G Model
ASOC 3777 1–15
ARTICLE IN PRESS

Applied Soft Computing xxx (2016) xxx–xxx

Contents lists available at ScienceDirect

Applied Soft Computing

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

Design and analysis of evolutionary bit-length optimization algorithms for floating to fixed-point conversion

Q1 L.S. Rosa*, A.C.B. Delbem, C.F.M. Toledo, V. Bonato

The Institute of Mathematics Sciences and Computation, The University of São Paulo, Brazil

ARTICLE INFO

Article history:
Received 14 October 2015
Received in revised form 11 August 2016
Accepted 19 August 2016
Available online xxx

Keywords:
Fixed-point
Floating-point
Evolutionary genetic algorithms

ABSTRACT

Hardware designs need to obey constraints of resource utilization, minimum clock frequency, power consumption, computation precision and data range, which are all affected by the data type representation. Floating and fixed-point representations are the most common data types to work with real numbers where arithmetic hardware units for fixed-point format can improve performance and reduce energy consumption when compared to floating point solution. However, the right bit-lengths estimation for fixed-point is a time-consuming task since it is a combinatorial optimization problem of minimizing the accumulative arithmetic computation error. This work proposes two evolutionary approaches to accelerate the process of converting algorithms from floating to fixed-point format. The first is based on a classic evolutionary algorithm and the second one introduces a compact genetic algorithm, with theoretical evidence that a near-optimal performance, to find a solution, has been reached. To validate the proposed approaches, they are applied to three computing intensive algorithms from the mobile robotic scenario, where data error accumulated during execution is influenced by sensor noise and navigation environment characteristics. The proposed compact genetic algorithm accelerates the conversion process up to 10.2× against the state of art methods reaching similar bit precision and robustness.

1. Introduction

Q3

Hardware and software optimizations are crucial for embedded systems customized for specific applications. The optimization of these components can improve system performance and energy consumption. Based on the application behavior, a designer can exploit several optimizations to avoid an unnecessary or inappropriate use of hardware resources. For instance, scratchpad memory may be preferable instead of traditional cache memory in order to improve the energy efficiency of a system [1].

Optimizations related to arithmetic operations play a central role in a customization process, especially for embedded computing systems, which are highly sensitive to energy consumption and hardware cost. Important project decisions can be made only by knowing how many bits are necessary for their representations. For instance, this can enable a designer to choose whether it is necessary to have a dedicated arithmetic hardware unit as well as to determine the operations to be implemented on it. All these aspects are important to decide the hardware technology to be used. The authors in [2] present a survey evaluating hardware implementations for several applications.

The present paper introduces a multi-objective compact genetic algorithm (mo-cGA) based on a previous evolutionary algorithm proposed in [3] and on the compact genetic algorithm (cGA) [4]. This approach is applied to estimate bit-lengths for variables with real domain in algorithms according to a maximum error defined by the user.

The method is validated using classical algorithms for mobile robotics, where optimizations regarding performance, power consumption and size are important. The case study is reported over EKF-SLAM [5], Particle Filter (PF) [6] and the Gauss–Jordan Matrix Inversion (MI) [7] algorithms. In this context, the main contributions of this paper are:

- A mo-cGA applied to the bit-lengths estimation problem;
- A practical solution to accelerate the computationally heavy process of defining fixed-point arithmetic parameters, mitigating the whole procedure of design space exploration in hardware design;
- Theoretical evidence that the proposed mo-cGA has reached a near optimal performance, reducing the algorithm size impact

Q2 * Corresponding author at: The Institute of Mathematics Sciences and Computation, Avenida Trabalhador são-carlense, 400, 13566-590 São Carlos, São Paulo, Brazil.
E-mail addresses: leandrors@usp.br (L.S. Rosa), acbd@icmc.usp.br (A.C.B. Delbem), claudio@icmc.usp.br (C.F.M. Toledo), vbonato@icmc.usp.br (V. Bonato).

http://dx.doi.org/10.1016/j.asoc.2016.08.035
1568-4946/

during the conversion process and accelerating up to 10.2×, when compared with the state of art methods of floating to fixed-point conversion, without compromising the bit precision and robustness.

The paper is organized as follows. Section 2 reviews related works with the floating to fixed-point conversion algorithm. Section 3 presents the bit-lengths estimation problem during the floating to fixed-point conversion of an algorithm. Section 4 presents the heuristic approach to the bit-lengths estimation problem. Section 5 presents a classical evolutionary approach, previously applied to the bit-lengths estimation problem, and introduces mo-cGA for the same problem. Section 6 presents a performance study comparing the proposed methods. Finally, Section 7 concludes the paper.

2. Related work

The conversion of an algorithm from floating to fixed-point demands the estimation of bit-lengths for each single variable. The aim is to find the smallest lengths that do not violate the maximum error defined by the user.

Recent works already use fixed-point representation to implement the Extended Kalman Filter (EKF) algorithm to solve the Simultaneous Localization and Mapping (SLAM) problem. The authors in [8] present a fixed-point implementation, which uses a constant bit-length for all variables. Another approach using fixed-point for SLAM is described in [9], where some variables have the bit-lengths defined according to physical constraints of a robot, and the remaining EKF-SLAM variables are left without optimization. These solutions apply fixed-point approach to reduce the computational cost of the whole system. However, further improvements could be achieved if the bit-lengths of each variable are properly defined following a given error.

Methods of floating to fixed-point conversion can be divided into two main classes: formal and non-formal methods. Formal methods are methods that, given the algorithm input range and a maximum acceptable error, give a solution (ranges or bit-lengths) that are mathematically proven to respect the maximum acceptable error as long as the input range is within the input range given to the method. Note that, in order to prove the ranges, a formal method might restrain the algorithm of having some structures, which are generally non-affine loops or unpredictable branches. On the other hand, non-formal methods cannot guarantee the maximum acceptable error obedience, but they generally do not imply constraints on the algorithm to be converted. It is worthy to note that these definitions do not imply optimality.

Approaches, orientated to Digital Signal Processors (DSP) applications, were proposed to convert from floating-point to fixed-point format [10–15] focusing DSP applications. These approaches are not applicable to algorithms with unpredictable feedbacks (e.g. `while` loops with stop condition statically indeterminable), leaving an open gap related to the types of algorithms that can be converted.

Formal method approaches for fixed to floating-point conversion, such as *Interval Arithmetic (IA)*, *Affine Arithmetic (AA)*, and *Symbolic Methods*, are also oriented to DPS applications. These methods present performance decay when applied on strongly non-affine computations, which is a problem mitigated by *Satisfiability-modulo Theory (SMT)* based methods [16,17]. Kinsman and Nicolici [16] present an SMT-based solution which allows estimating fixed or floating-point custom bit lengths given an error for DSP applications. [17] extends [16] to apply SMT on iterative computations (a.k.a. `for` loops) based on representing the error as *error* = (*knee*, *slope*) instead of the general error magnitude, what mitigates "Catastrophic Cancellation" problems and is more robust than the previously cited methods.

As reported in [17], the capabilities of applying the method to iterative computations is restricted to the solver capabilities of solving the equation systems for the errors and precisions, which are limited. It is worth to note that in [17] all cases of study have its iteration spaces bounded by the user, based on mathematical formulations, what can not be generically applied, especially if we consider algorithm with unpredictable feedbacks, which are common in the autonomous robotics fields.

Boland and Constantinides [18] present a *Polynomial Algebraic Approach (PAA)*, which represents the computations as polynomials of the δ, suchas $|\delta| \leq \Delta = 2 - m$, and m is the mantissa size of a floating point representation. Then, the equations pass through a heuristic to define the bit-sizes. Furthermore, the Polynomial Algebraic Approach presents a promising scalability that is not presented in the SMT solvers [18,19].

Boland and Constantinides [19] present a detailed analysis of the IA, AA, *Polynomial Algebraic Approach using Handelman representations (Handelman)* [20] and *Taylor methods with Interval Remainder bounds (TwIR)* [21] showing that these approaches scalability fades quickly when applied to large algorithm. Further then, Boland and Constantinides [19] present a scalable approach to the bit estimation problem, which represents the source code operations by a pair of different polynomials, gathering the IA and Handelman approaches in order to balance the Handelman complexity (NP-Hard) with the IA complexity (linear), and also balancing the IA loose solutions (too many bits) with the Handelman solutions tightness.

The approaches presented in [18,19] calculate bit lengths for floating point representations given an algorithm, which is different from our floating to fixed-point conversion problem. Furthermore, [18] is not applicable to feedback computations, while [19] handles statically bounded iterative computations (`for` loops with bounds defined at compilation time) by unrolling the loops. Thus, these approaches cannot be applied in our scenario.

Sarbishei et al. [22] present an algorithm to estimate fixed-point bit lengths for an input algorithm with unpredictable feedbacks. This approach targets infinite impulse response filters, supposing that the application is Bounded-Input–Bounded-Output and that there is a user given parameter W which is greater than the filter order. Note that these two suppositions are not true in our scope, making this approach not applicable as well.

A fundamental limitation of these formal approaches is that they can only handle data flows which can be converted to static single assignment (SSA) form. In other words, they cannot handle algorithms which the branch conditions can depend on data values and loops with iteration space dynamically defined [23]. Boland and Constantinides [23] present an approach to contour these limitations based on substituting the stop conditions by a ranking function, aiming to estimate floating-point mantissa and exponent lengths. Even though, there are no scalable techniques to find such functions if the loop body contains non-linear functions, which is present in most of the autonomous robotic algorithms. If [23] tool fails in its attempt to find a ranking function, the user will be inquired for one.

Extensions of bit-lengths estimation for algorithms with unpredictable feedbacks are presented in [24]. The authors in [24] extend [15] to handle unpredictable feedbacks based on training sets. However, the proposed methods are computer-intensive and time-consuming for complex algorithms. The authors in [3] introduce improvements over [24] with an evolutionary algorithm (EA), reducing both conversion time and bit lengths.

In the present paper, a complete analysis is carried out on this previous EA. We also introduce a mo-cGA to solve this problem, which is based on an estimation of distribution algorithm proposed

i	1	2	3	...	n
m	m_1	m_2	m_3	...	m_n
p	p_1	p_2	p_3	...	p_n

Fig. 1. An example of a candidate solution for the bit-lengths estimation problem.

by Harik et al. [4]. This method uses a probability distribution to represent a population (solutions) avoiding the necessity of storing a large population. The compact genetic algorithm (cGA) has performance equivalent to the simple genetic algorithm (GA) with uniform crossover [4].

The motivation for this work comes from mobile robotic field research which indicates hardware customization as a feasible way to achieve system requirements. For instance, the authors in [25] propose an optimized processing system, where an EKF-SLAM floating point based system was implemented on FPGA. Furthermore, the Particle Filter (PF) is a spread option to handle mobile robot localization [6,26] and hardware implementations of the PF can explore parallelism and pipeline. This processing system can take advantage of the fixed-point arithmetic representation [27] instead of processors with floating-point implementations [28,29]. Furthermore, as motivation to the use of evolutionary algorithms, we can cite [30], where the authors define variants from cGA to evolvable hardware applications, where a superior performance is reached for static and dynamic optimizations applied to standard and developed benchmarks. A microcontroller is optimized in [31] using cGA with real-coded implementation and its method was competitive against a standard cGA and other population-based algorithms. A hybrid approach combining cGA and mathematical programming techniques is proposed in [32], where the hybrid cGA is applied to solve a multi-level lot sizing problem.

3. Conversion from floating to fixed-point problem

The floating to fixed-point conversion problem consists in finding n pairs (m_i, p_i) with $i = 1, \ldots, n$, where m_i is the integer and p_i the fractional length of the ith variable of the algorithm to be converted. A set of n pairs (m_i, p_i) is “a candidate solution” for the bit-lengths estimation problem, which is shown in Fig. 1. For the rest of this paper, the “candidate solution” will be considered as two separate parts, the m_i and the p_i values, which are calculated in different moments, as described in Section 4. Being so, the genetic algorithms presented in this paper consider as solution the p_i values only.

Each solution has an associated error measure, defined by Eq. (1). It measures the difference between the result of the floating-point and fixed-point executions over a training set β, where $outdata_{float}$ and $outdata_{fixed}$ are the output of the floating and fixed-point versions, respectively.

$$error = avg_{\beta}\left\{\frac{norm(outdata_{float} - outdata_{fixed})}{norm(outdata_{float})}\right\} \tag{1}$$

4. Conversion algorithm

The conversion algorithm described in this Section was first proposed by Roy and Banerjee [15] and extended by de Souza Rosa and Bonato [24]. The algorithm is divided into eight steps summarized as follows (details about these steps are explained in [24]):

Step 1: *Levelization* divides multi-operations assignments into single operation assignments, by e.g. $a = b \times c \times d$; is divided into $t = b \times c$ and $a = t \times d$.

