that make use of linear models.

Table 7: Results of KP (averaged of 100 trials) using *Nguyen* benchmark functions as in [5]. The raw fitness is the sum of absolute error on all fitness cases.

Problem	Maximum Error	Raw Fitness	RMSE Training	Func. Eval.	Node Eval.	RMSE Testing	Succ. Runs
nguyen1	0.00168	0.01256	0.00075	62.72	148.98	0.11654	100
nguyen2	0.00256	0.02013	0.00117	71.44	183.57	0.01779	100
nguyen3	0.00268	0.01992	0.00119	78.80	216.17	0.01753	100
nguyen4	0.00284	0.02211	0.00130	90.00	246.35	0.01988	100
nguyen5	0.00211	0.01613	0.00095	66.80	160.09	0.01863	100
nguyen6	0.00311	0.02356	0.00139	75.44	190.91	0.10314	100
nguyen7	0.00138	0.01037	0.00062	59.04	146.23	0.98138	100
nguyen8	0.00301	0.02075	0.00126	76.56	199.86	0.29934	100
nguyen9	0.00777	0.17801	0.00235	1,792.27	10,682.74	0.00437	100
nguyen10	0.00588	0.04586	0.00271	1,178.51	5,461.95	0.03762	100

Table 8: Number of successful runs using *Nguyen* benchmark functions (F1-F10) as in [5].

Method	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
KP	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
ABCP	89	50	22	12	57	87	58	37	33	21
SC	48	22	7	4	20	35	35	16	7	18
NSM	48	16	4	4	19	36	40	28	4	17
SAC2	53	25	7	4	17	32	25	13	4	4
SAC3	56	19	6	2	21	23	25	12	3	8
SAC4	53	17	11	1	20	23	29	14	3	8
SAC5	53	17	11	1	19	27	30	12	3	8
CAC1	34	19	7	7	12	22	25	9	1	15
CAC2	34	20	7	7	13	23	25	9	2	16
CAC4	35	22	7	8	12	22	26	10	3	16
SBS31	43	15	9	6	31	28	31	17	13	33
SBS32	42	26	7	8	36	27	44	30	17	27
SBS34	51	21	10	9	34	33	46	25	26	33
SBS41	41	22	9	5	31	34	38	25	19	33
SBS42	50	22	17	10	41	32	51	24	24	33
SBS44	40	25	16	9	35	43	42	28	33	34
SSC8	66	28	22	10	48	56	59	21	25	47
SSC12	67	33	14	12	47	47	66	38	37	51
SSC16	55	39	20	11	46	44	67	29	30	59
SSC20	58	27	10	9	52	48	63	26	39	51

Acknowledgments

We thank the anonymous reviewers for their careful reading of this manuscript and their many insightful comments and suggestions. This paper was supported by CNPq (Universal) grant (486950/2013-1).

7. REFERENCES

[1] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012.

[2] H. Gitlow, S. Gitlow, A. Oppenheim, and R. Oppenheim. *Tools and Methods for the Improvement of Quality*. Irwin series in quantitative analysis for business. Taylor & Francis, 1989.

[3] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[4] M. Imai. *Kaizen (Ky'zen), the key to Japan's competitive success*. McGraw-Hill, 1986.

[5] D. Karaboga, C. Ozturk, N. Karaboga, and B. Gorkemli. Artificial bee colony programming for symbolic regression. *Information Sciences*, 209(0):1 – 15, 2012.

[6] M. F. Korns. Abstract expression grammar symbolic regression. In *Genetic Programming Theory and Practice VIII*, volume 8 of *Genetic and Evolutionary Computation*, chapter 7, pages 109–128. Springer, Ann Arbor, USA, 20-22 May 2010.

[7] M. F. Korns. Abstract expression grammar symbolic regression. In *Genetic Programming Theory and Practice VIII*, volume 8 of *Genetic and Evolutionary Computation*, pages 109–128. Springer New York, 2011.

[8] J. R. Koza. *Genetic programming - on the programming of computers by means of natural selection*. Complex adaptive systems. MIT Press, 1993.

[9] J. McDermott, D. R. White, S. Luke, L. Manzoni, M. Castelli, L. Vanneschi, W. Jaskowski, K. Krawiec, R. Harper, K. A. D. Jong, and U.-M. O'Reilly. Genetic programming needs better benchmarks. In T. Soule and J. H. Moore, editors, *GECCO*, pages 791–798. ACM, 2012.

[10] A. Moraglio, K. Krawiec, and C. G. Johnson. Geometric semantic genetic programming. In C. A. C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, editors, *PPSN (1)*, volume 7491 of *Lecture Notes in Computer Science*, pages 21–31. Springer, 2012.

[11] M. O'Neill, L. Vanneschi, S. Gustafson, and W. Banzhaf. Open issues in genetic programming. *Genetic Programming and Evolvable Machines*, 11(3-4):339–363, 2010.

[12] C. L. Pennachin, M. Looks, and J. A. de Vasconcelos. Robust symbolic regression with affine arithmetic. In *GECCO '10: Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 917–924, Portland, Oregon, USA, 7-11 July 2010. ACM.

[13] R. Poli, W. Langdon, and N. McPhee. *A field guide to genetic programming*. Lulu Enterprises Uk Ltd, 2008.

[14] K. Veeramachaneni, O. Derby, D. Sherry, and U.-M. O'Reilly. Learning regression ensembles with genetic programming at scale. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, pages 1117–1124, New York, NY, USA, 2013. ACM.