Step 2: *Scalarization* changes all vectorized operations into scalar operations by rewriting them using `for` loops.

Step 3: *Computation of Ranges for Variables* estimates the maximum values of each algorithm variable, using a training set β composed by n_β different random EKF-SLAM executions.

Step 4: *Evaluate Integer Variables* identifies the source code variables that are integers, saving a considerable amount of computation time during **Step 8**.

Step 5: *Generation of a Fixed-point (Matlab) Code* converts the source code to fixed-point representation.

Step 6: *Fixed Integral Range* evaluates the integer range for the fixed point representation of each variable. In other words, this Step calculates the m_i values for all n variables of the algorithm to be converted.

Step 7: *Coarse Optimize* uses a binary search, over the solution *error*, to estimate one single value of bit-length for the fractional part for all variables of the algorithm. Based on the fact that the larger the bit-length is, the smaller will be *error*, we define this unique value as p_c, representing the maximum value for all the p_i values respective to the n variables.

Step 8: *Fine Optimize* (FO) makes a variable-level optimization over the *Coarse Optimize* results using a two-phase heuristic. The first phase reduces a bit from each variable independently and calculates *error* due to each reduction (n times), then it chooses the smaller *error* and sets the respective variable to have a 1 bit reduction in its p_i; this phase is repeated while $error < E_{max}$. The second phase increases a bit in each variable and calculates new values of *error* due to each increment (n times), then it selects the variable respective to the bigger error reduction and increment it one bit, moreover, the second phase selects the variable with smaller error reduction and decrements two bits from it; this phase is repeated until $error > E_{max}$. Note that, to reduce a single bit, this heuristic calculates *error* n times.

For an adequate estimation, the training set β should contain all expected range of values and combinations. Even though this is not always possible, a large enough set should be adequate. Thus, the *error* calculation is the bottleneck of the conversion algorithm, since it is evaluated several times in the *Fine Optimize* (FO), **Step 8**.

The FO step is the most time-consuming, so a heuristic procedure was proposed in [3] called `Evolutionary Optimize` (EO). In this work, we propose a `multi-objective Compact Genetic Algorithms` (mo-cGA), that will be named *mo-cGA Optimize* (mo-cGAO). It is worthy to notice that, since the FO is a heuristic, there is no guarantee that the results will be a global optimum.

The integer bits lengths are calculated based on the maximum values of the *Computation of Ranges for Variables*, **Step 3**, which executes the algorithm to be converted over the training set only once. The *Coarse Optimize* reduces the search space for the FO, EO and, mo-cGAO applying a binary search, when the algorithm to be converted is executed over the training set $log_2(p_{max})$ times, where p_{max} is the maximum desirable value for the precision bit-lengths. That means that p_{max} is an initial upper bound defined by the user, which is refined by the *Coarse Optimize*. To set $p_{max} = 32$ or $p_{max} = 64$ is usually more than enough.

5. Methods

This section will describe the EO (Section 5.2) and mo-cGAO (Section 5.3) algorithms, but firstly, common aspects as encoding, fitness function, and parameters are explained (Section 5.1).

5.1. Fitness function and fixed parameters

Since the m_i values are calculated in **Step 6** (*Fixed Integral Range*) of the conversion algorithm (Section 4), the FO, EO and mo-cGAO

Please cite this article in press as: L.S. Rosa, et al., Design and analysis of evolutionary bit-length optimization algorithms for floating to fixed-point conversion, Appl. Soft Comput. J. (2016), http://dx.doi.org/10.1016/j.asoc.2016.08.035

just have to calculate the p_i values. We define p as a vector of size $p_c + 1$, with $p_i \in [0, p_c]$. Vector p is called chromosome in both the EO and mo-cGAO, and p_i values are called alleles. A maximum fractional length p_c, for all non-integer variables, is calculated in **Step 7** (*Coarse Optimize*).

The solutions must satisfy the condition $error - E_{max} \leq 0$, where *error* is the error associated with the solution represented by vector p, calculated by Eq. (1), and E_{max} is the maximum acceptable error.

Moreover, EO and mo-cGAO have to minimize all the values in vector p. Thus, we define $p_t = \sum_{i=1}^{p_c+1} p_i$ to measure the quality of a solution related to its number of bits.

Finding a solution to reduce *error* and p_t simultaneously, is a multi-objective problem. There are optimization approaches that reduce the problem to a single objective by applying a penalty/reward to the error. Eq. (2) shows a single objective function that penalizes relatively large errors, where *Severity* is a positive constant.

$$Fitness = p_t + Severity \times p_c \times 100 \times (erro - E_{\max}) \tag{2}$$

It is worth to notice that the error propagation between the variables make us expect that the solutions will have its p_i values close to the maximum p_c, except by the ones representing integer variables. In other words, several variables will have their bit-lengths reduced of few bits, instead of few variables having a relatively large number of bits reduced.

Fig. 2 illustrates the effects of error propagation. If 2 bits are reduced from the same variable *A*, all the fractional information is lost, while if 2 bits are reduced, one from *A* and another from *B*, only part of the fractional is lost, implying in a smaller error.

We decided to choose this transformation from a multi-objective to a single-objective function (here called fitness) to focus on the bit-reduction objective since the error restriction can be mathematically contoured by several methods as presented in [33].

For example, in [34] an error contour method is presented focusing the EKF-SLAM. In this paper, it is presented two symptoms of errors, and methods to handle them, that lead to filter inconsistencies.

5.2. Evolutionary Optimize (EO)

The EO steps, as proposed in [3], are summarized below and detailed in the sequel.

Step 1: Initialize population.
Step 2: Generate offspring.
Step 3: Evaluate and Select offspring.
Step 4: Repeat **Step 2** until the stop criteria is reached.

Step 1 initializes 100 individuals, which is an adequate value according to empirical tests [3], with alleles randomly generated from the exponential distribution defined in Eq. (3), where

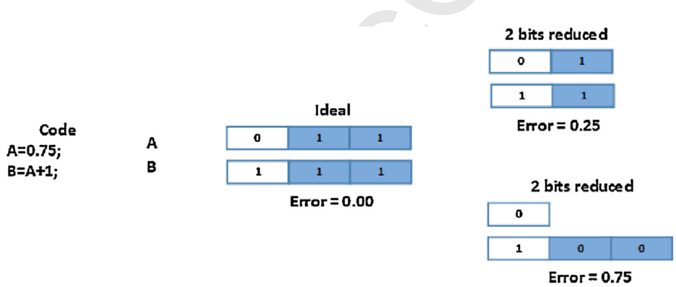

Fig. 2. An example of error propagation when reducing two bits in different ways between code variables. The integer part is represented in white background and the fractional part in darker background.

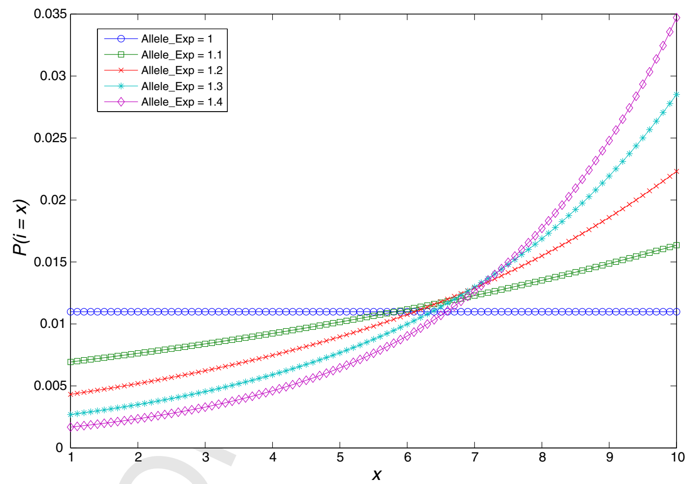

Fig. 3. Probability (Eq. (3)) of choosing *x* bits for the alleles values, parameterized by *Allele_Exp*.

$base = Allele\_Exp$ and $x_{max} = p_c$. Letting *Allele_Exp* being a parameter in the EO

$$Prob(x) = \frac{base^x}{\sum_{k=1}^{x_{\max}} base^k} \tag{3}$$

Fig. 3 shows the probability of a value x be chosen as the allele, given a value for *Allele_Exp* (based on Eq. (3)). Note that values near p_c are more probable of being chosen.

After each generation, the fitness is calculated for each individual and the population set is sorted by its value in decreasing order.

Step 2 exploits the EO fast convergence, where an elitist gap (μ, λ) is used [35], where $\mu = Population\_Size$ is the population size, $\lambda = Gap \times \mu$, and *Gap* is a parameter that defines a fraction of μ for the generation offspring. Eq. (4) defines the offspring size.

$$Offspring\_Size = Gap \times Population\_Size \tag{4}$$

The population is sorted, by decreasing fitness, before the offspring generation. Next, two individuals are chosen according to an exponential rank with replacement. The rank is given by the individual position in the population set, and the probability of being chosen is defined by Eq. (3), where $base = Population\_Exp$ and $x_{max} = Population\_Size$. The replacement means that individuals selected to reproduce are not removed from the population.

Eq. (3) defines that the individuals in the higher positions, that are the ones with smaller fitness according to the decreasing sorting, have more probability to be chosen to reproduction.

Two children are created applying one point crossover with a random allele position as crossover point (Fig. 4) to the chosen individual. A parameter *Mutation_Rate* defines the chance of mutation for each allele. The mutation replaces an allele by a new one accordingly to the probability distribution of Eq. (3) (Fig. 4).

Step 3 aims at a high convergence rate to reduce the number of generations and fitness calculations. We proposed an extreme elitist selection (a steady-state approach), where the new population is the *Population_Size* individuals with the best fitness values between the current population and its offspring.

Once a high convergence rate is imposed, it is expected the algorithm converges to a local minimum value before the maximum number of iterations defined by the user (which we define as *Max_Interactions*). In order to overcome this drawback, we define

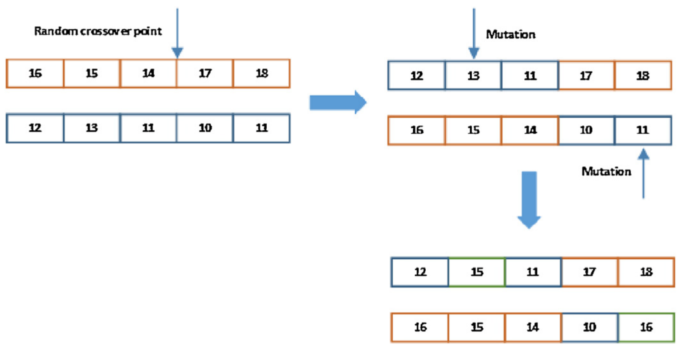

Fig. 4. Example of offspring generation given two parents. The crossover point (indicated by an arrow) is randomly chosen and each allele has a change of mutating.

another stop criteria by including the selection intensity metric defined by Eq. (5) [36].

$$Selection\_Intensity = \frac{|\overline{f_{sel}} - \overline{f}|}{\sigma_{sel}}, \quad (5)$$

where $\overline{f}$ is the average fitness value for individuals in the population before selection, $\overline{f_{sel}}$ is the same average fitness after selection, and σ_{sel} is the covariance of the fitness of the population before selection.

The selection intensity is small for the first generations since the covariance is large for the population in the first generations. Thus, we decided to add a takeover rate to the stop criteria. We limited the takeover rate to half of the population since the chance of worse individuals to reproduce is very small.

The takeover rate is calculated only for the last few generations by evaluating the average covariance of alleles for all individuals in the "best half" of the population after selection as shown in Eq. (6). The number of generations which the takeover rate will be calculated is defined by the user in a parameter that we call *Stop_Variation*. We can increase the likelihood to *Takeove_Rate* be a small number by reducing *Stop_Variation*, that means, if half of the population changes slightly in the last *Stop_Variation* generations, *Stop_Variation* will be a small value. In the other hand, if *Stop_Variation* is large, the population will certainly have changed, since the first generations are composed by a "random" population and the newer ones are composed by selected individuals.

$$Takeove\_Rate_{g=j-k} = avg^{g}(avg^{v}(cov^{Pop/2}(p_i)))$$
$$\text{where} \begin{cases} i = 1 \ldots n \\ cov^{Pop/2}(x) \text{covariance of } x \text{ in half of the population} \\ avg^{v}(x) \text{average of } x \text{ in the } n \text{ variables of the algorithm} \\ avg^{g}(x) \text{average of } x \text{ in the last } Stop\_Variation \text{generations} \end{cases} \quad (6)$$

The average of the selection intensity and the takeover rate, for 100 executions of the EO, is presented in Figs. 5 and 6 respectively. It is shown that the selection intensity is small for the first interactions and it is maintained constant due the mutation rate. Note that without mutation, after convergence, the variation in the population would be zero, making the selection intensity (Eq. (5)) goes to infinity.

Thus, according to Fig. 5, we conclude that the convergence occurs with approximately 20 generations. These values can be set as a minimal value for this the number of generations, so we can assume that EO has not converged for values smaller than that.

Fig. 6 shows the takeover convergence through the generations, where it suffers a sudden reduction with few generations, as expected. This happens because of the high convergence of the reproduction and selection methods, and they keep a small variation as consequence of mutation over alleles.

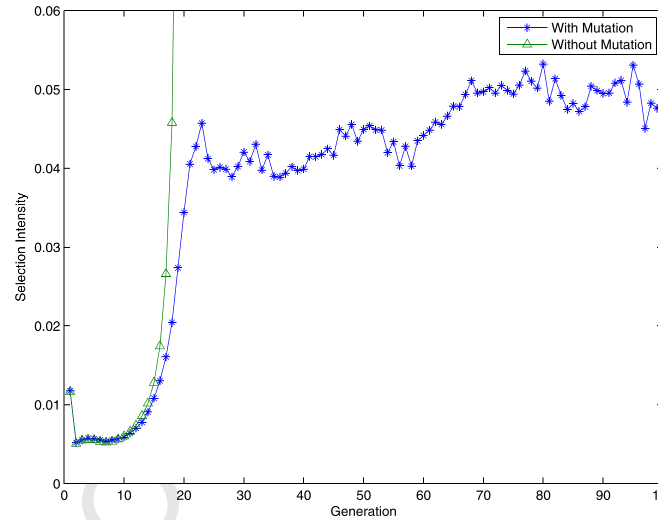

Fig. 5. Average selection intensity, in function of the generations, of a hundred trials of the *EO* algorithm without interrupting the execution by the stop criteria.

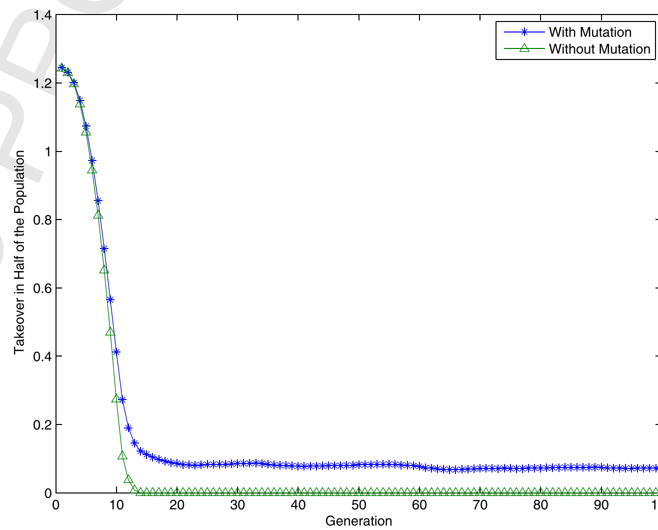

Fig. 6. Average takeover rate in half of the population, in function of the generations, of a hundred trials of the *EO* algorithm without interrupting the execution by the stop criteria.

Based on the previous results, the stop criterion was established as follows: if the best solution satisfies $error \leq E_{max}$, and $Selection\_Intensity \leq Threshold$, and $Takeove\_Rate \leq Threshold$, then stop.

Finally, we summarize next the seven input parameters.

Allele_Exp – Basis of the exponential distribution for the allele initialization and mutation value.
Severity – Reward/penalty rate for the error impact in the fitness.
Mutation_Rate – Probability of each allele be modified after offspring generation.
Population_Exp – Basis of the exponential distribution for the choice of individual to reproduction.
Stop_Variation – Percentage of the maximum number of generations to evaluate the takeover rate over half of the population.
Threshold – Upper bound value for *Takeover_Rate* and *Selection_Intensity* used in the stop criterion.
Gap – Fraction of the original population corresponding to the offspring size.

Please cite this article in press as: L.S. Rosa, et al., Design and analysis of evolutionary bit-length optimization algorithms for floating to fixed-point conversion, Appl. Soft Comput. J. (2016), http://dx.doi.org/10.1016/j.asoc.2016.08.035

In Section 6 an experimental analysis of EO parameters is presented.

5.3. Multi-objective compact genetic algorithm

The authors in [4] introduce cGA, whose basic steps are synthesized next. First, the algorithm initiates a probability vector with size n and value 0.5 on each position (allele), meaning that each allele has equal probability to be either 0 or 1. Two individuals (binary strings) are randomly generated (sampled) at each iteration based on the current probability vector. They are evaluated and the one with the best score will be used to update the probability vector. At this step, each position i of the probability vector can increase (or decrease), if the ith positions are different on both individuals. In this case, if the ith position of the best individual is 1 (0), the same position in the probability vector increases (decreases) by $\frac{1}{n}$. When all probabilities of the vector have converged, cGA stops and its values become the final solution.

cGA can reduce memory requirements and computation time once it simulates an entire population using only a probability vector and two individuals created from this vector. The present paper extrapolates the probability vector by a matrix Ψ, where each row represents a variable of the algorithm to be converted and each column represents a possible amount of bits. Ψ has n rows and $p_c + 1$ columns, where $\psi_{i,j}$ represents the probability of variable i to have j bits on its representation, $1 \le i \le n$, and $0 \le j \le p_c$.

Fig. 7b presents three example individuals and Fig. 7a presents the Ψ matrix generated from them. In this case, we have $p_c = 4$ that implies in 5 columns in Ψ. All individuals have 0 bits in variable *Var*1, implying that the related probability in Ψ is $\psi_{(1,0)} = 100\%$ (3/3). Variable *Var*2 has 66.7% of chance to have 2 bits, given the values from individuals $P1(3) = 3$, $P2(3) = 3$, and $P3(3) = 2$.

For the selection process, instead of using Eq. (2) similar to EO (Section 5.2), we propose an extension of the cGA, called mo-cGA, that deals with two objectives without weighting them. The selection process is oriented by the Dominance Strength (*DS*) defined in Eq. (7). This approach is based on the *Timmel's population based method*, which is shown to be adequate for two objectives optimization with convex well-defined Pareto-optimal front [37].

$$DS(p) = \frac{DD(\overline{p})}{1 + DT(\overline{p})} \quad (7)$$

where $\overline{p}$ is the population without the individual p. $DD(\overline{p})$ is the number of individuals that satisfies the two condition below:

- $error(p) \le error(\overline{p})$
- $p_t \le \overline{p_t}$

Moreover, $DT(\overline{p})$ is the number of individual that satisfies the two conditions below:

- $error(p) > error(\overline{p})$
- $p_t > \overline{p_t}$

Ψ	0 bits	1 bit	2 bits	3 bits	4 bits
Var1	$\frac{3}{3}$	0	0	0	0
Var2	0	0	$\frac{2}{3}$	$\frac{1}{3}$	0
Var3	0	0	$\frac{1}{3}$	$\frac{2}{3}$	0
Var4	0	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	0

(a)

Individuals	Var1	Var2	Var3	Var4
P1	0	3	3	1
P2	0	2	3	2
P3	0	2	2	3

(b)

Fig. 7. Example of Ψ matrix, for a population of three individuals and $p_c = 3$.

In other words, that all individuals in a Pareto-optimal front, defined by *error* and p_t, have the same *DS* and the same quality, since we are not defining which objective is more important.

The proposed mo-cGA Optimize (mo-cGAO) can be summarized in the following steps:

1. Initialization
   (a) Initiate all $\psi_{i,j} = 0$.
   (b) Generate an individual p according to a Poisson distribution and add 1 to $\psi_{i,j}$ corresponding to j bits in variable *Var i*.
   (c) Save p_t and *error* of p, forming an information pool about the population.
   (d) Go to **Step 1b** until complete the initial population.
   (e) Normalize Ψ by dividing all values by the population size.
   (f) Select the individual with the best *DS* using Eq. (7).
2. Offspring generation
   (a) Create 2 individuals sampled from probability matrix Ψ.
   (b) Add theirs p_t and the corresponding *error* to the pool created in **Step 1c**.
3. Selection
   (a) Choose the individual with best *DS* between the two offspring and the current best individual.
4. Update the probability matrix
   (a) For the best offspring, add its respective bits to Ψ maintaining it normalized.
   (b) Return to **Step 2** until the stop criterion is reached.

Step 1 initializes *Population_Size* individuals using a Poisson distribution, where the probability of allele p_i have the value p_k, with $i = 1, \ldots, n$ and $k = 0, \ldots, p_c$, is given in Eq. (8).

$$P(p_i = p_k) = \frac{\lambda^{p_c - k} e^{-\lambda}}{(p_c - k)!} \quad (8)$$

Thus, values closer to p_c have a larger probability of been chosen for p_i. Note that the smaller the value of λ, the closer to p_c the values will be. This implies in creating an additional parameter λ, that must be set in mo-cGAO.

After creating the initial population, *error* and p_t are calculated for each individual.

Step 2 creates two individuals, where theirs values of p_i are sampled from the probability matrix Ψ.

Since Pareto-optimal fronts need a relatively large set of individuals to be properly estimated (otherwise *DS* would often be zero or small values [38]), *error* and p_t of the two individuals in the offspring are also added to the information pool created in **Step 1**.

Step 3 since our problem is a bi-objective optimization, without weighting of objectives (as the fitness in Eq. (2)), we aim at finding a set of efficient solutions with a trade-off between the objectives. Thus, we decide by the selection to be oriented by the *DS* of an individual [37], being the individual with best *DS* among offspring the chosen one.

Step 4 adds $\frac{1}{1+p_c}$ in Ψ corresponding to the best offspring, and reduces $\frac{1}{1+p_c}$ in Ψ related to the worse individual.

We define *Stable_Variation* as the number of generation such as the DS of the best individual has not been out-bested. Then, we define the stop criterion as *Stable_Variation* $\le$ *Threshrold*.

Finally, the individual with the best DS is always returned.

6. Computational results

This section evaluates the EO and mo-cGAO tuning their parameters in order to reach an adequate convergence rate and error precision. The fine tuning is made by varying the parameters defined in Section 5 and executing both algorithms 100 times on

Table 1
Variable parameters of the EO, range they were varied to analyze the influence of each value in the performance of the EO, as well their base values.

Parameter	Range	Base
Allele _Exp	1–2	1.2
Severity	0–20	10
Mutation _Rate	0–0.02	0.001
Population _Exp	1–2	1.2
Stop _Variation	0–1	0.1
Threshold	0.05–0.25	0.2
Gap	0.1–1	0.1

the training set β. Set β is composed by a 100 execution of each algorithm to be converted.

For EKF-SLAM and PF, in each execution, the number of features ranging randomly between 100 and 200 and the map size ranging randomly between 200 and 400. For MI, in each execution, the matrix size is varied randomly between 100 and 150 and the input range is varied randomly between -2^8 and 2^8.

6.1. Analysis of the Evolutionary Optimize

The fixed parameters for EO aim to evolve few individuals leading to a reduced number of generation along with *fitness* calculation.

E _max = 1%: this value was fixed since it is a fair value for the EKF-SLAM, as presented in [24].
Max _interactions = 100: this parameter limits the number of generations, which is a reasonable value based on empirical tests.
Population _size = 100: the population size is limited to 100 individuals, which is also a reasonable value based on previous tests.

The variable parameters of the EO presented in Table 1 will be evaluated one at a time. Firstly, their values are set accordingly to their respective base values, as presented in Table 1. Then, one parameter varies for 11 values equally spaced within the ranges defined in Table 1 and, for each variation, the EO is executed 100 times. After a parameter analysis, the base value in Table 1 is updated with a more suitable value found between the 11 values tested. Finally, the next parameter is analyzed in the same way.

6.1.1. Allele_Exp

Figs. 8 and 9 show that the maximum value of generations was reached and the other stop criteria were not satisfied for small values of *Allele _Exp*. The solutions error is expected once the EO could not find solutions satisfying all stop criteria.

Fig. 10 shows that the best solutions given by the evolutionary algorithm have a small value of p_t, what explains the error greater than E_{max}.

In order to guarantee the convergence of the EO, we chose *Allele _Exp*$_{EKF-SLAM}$ = 1.3. For the PF and MI we choose, respectively, *Allele _Exp*$_{PF}$ = 1.4 and *Allele _Exp*$_{MI}$ = 1.4.

6.1.2. Severity

As described in Equations (2), the *Severity* parameter scales the error influence into the fitness evaluations. When *Severity* increases, solutions with a smaller error are expected as shown in Fig. 11.

Fig. 12 shows that, without the error influence guaranteed by *Severity*, the EO makes more efforts to find a better solution that satisfies *error* < E_{max} and exhausts the maximum number of generations (100), what results in no data for *Severity* = 0 in Fig. 11. Thus, we can say that the introduction of *Severity* forces a faster

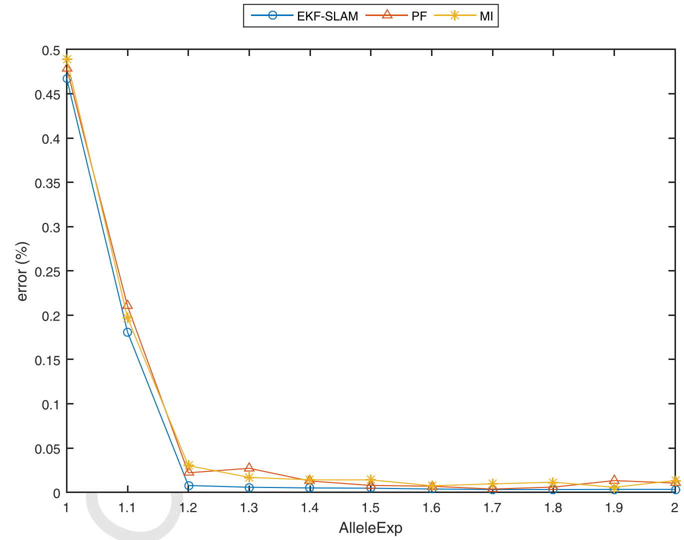

Fig. 8. Average error after a hundred executions, over the training set β, of the best solution found by the EO according to *Allele _Exp*.

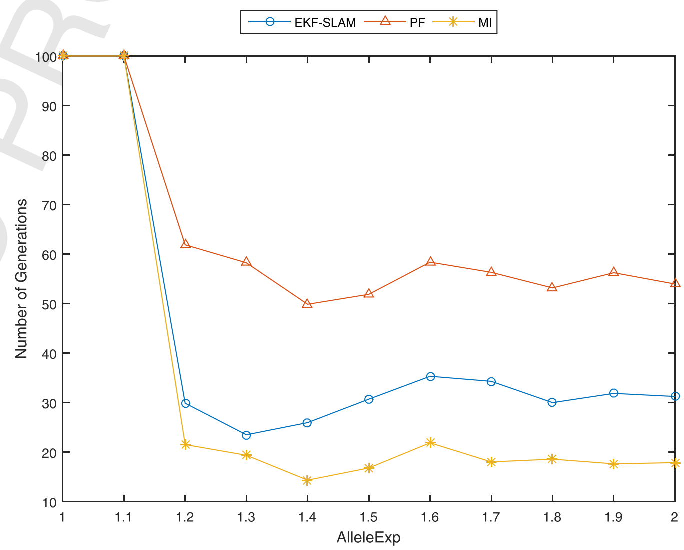

Fig. 9. Average number of generations after a hundred executions, over the training set β, of the best solution found by the EO according to *Allele _Exp*.

convergence and increasing its value the algorithm tends to reach solutions with a smaller error.

Fig. 13 shows that p_t behaviors as expected, where solutions with smaller error requires larger bit-lengths.

The priority of our approach is to reduce the number of generations, therefore, a larger p_t means only that the solution is more robust to truncation errors and demands more hardware resources. Taking this into account, it was chosen *Severity* = 4 as the final value of this parameter for the EKF-SLAM, PF, and MI algorithms.

6.1.3. Mutation_Rate

Figs. 14 and 15 indicate that larger values for *Mutation _Rate* leads to a reduced number of generations as well to small values for p_t. Thus, the choice based on the results found are *Mutation _Rate* = 0.02 for all algorithms.

6.1.4. Poppulation_Exp

If the parameter *Population _Exp* increase, more often the best individual is selected to reproduce. In the other hand, a small value for *Poppulation _Exp* makes individuals with worse fitness to be

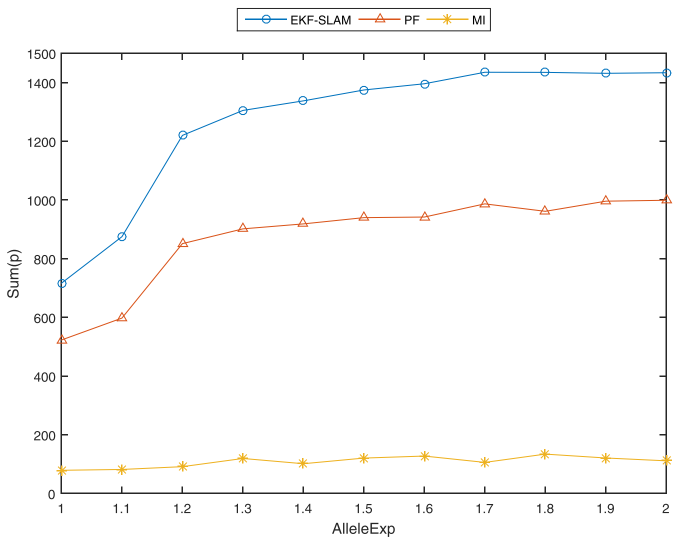

Fig. 10. Average p_t after the best solution found by the EO of a hundred executions according to *Allele_Exp*.

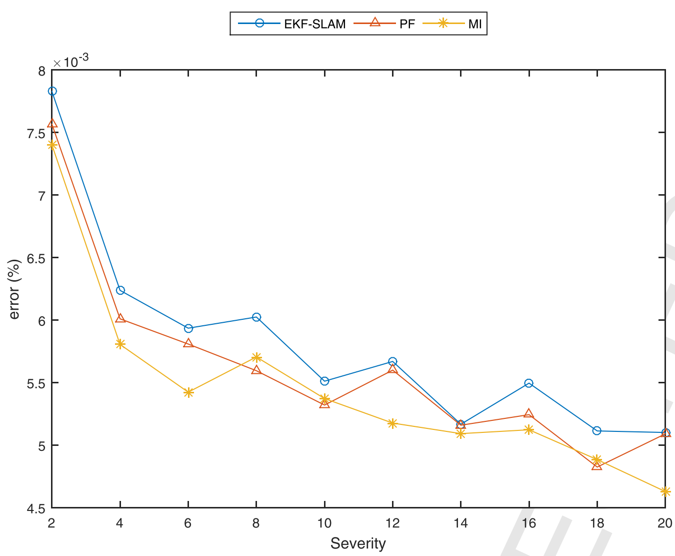

Fig. 11. Average error after a hundred executions, over the training set β, of the best solution found by the EO according to *Severity*.

chosen to reproduce, what means more time spent searching for solutions, what explains Fig. 16 behavior.

Fig. 17 shows that, even with more time spent searching for good solutions when *Population_Exp* increases, the solutions quality does not increase as well. This is explained because individuals with worse quality are chosen for reproduction, more often, keeping their alleles longer for the further generations.

We chose *Poppulation_Exp* = 2 for the three algorithms, to take advantage of fast convergence and better quality.

6.1.5. *Stop_Variation*

Fig. 18 illustrates that a small value for *Stop_Variation* can make the takeover rate over half of the population to satisfy its stop criterion within few generations. On the other hand, since the search ends quickly, the solutions given are expected to have a high value of p_t as shown in Fig. 19.

Since the number of generations is already limited by the other parameters, as indicated by the axis range in Fig. 18, we decided to choose values which will explore a better p_t. The chosen values are

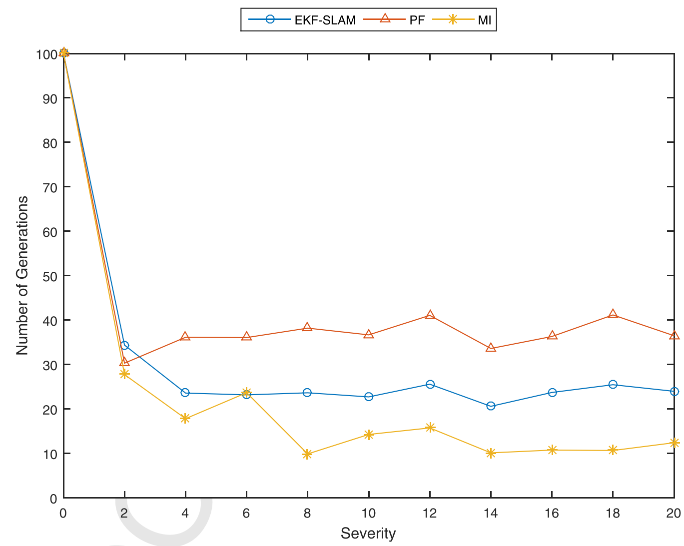

Fig. 12. Average number of generations after a hundred executions, over the training set β, of the best solution found by the EO according to *Severity*.

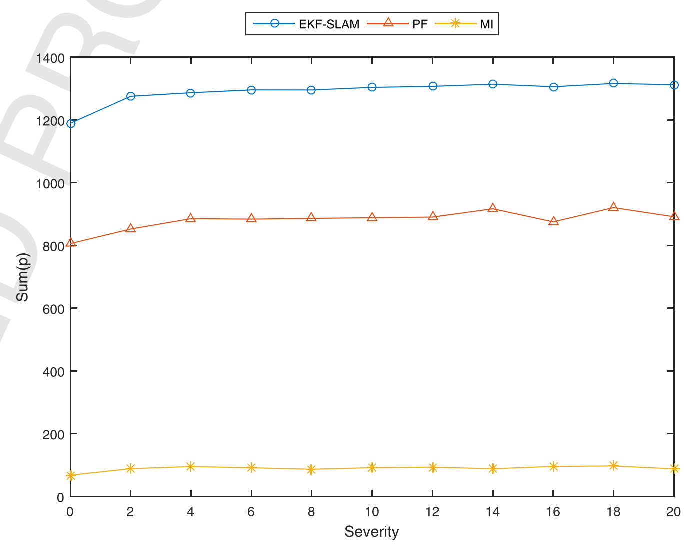

Fig. 13. Average p_t of the best solution found by the EO after a hundred executions according to *Severity*.

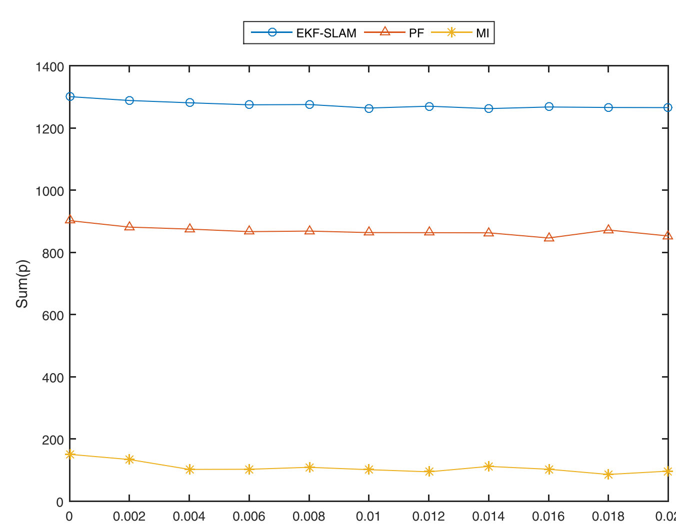

Fig. 14. Average p_t, after a hundred executions, taken by the EO to satisfy the stop criteria according to *Mutation_Rate*.

Please cite this article in press as: L.S. Rosa, et al., Design and analysis of evolutionary bit-length optimization algorithms for floating to fixed-point conversion, Appl. Soft Comput. J. (2016), http://dx.doi.org/10.1016/j.asoc.2016.08.035

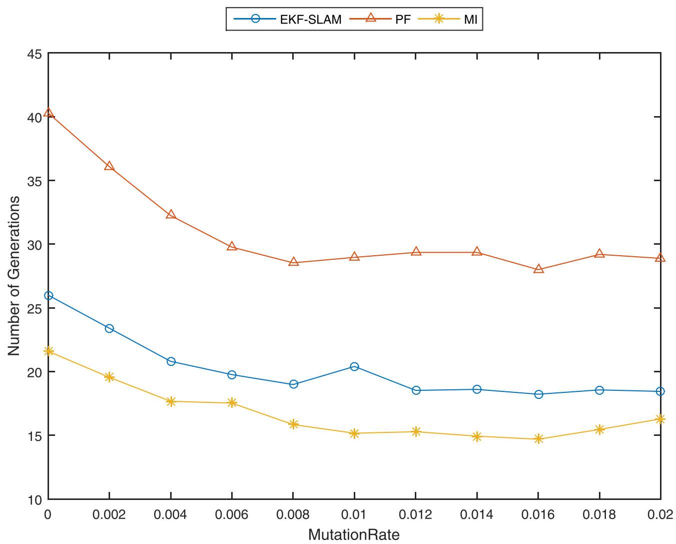

Fig. 15. Average number of generations, after a hundred executions, taken by the EO to satisfy the stop criteria according to *Mutation _Rate*.

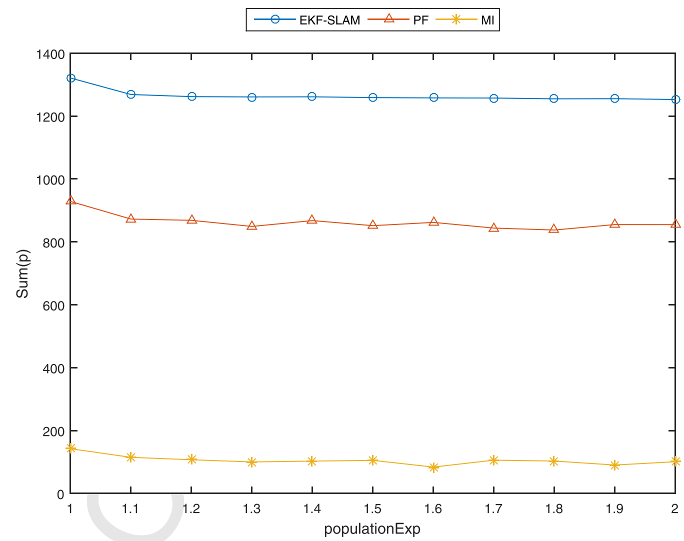

Fig. 17. Average $\sum_{i=1}^{n}$, after a hundred executions, taken by the EO to satisfy the stop criteria according to *Poppulation _Exp*.

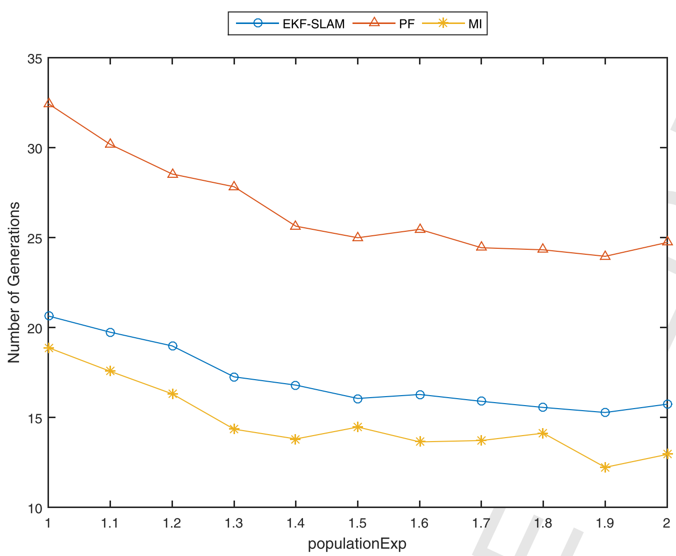

Fig. 16. Average number of generations, after a hundred executions, taken by the EO to satisfy the stop criteria according to *Poppulation _Exp*.

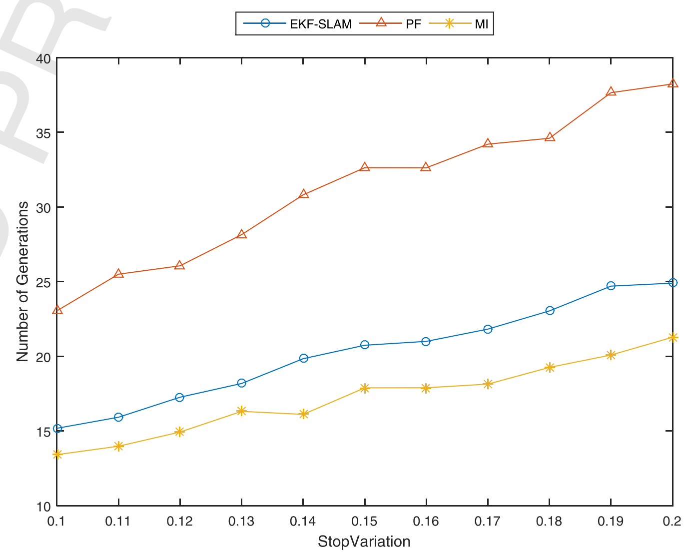

Fig. 18. Average number of generations, after a hundred executions, taken by the EO to satisfy the stop criteria according to *Stop _Variation*.

Stop _Variation = 0.2 for all algorithms, given the decreasing behavior of p_t in function of the *Stop _Variation* as shown in Fig. 19.

6.1.6. Threshold

Analogue to the *Stop _Variation* case, a larger value for the *Threshold* parameter allows the AE to satisfy its stop criteria easily, and requiring few generations, but reaching solutions with larger p_t values. These behaviors are shown in Figs. 21 and 20.

Fig. 21 shows that p_t varies in a limited range when compared with the number of variables in the algorithm, shown in Table 5. What means that only a few variables will have their number of bits reduced.

The reflection of that in hardware accelerators implementation is usually minimal since the maximum frequency of a hardware is mostly bounded by the arithmetic unit with more bits, reducing the gains of this bit reduction to a slightly smaller power consumption and hardware resources.

Thus, we focus a fast convergence of the algorithm by taking *Threshold* = 0.25 for the three algorithms.

6.1.7. Gap

Fig. 22 shows that EO converges slowly for small *Gap* values. The consequence of a large *Gap* value is a raise in the number of fitness evaluations that the method has to calculate per generation, which is the number *offspring*, as described in Eq. (4). The total amount of individuals created is given by Eq. (9).

$$\begin{aligned} Number\_of\_individuals &= Population\_Size + \cdots + Offspring\_Size \\ &\quad \times Number\_of\_Generations \end{aligned} \tag{9}$$

Fig. 23 shows the number of individuals calculated through the algorithm execution, what lead us to chose *Gap* = 0.05 as the final value of this parameter, for the three algorithms.

6.1.8. Final EO parameters values

Table 2 summarizes the final value for each parameter described from Table 1.

Please cite this article in press as: L.S. Rosa, et al., Design and analysis of evolutionary bit-length optimization algorithms for floating to fixed-point conversion, Appl. Soft Comput. J. (2016), http://dx.doi.org/10.1016/j.asoc.2016.08.035

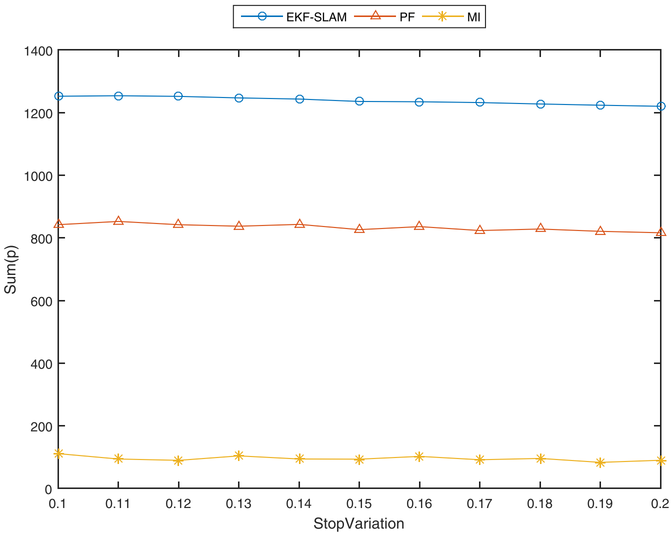

Fig. 19. Average p_t, after a hundred executions, taken by the EO to satisfy the stop criteria according to *Stop_Variation*.

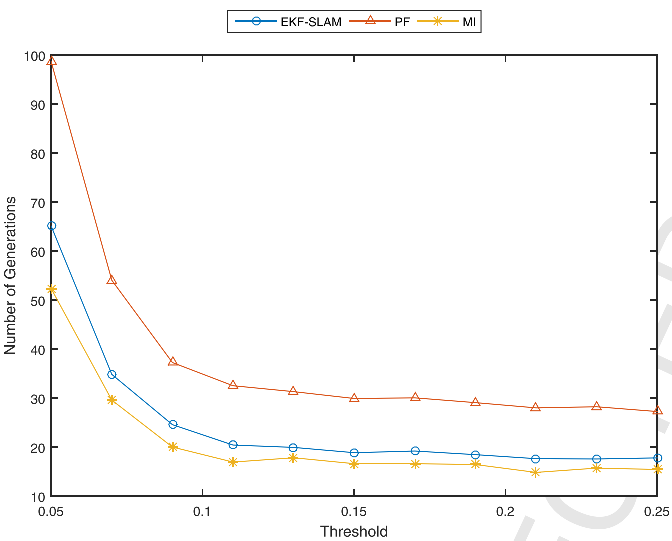

Fig. 20. Average number of generations, after a hundred executions, taken by the EO to satisfy the stop criteria according to *Threshold*.

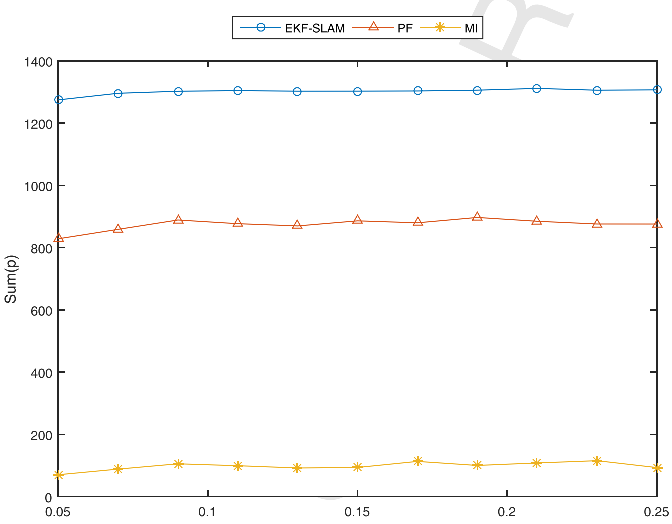

Fig. 21. Average p_t, after a hundred executions, taken by the EO to satisfy the stop criteria according to *Threshold*.

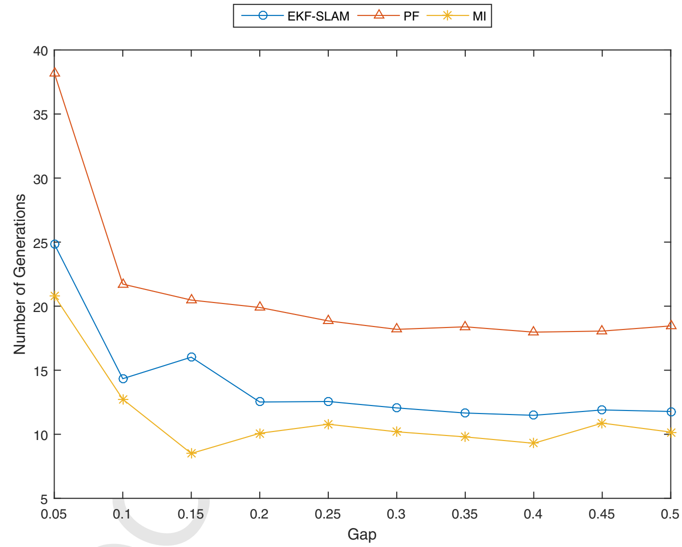

Fig. 22. Average number of generations, after a hundred executions, taken by the EO to satisfy the stop criteria according to *Gap*.

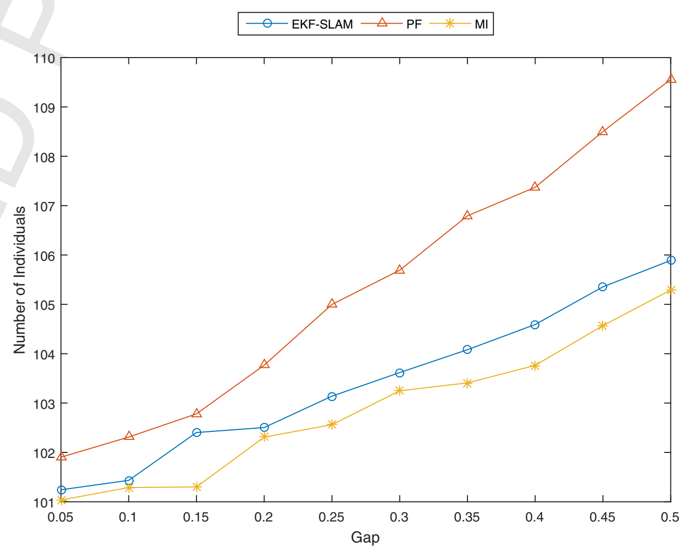

Fig. 23. Average number of individuals created over the algorithm execution, after a hundred executions, taken by the EO to satisfy the stop criteria according to *Gap*.

Table 2
Variable parameters of EO, and its final values after analyzing the influence of each parameter in the number of generations, *error* and p_t for the EKF-SLAM, PF and MI.

Parameter	Final values		
	EKF-SLAM	PF	MI
Allele_Exp	1.3	1.4	1.4
Severity	4	4	4
Mutation_Rate	0.02	0.02	0.02
Population_Exp	2	2	2
Stop_Variation	0.2	0.2	0.2
Threshold	0.25	0.25	0.25
Gap	0.05	0.05	0.05

6.2. Analysis of mo-cGAO

Recent works have shown that the *Holland's building blocks* (BB) *hypothesis* [39] and *Simon's near-decomposability principle* [40] can give insights to help the GA through the search for near-optimal solutions [41].

The BBs occur naturally in the bit estimation problem since the error in a variable can affect a set of other variables, forming a block. In this study, we will consider that each variable is a single BB, in order to simplify our modeling.

In this sense, first, we will estimate the number of individuals that the initial population must have in order to guarantee that a building block of each allele will be present in the initial population. First, we need to evaluate the initial population distribution accordingly to the Poisson Distribution defined by Eq. (8).

Fig. 24 shows that for $\lambda = 0.5$, 1.0, 1.5 and 2.0, the probability of a value x to happen is largely reduced to almost zero for $x \geq 4$, 5, 6 and 7 respectively. For simplicity, we can exchange the Poisson distribution for a uniform probability for $x \leq x_{lim}$, and zero probability for $x > x_{lim}$.

Thus, for further analysis, we will consider a uniform distribution between $1 \leq x \leq x_{lim}$ interval, which we will refer by ℓ (problem size). This distribution does not depend on the algorithm to be converted.

Since our problem is difficult to model, we will estimate only the first size of the initial population and convergence time based on a population sampled. We can expect that the complexity is somewhere between the complexity of the **BinInt** and **OneMax** problems [42], which define Eq. (10), for the initial population, and Eq. (11), for the convergence time. A complete characterization of these estimations can be found in [43].

$$nPop = \ln(\alpha) 2^{k-1} \frac{\sigma_{bb}\sqrt{2m}}{d} \tag{10}$$

where α is defined as the failure probability in finding the optimal solution, k is the BB size, σ_{bb} can be estimated by the standard deviation of the fitness due to the instances of a *BB* from the set of selected individuals (assuming all *BBs* have a similar contribution to the fitness values), d is defined as the difference of the contribution of a *BB* in the fitness of the best solution and the local optima and m is defined as $m = \ell / k$.

Eq. (10) considers that all BB have the same variance σ_{bb}, what is not true in our case since some variables have more influence in the error than other variables. Thus, we should estimate an upper bound value for this parameter.

The values of the variables in Eq. (10) used in this work are: $\alpha = 0.001$ and $k = 1$. The values for σ_{bb} and d will be estimated further on.

$$g = \begin{cases} \dfrac{\sqrt{l}}{\mathbb{I}}\dfrac{\pi}{2}, & \text{for } \textit{OneMax} \\[2ex] \dfrac{\sqrt{3}\,\ln(2)}{\mathbb{I}}\ell, & \text{for } \textit{BinInt} \end{cases} \tag{11}$$

where $\mathbb{I}$ is the *Selection_Intensity*, defined by Eq. (5). This value can be assumed as nearly constant during population evolution and it goes to infinity after the population converges to a single individual.

In order to estimate selection effects on the population, we generated a total of 10,000 individuals with Poisson distribution and re-sampled the entire population. The results obtained are shown in Fig. 25. Note that the re-sampling (using 8-size tournament) makes salient two peaks from the original distribution. We assume that those peaks are in the neighborhood of a local and the global optima.

Thus, we can estimate d as the difference from the *DS* of the best solution found (assumed neighborhood of the global optimum, near the peak in Fig. 25 with the highest DS after resampling by tournament) and the *DS* of the local optima (in the neighborhood of the other peak in Fig. 25), which results in $d = |959 - 6| = 953$.

It is not possible to calculate an exact value for σ_{bb} and a proper estimation for σ_{bb} would be calculate *error* and p_t for all solutions p varying each p_i, with $i = 0 \ldots p_c$, independently, what would give $(p_c + 1)^n$ combinations, what is exactly the same thing as making an exhaustive search. On the other hand, we can major this estimation by using $\sigma^2 = \ell \sum_{i=1}^{n} \sigma_{bb_i}^2 = \ell \sigma_{bb}^2$ (that implies in $\sigma > \sigma_{bb}$), and replacing σ_{bb} by σ in Eq. (10), where $\sigma = 859.094$ is the covariance in the population without tournament in Fig. 25.

Note that we still do not have a defined value for l which is usually known while developing compact genetic algorithms. In our specific case, it depends on λ, which will be estimated as described in sequel.

As stated in Section 5.3, the parameter λ used in the population generation affects d, which also affects the initial population and the number of generations to converge. For this reason, we decided to evaluate the influence of the parameter λ over the number of bits and generations of mo-cGAO, in order to define a suitable value for ℓ.

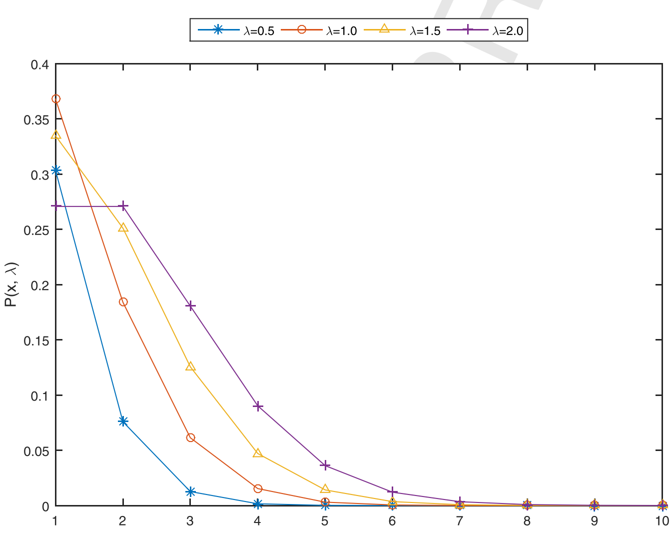

Fig. 24. Poisson distribution parameterized by λ.

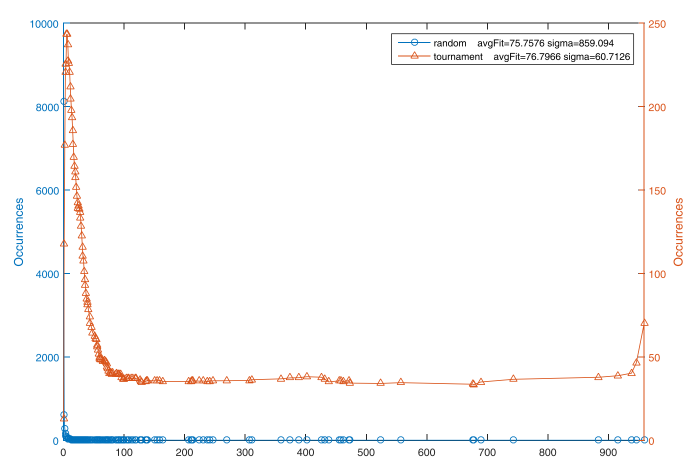

Fig. 25. Occurrences of *DS* in a thousand initialized with Poisson distribution, $\lambda = 0.5$, individuals before (line and circles) and after re-sampling (line and triangles) the set with 8-size tournaments.

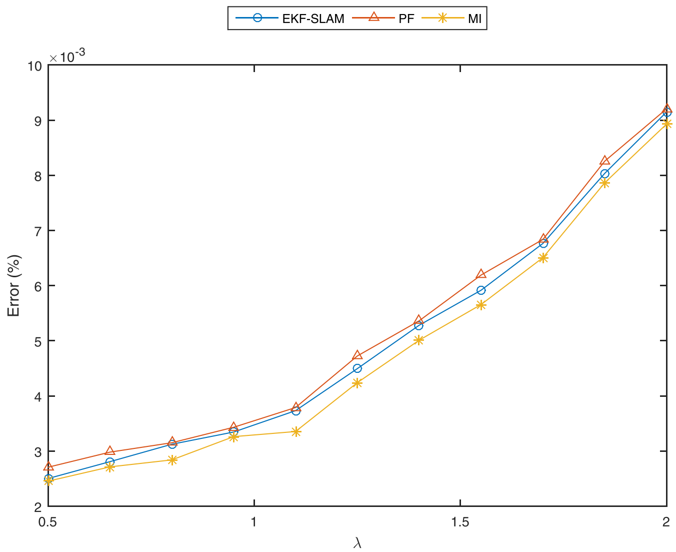

Fig. 26. Average error of mo-cGAO final solution in 100 executions according to λ (from the Poisson distribution used to generate the initial population).

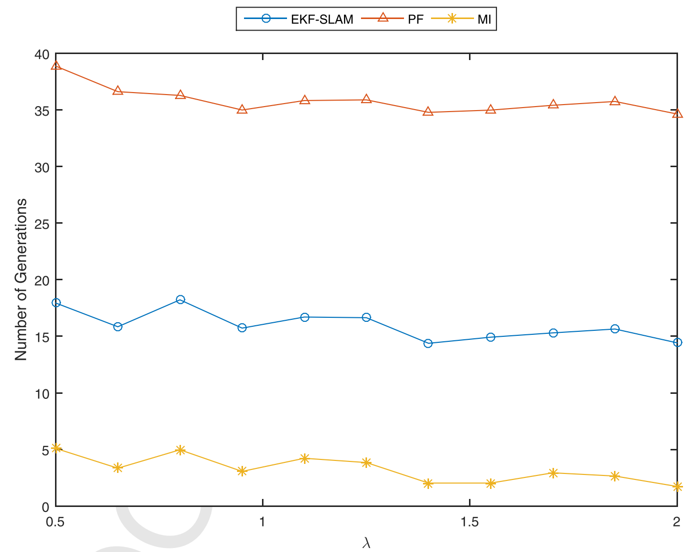

Fig. 28. Average g of mo-cGAO to convergence in 100 executions according to λ (from the Poisson distribution used to generate the initial population).

Table 3
Bounds values for number of generations, g, to convergence according to the problems **BinInt** and **OneMax**.

	EKF-SLAM	*PF*	*MI*
g_{BinInt}	21	20	19
g_{OneMax}	13	13	12

For the tests, we used *treshold* = 5 as a limit to the stop criteria defined in Section 5.3. Using ℓ=64 (the maximum value for ℓ) as base for this study, we set *nPop* = 71 and $\lambda = 1$.

Fig. 26 shows the average of 100 executions of mo-cGAO as function of λ in the Poisson distribution (see Eq. (8) and Fig. 24), where we can correlate that the error increases with λ since the number of bits decreases with a large λ (Fig. 24). Fig. 27 shows that the general number of bits is reduced with λ, since the p_i values can be smaller, forcing the mo-cGAO to find solutions with slightly smaller p_t which respect $error < E_{max}$ in Fig. 26. Thus, it is expected that the mo-cGAO finds solutions with less number of bits when using larger λ.

Fig. 28 shows the average number of generations, g, according to λ. The small range in y-axis indicates that large λ values lead to individuals with a small number of bits in the initial population (**Step 1**, mo-cGA algorithm). Note that, in this case, the initial population is still large and the probability of having individuals with a small number of bits will decrease in a small initial population.

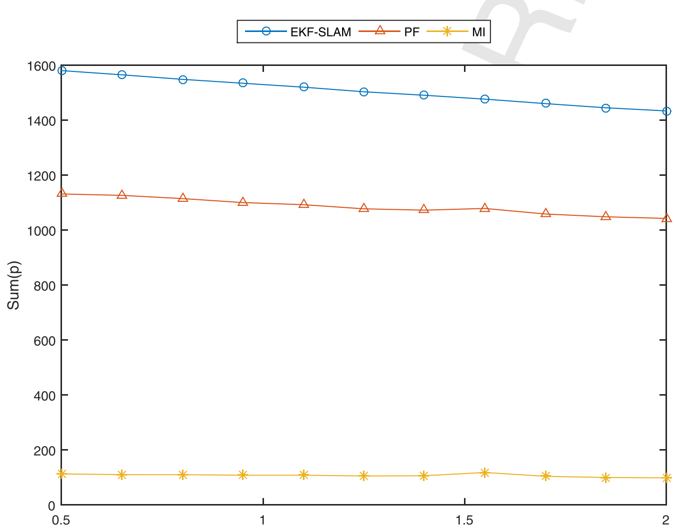

Fig. 27. Average p_t of mo-cGAO final solution according to λ (from the Poisson distribution used to generate the initial population).

As shown in [3], reducing the number of bits impacts the algorithm robustness, and reducing the number of generations is the key for a "not so slow" algorithm. Those results indicate that $\lambda = 0.5$ generates an adequate trade-off among all aspects evaluated. Thus, we decided to use $\lambda_{EKF-SLAM} = \lambda_{PF} = \lambda_{MI} = 0.5$, giving us ℓ=4 ($x_{lim} = 4 = \ell$ for $\lambda = 0.5$ as discussed with Fig. 24), and $\sigma_{bb} = \frac{\sigma}{2}$ consequently (Fig. 25). Thus, Eq. (10) results in $nPop_{EKF-SLAM} = 18$, $nPop_{PF} = 17$ and $nPop_{MI} = 16$ for the EKF-SLAM, PF and MI problems.

We measured the selection intensity over a hundred executions of mo-cGAO, during its quasi-constant interval, which resulted on average in $\mathbb{I} = 0.231 \pm 0.0021$. Eq. (11) resulted in the values presented in Table 3.

The values to determine the initial population size were overestimated by Eq. (10), so this size might be a little different in practice. We can study the population size effect in the *error*, g and p_t to set a more adequate value.

6.2.1. Empirical evaluation of mo-cGAO parameters

Fig. 29 shows the average *error* in function of *nPop* where a large initial population helps to find better individuals, resulting in solutions with small error. Fig. 30 shows the average g to convergence in function of the initial population. The number of generations decreases with the initial population, since a larger initial population increases the probability of relevant individuals to happen, reducing the efforts to find them.

Fig. 30 shows that the number of generations taken by the mo-cGAO to find a solution decreases with the initial population size as expected, since the probability of good individuals to be presented in the initial populations grows with the initial population size. Note that, it is desirable to reduce the number of generations in order to reduce the number of individuals evaluated, but the initial population also needs to be evaluated. Thus, increasing the initial population reduces the number of individuals to be evaluated through the generations, but also increases it in the initial population evaluation. Since our goal

Please cite this article in press as: L.S. Rosa, et al., Design and analysis of evolutionary bit-length optimization algorithms for floating to fixed-point conversion, Appl. Soft Comput. J. (2016), http://dx.doi.org/10.1016/j.asoc.2016.08.035

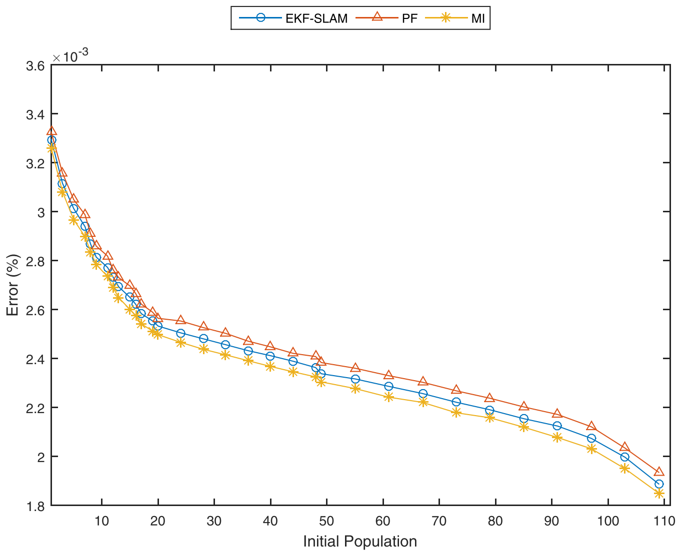

Fig. 29. Average *error* of the mo-cGAO in 100 executions according to the initial population.

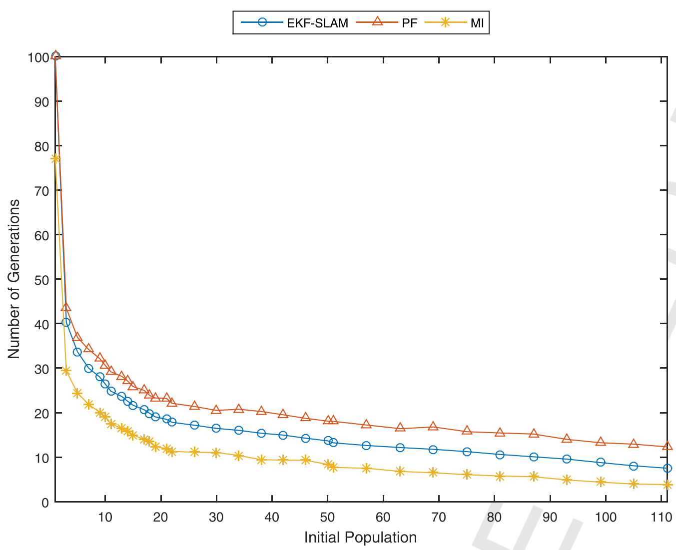

Fig. 30. Average *g* of mo-cGAO in 100 executions according to the initial population.

is to minimize the number of individuals evaluations, which is given by *Number_Individuals* = 2 × *g* + *nPop* we choose *nPop* which gives the minimal number of individuals in Fig. 30 resulting in $nPop_{EKF-SLAM} = 19$, $nPop_{PF} = 18$ and $nPop_{MI} = 17$. These values are near half of the estimated with Eq. (10), since some parameters were overestimated.

Fig. 31 shows the average p_t in function of the initial population, indicating the mo-cGAO efficiency in exploring the search space.

6.2.2. Final mo-cGAO parameters values

Table 4 summarizes the final value for each parameter described from mo-cGAO evaluation, estimated in Section 6.2.

6.3. Methods comparison

The methods solutions are compared running EKF-SLAM, PF and MI algorithms over a total of 1, 000 different executions. The number of times that the condition $error < E_{max}$ is satisfied is defined as the *Hit_Rate*. This evaluation was carried on all training sets β_j, $j = 1 \ldots 100$, for the FO, EO, and mo-cGAO algorithms.

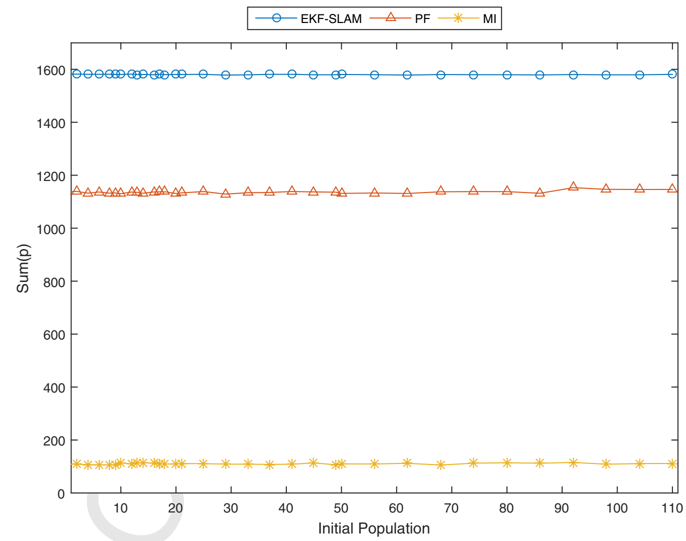

Fig. 31. Average mo-cGAO final solution number of bits according to the initial population.

Table 4
Variable parameters of mo-cGAO, and its final values after analyzing the influence of each parameter in the number of generations, error and p_t for the EKF-SLAM, PF and MI.

Parameter	Final values		
	EKF-SLAM	PF	MI
Threshrold	5	5	5
λ	0.5	0.5	0.5
nPop	19	18	17

Table 5 summarizes the average *Hit_Rate*, the number of *error* calculations, the $\sum_{i=1}^{n} p_i$ for each solution given by FO, EO and mo-cGAO over the training sets β_j, and the number of the variables of each algorithm.

Table 5 shows that mo-cGAO is 6.9× faster on average for the EKF-SLAM, 10.2× for the Particle Filter and 9.2× for the matrix inversion. This last value shows that mo-cGAO efficiently exploits the solutions space even for algorithms with few variables, reducing the gap in relation to EO presented in a previous work [3].

mo-cGAO calculates the *error*, which is the bottleneck of the conversion process fewer times than other methods for all algorithms. Since the measures are made based on the error calculation, this speed up does not depend significantly on the training set β, making possible to obtain a more robust solution by increasing the number of elements in β without a relevant impact on the speed up of the mo-cGAO over the EO. We emphasize that a single *error* calculation take up to 15 min for the EKF-SLAM on an *2.5 GHz Intel core i5* processor with 6 Gb of *RAM*.

mo-cGAO algorithm smooths the gap for algorithms with few variables, which is shown by the previous MI speedup of 1.16× from FO to EO and the speed up of 9.2× from EO to mo-cGAO. The increasing speed up in function of the decreasing size of the algorithm happens once EO had a decreasing speed up due the decreasing size of the algorithm.

Table 5 indicates that EO and mo-cGAO bit reduction grows with the number of variables, what implies in a less robust result, as shown by the *Hit_Rate*. It can be justified once *Fine Optimize* solutions have more bits for most of the variables. However, the robustness of EO solutions is not a critical issue once there are hardware independent solutions that can be applied [34].

Furthermore, the small number of extra bits found by mo-cGAO in relation to EO is relevant bits. In other words, they impact on

Please cite this article in press as: L.S. Rosa, et al., Design and analysis of evolutionary bit-length optimization algorithms for floating to fixed-point conversion, Appl. Soft Comput. J. (2016), http://dx.doi.org/10.1016/j.asoc.2016.08.035

Table 5
Number of variables and average *Hit_Rate*, the number of *error* calculations, and p_t, ± standard variation, for each solution given by the EO and by the FO for each respective training set β_j.

Algorithms	Optimize	# variables	$\overline{Hit}\_\overline{Rate}$ (%)	Avg # of *error* calculations	p_t ($\times 10^3$)
EKF-SLAM	FO	107	98.2 ± 0.3	963 ± 10	1.641 ± 0.009
	EO	107	97.1 ± 0.4	373 ± 13	1.576 ± 0.011
	mo-cGAO	107	97.8 ± 0.3	54 ± 15	1.581 ± 0.010
PF	FO	43	98.2 ± 0.2	808 ± 9	0.986 ± 0.008
	EO	43	97.1 ± 0.3	550 ± 11	0.944 ± 0.010
	mo-cGAO	43	97.2 ± 0.2	54 ± 14	0.950 ± 0.009
MI	FO	10	99.3 ± 0.1	331 ± 2	0.110 ± 0.001
	EO	10	98.8 ± 0.1	285 ± 4	0.106 ± 0.005
	mo-cGAO	10	99.1 ± 0.1	31 ± 7	0.108 ± 0.004

Table 6
Bounds comparison between our proposed method and the precise method presented in [20].

	[20]		Proposed	
	Lower	Upper	Lower	Upper
Polly approx	0	0.6932	0	0.6933
B-spline 3	−0.1667	0	−0.1667	0
rand	−192	128	−64	128
mitch	−719	641	−32	641
rat	−6.1E+08	3.34E+11	−4	3.34E+11

the *Hit_Rate*, which means that these extra bits are not placed on variables that do not need them.

Finally, in order to guarantee the precision of our method we direct compare the range of the values of its estimations with the ones found in [20] presented in Table 6. To make this comparison, we have implemented the test-benches and estimated their (m_i, p_i) values with our proposed mo-cGAO and calculated their respective bounds, which are values directly comparable to the ones presented in [20]. These test-benches are not representative for our proposed approach since they do not contain unpredictable `for` loops and are small enough to not justify the usage of a training-test-based approach as ours, but this corroborates with the accuracy of our proposed approach since their results are comparable with a precise method approach on their class of algorithms.

7. Conclusions

In this work, we presented EO and mo-cGAO as optimization methods to the bit-lengths estimation for a floating to fixed-point conversion as well as a systematic study to define parameter values for such methods.

The bit-lengths estimation during the conversion from floating to fixed-point demand significant computational processing when dealing with unpredictable algorithms, commonly found in many fields of application as robotics. This is caused by the big training sets to estimate the error.

Heuristics based on the error calculation have presented poor quality results that are achieved after a considerable computation time. Evolutionary approaches, on the other hand, are known for their exploration capability. They are usually able to return good solutions within a short computational time.

The mo-cGAO propose in this paper is an estimation of distribution algorithm that integrates the exploration idea of Evolutionary approaches with the probability distribution of solutions in the search space.

In the floating to fixed-point conversion, the application of the mo-cGAO accelerates the bit-lengths estimation. This improves project decisions related with the more appropriated data type to a given design. Furthermore, the reduced bit-lengths leads to a more compact hardware, with lower energy consumption and a possibly higher maximum frequency.

The coherency of the theoretical results for the mo-cGAO parameters with the experimentally estimated ones, presented in Section 6.2, shows that the difficulty of the problem is correctly supposed to be between the **BitInt** and **OneMax** problems. Such theoretical model for the mo-cGAO indicates that no other evolutionary approach will have a better performance than the mo-cGAO adjusted according to the model without losing the confidence that the algorithm will find, if not the best, a near-optimal solution.

As future work, we encourage research exploring different BBs sizes and its structures to further improve the bit estimation problem efficiency.

The results in Table 6 shows that our proposed mo-cGAO find bounds comparable if not better than the formal approaches, which do not guarantee to find the best solution, but guarantee error obedience. On the other hand, our proposed approach cannot guarantee either error obedience or optimality, although we have theoretical evidence of near-optimal solutions as discussed before.

Acknowledgments

The authors would like to thank FAPESP (Ref. 2014/14918-2), for the financial support given to the development of this project, and Marcillyanne Gois for helping us with the writing verification.

References

[1] R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, P. Marwedel, Scratchpad memory: design alternative for cache on-chip memory in embedded systems, in: Proceedings of the Tenth International Symposium on Hardware/Software Codesign, CODES'02, ACM, New York, NY, USA, 2002, pp. 73–78, http://dx.doi.org/10.1145/774789.774805.
[2] A. Malinowski, H. Yu, Comparison of embedded system design for industrial applications, IEEE Trans. Ind. Inform. 7 (2) (2011) 244–254, http://dx.doi.org/10.1109/TII.2011.2124466.
[3] L. Rosa, C. Toledo, V. Bonato, Accelerating floating-point to fixed-point data type conversion with evolutionary algorithms, Electron. Lett. 51 (2015) 244–246, http://dx.doi.org/10.1049/el.2014.3791.
[4] G. Harik, F. Lobo, D. Goldberg, The compact genetic algorithm, IEEE Trans. Evol. Comput. 3 (4) (1999) 287–297, http://dx.doi.org/10.1109/4235.797971.
[5] R. Smith, M. Self, P. Cheeseman, Estimating uncertain spatial relationships in robotics, in: Autonomous Robot Vehicles, Springer-Verlag New York, Inc., New York, NY, USA, 1990, pp. 167–193 http://dl.acm.org/citation.cfm?id=93002.93291.
[6] D. Fox, S. Thrun, W. Burgard, F. Dellaert, Particle filters for mobile robot localization, in: A. Doucet, N. de Freitas, N. Gordon (Eds.), Sequential Monte Carlo Methods in Practice, Statistics for Engineering and Information Science, Springer, New York, 2001, pp. 401–428, http://dx.doi.org/10.1007/978-1-4757-3437-9_19.
[7] M.L. James, G.M. Smith, J. Wolford, Applied Numerical Methods for Digital Computation, vol. 2, Harper & Row, New York, 1985.
[8] M. Moyers, D. Stevens, V. Chouliaras, D. Mulvaney, Implementation of a fixed-point FastSLAM 2.0 algorithm on a configurable and extensible VLIW processor, in: IEEE International Conference on Eletronics Circuits and Systems (ICECS), Tunisia, 2009.
[9] G. Mingas, E. Tsardoulias, L. Petrou, An FPGA implementation of the SMG-SLAM algorithm, Microprocess. Microsyst. 36 (3) (2012) 190–204.

[10] T. Hill, AccelDSP synthesis tool floating-point to fixed-point conversion of MATLAB algorithms targeting FPGAs, in: White Papers, Xilinx, 2006, p. 18.
[11] P. Belanovic, M. Rupp, Automated floating-point to fixed-point conversion with the fixify environment, in: The 16th IEEE International Workshop on Rapid System Prototyping, 2005 (RSP 2005), IEEE, 2005, pp. 172–178.
[12] D. Menard, D. Chillet, O. Sentieys, Floating-to-fixed-point conversion for digital signal processors, EURASIP J. Appl. Signal Process. 2006 (2006) 77, http://dx.doi.org/10.1155/ASP/2006/96421.
[13] C. Shi, R.W. Brodersen, An automated floating-point to fixed-point conversion methodology, in: Proc. IEEE Int. Conf. on Acoust., Speech, and Signal Processing, 2003, pp. 529–532.
[14] A. Banciu, E. Casseau, D. Menard, T. Michel, Stochastic modeling for floating-point to fixed-point conversion, in: 2011 IEEE Workshop on Signal Processing Systems (SiPS), 2011, pp. 180–185, http://dx.doi.org/10.1109/SiPS.2011.6088971.
[15] S. Roy, P. Banerjee, An algorithm for converting floating-point computations to fixed-point in MATLAB based FPGA design, in: Proceedings of the 41st Annual Design Automation Conference, DAC'04, ACM, New York, NY, USA, 2004, pp. 484–487, http://dx.doi.org/10.1145/996566.996701.
[16] A.B. Kinsman, N. Nicolici, Bit-width allocation for hardware accelerators for scientific computing using SAT-modulo theory, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 29 (3) (2010) 405–413, http://dx.doi.org/10.1109/TCAD.2010.2041839.
[17] A. Kinsman, N. Nicolici, Automated range and precision bit-width allocation for iterative computations, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 30 (9) (2011) 1265–1278, http://dx.doi.org/10.1109/TCAD.2011.2152840.
[18] D. Boland, G.A. Constantinides, Automated precision analysis: a polynomial algebraic approach, in: 2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2010, pp. 157–164.
[19] D. Boland, G.A. Constantinides, A scalable approach for automated precision analysis, in: Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA'12, ACM, New York, NY, USA, 2012, pp. 185–194, http://dx.doi.org/10.1145/2145694.2145726.
[20] D. Boland, G.A. Constantinides, Bounding variable values and round-off effects using Handelman representations, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 30 (11) (2011) 1691–1704, http://dx.doi.org/10.1109/TCAD.2011.2161307.
[21] K. Makino, M. Berz, Taylor models and other validated functional inclusion methods, Int. J. Pure Appl. Math. 4 (4) (2003) 379–456.
[22] O. Sarbishei, Y. Pang, K. Radecka, Analysis of range and precision for fixed-point linear arithmetic circuits with feedbacks, in: 2010 IEEE International High Level Design Validation and Test Workshop (HLDVT), 2010, pp. 25–32, http://dx.doi.org/10.1109/HLDVT.2010.5496667.
[23] D.P. Boland, G.A. Constantinides, Word-length optimization beyond straight line code, in: Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, ACM, 2013, pp. 105–114.
[24] L. de Souza Rosa, V. Bonato, A method to convert floating to fixed-point EKF-SLAM for embedded robotics, J. Braz. Comput. Soc. 19 (2) (2013) 181–192, http://dx.doi.org/10.1007/s13173-012-0092-4.
[25] V. Bonato, E. Marques, G. Constantinides, A floating-point extended Kalman filter implementation for autonomous mobile robots, J. Signal Process. Syst. 56 (1) (2009) 41–50, http://dx.doi.org/10.1007/s11265-008-0257-8.
[26] S. Thrun, D. Fox, W. Burgard, F. Dellaert, Robust Monte Carlo localization for mobile robots, Artif. Intell. 128 (1–2) (2001) 99–141, http://dx.doi.org/10.1016/S0004-3702(01)00069-8.
[27] S. Hong, M. Bolic, P.M. Djuric, An efficient fixed-point implementation of residual resampling scheme for high-speed particle filters, IEEE Signal Process. Lett. 11 (5) (2004) 482–485.
[28] H. Abd El-Halym, I. Mahmoud, S. Habib, Proposed hardware architectures of particle filter for object tracking, EURASIP J. Adv. Signal Process. 2012 (1) (2012), http://dx.doi.org/10.1186/1687-6180-2012-17. Q4
[29] S.-A. Li, C.-C. Hsu, W.-L. Lin, J.-P. Wang, Hardware/software co-design of particle filter and its application in object tracking, in: 2011 International Conference on System Science and Engineering (ICSSE), IEEE, 2011, pp. 87–91.
[30] J. Gallagher, S. Vigraham, G. Kramer, A family of compact genetic algorithms for intrinsic evolvable hardware, IEEE Trans. Evol. Comput. 8 (2) (2004) 111–126, http://dx.doi.org/10.1109/TEVC.2003.820662.
[31] E. Mininno, F. Cupertino, D. Naso, Real-valued compact genetic algorithms for embedded microcontroller optimization, IEEE Trans. Evol. Comput. 12 (2) (2008) 203–219, http://dx.doi.org/10.1109/TEVC.2007.896689.
[32] C.F.M. Toledo, M. da Silva Arantes, R.R.R. Oliveira, A.C.B. Delbem, A hybrid compact genetic algorithm applied to the multi-level capacitated lot sizing problem, in: Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC'13, ACM, New York, NY, USA, 2013, pp. 200–205, http://dx.doi.org/10.1145/2480362.2480404.
[33] A. Methods, F. Vainstein, Error detection and correction in numerical computations, in: H. Mattson, T. Mora, T. Rao (Eds.), Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, vol. 539 of Lecture Notes in Computer Science, Springer, Berlin/Heidelberg, 1991, pp. 456–464, http://dx.doi.org/10.1007/3-540-54522-0_133.
[34] T. Bailey, J. Nieto, J. Guivant, M. Stevens, E. Nebot, Consistency of the EKF-SLAM algorithm, in: 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2006, pp. 3562–3568.
[35] T. Bäck, D.B. Fogel, Z. Michalewicz, Evolutionary Computation 1: Basic Algorithms and Operators, vol. 1, CRC Press, 2000.
[36] S.A. Frank, M. Slatkin, Fisher's fundamental theorem of natural selection, Trends Ecol. Evol. 7 (3) (1992) 92–95, http://dx.doi.org/10.1016/0169-5347(92)90248-A.
[37] P.K. Shukla, K. Deb, On finding multiple Pareto-optimal solutions using classical and evolutionary generating methods, Eur. J. Oper. Res. 181 (3) (2007) 1630–1652, http://dx.doi.org/10.1016/j.ejor.2006.08.002.
[38] K. Deb, P. Zope, A. Jain, Distributed computing of Pareto-optimal solutions with evolutionary algorithms, in: C. Fonseca, P. Fleming, E. Zitzler, L. Thiele, K. Deb (Eds.), Evolutionary Multi-Criterion Optimization, vol. 2632 of Lecture Notes in Computer Science, Springer, Berlin/Heidelberg, 2003, pp. 534–549, http://dx.doi.org/10.1007/3-540-36970-8_38.
[39] H. John, Holland, Adaptation in Natural and Artificial Systems, 1992.
[40] H.A. Simon, The Sciences of the Artificial, vol. 136, MIT Press, 1996.
[41] J.P. Martins, C.M. Fonseca, A.C. Delbem, On the performance of linkage-tree genetic algorithms for the multidimensional knapsack problem, in: Bridging Machine Learning and Evolutionary Computation (BMLEC) Computational Collective Intelligence, Neurocomputing 146 (2014) 17–29, http://dx.doi.org/10.1016/j.neucom.2014.04.069.
[42] D.E. Goldberg, A design approach to problem difficulty, in: The Design of Innovation: Lessons from and for Competent Genetic Algorithms, Springer US, Boston, MA, 2002, pp. 71–100, http://dx.doi.org/10.1007/978-1-4757-3643-4_6.
[43] M.K. Crocomo, Algoritmo de otimizaçao bayesiano com detecçao de comunidades, Universidade de São Paulo, 2012 (Ph.D. thesis).