Performance of Evolutionary Algorithms on the Root Identification Problem in Geometric Constraint Solving

Robert Joan-Arinyo[1], M. V. Luzón[2] and Enrique Yeguas[3]

[1]Universitat Politècnica de Catalunya, robert@lsi.upc.edu
[2]Universidad de Granada, luzon@ugr.es
[3]Universidad de Córdoba, eyeguas@uco.es

ABSTRACT

The Root Identification Problem is at the heart of CAD systems. In parametric constraint-based CAD, it can be seen as selecting one solution to a system of nonlinear equations among a potentially exponential number of solutions and has been classified as a NP-hard problem. Evolutionary algorithms have proven to be an effective technology for solving general constraint-satisfaction problems. However, very little is known on how evolutionary algorithms actually perform when applied to geometric problems. In this work, we develop a statistical model to describe the performance of evolutionary algorithms applied to solve the Root Identification Problem. The model allows to figure out the solution quality yielded by the algorithm for a given available run-time and vice versa. The model performance is empirically validated over a benchmark with a very large search space.

Keywords: geometric constraint solving, root identification, evolutionary algorithms.
DOI: 10.3722/cadaps.2010.45-57

1 INTRODUCTION

Modern Computer Aided Design (CAD) and Manufacturing systems [17] are built on top of parametric geometric modeling engines and geometric constraint solving is arguably a core technology of CAD and, by extension, of managing product design data. Since the introduction of parametric design by Pro/Engineer in the 1980s, every major CAD system has adopted geometric constraint solving into its design interface. Most prominently, 2D constraint solving has become an integral component of sketchers on which most feature-based design systems are built. See [10] and references therein. Beyond applications in CAD and manufacturing, geometric constraint solving is also applicable in other fields like virtual reality and is closely related in a technical sense to theorem proving. For solution techniques, geometric constraint solving also borrows heavily from symbolic algebraic computation and matroid theory.

The CAD field has developed sketching systems that automatically instantiate geometric objects from a rough sketch, annotated with dimensions and constraints input by the user. The sketch only has to be topologically correct where dimensions and constraints are normally not yet satisfied. Then the system checks whether the dimensions and constraints coherently define an object. If so, the geometric object is actually built.

Geometric problems defined by constraints have an exponential number of solutions in the number of geometric elements involved. Generally, the user is only interested in one instance such that besides fulfilling the geometric constraints, exhibits some additional properties. This solution instance is called the intended solution. Since users interact with CAD and Manufacturing systems in real time, and the search space is exponential, devising efficient techniques to select the intended solution is paramount. Selecting a solution instance amounts to selecting one among a number of different roots of a nonlinear equation or system of equations. In [4], the problem of selecting a given root was named the *Root Identification Problem* , from now on denoted RIP.

Evolutionary algorithms have proven to be an effective technology for solving general constraint-satisfaction problems. However, very little is known on how evolutionary algorithms actually perform when applied to geometric problems. In previous works, [12], a preliminary study was conducted to asses the potential behavior of a number of metaheuristics, [2], applied to solve the RIP in two-dimensional geometric constraint solving where reduced size problem instances were considered. The study showed that the most promising metaheuristics were clearly in the population-based category: specifically Population-Based Incremental Learning (PBIL), [14], and Cross generational elitist selection Heterogeneous recombination and Cataclismic mutation (CHC), [7].

However, metaheuristics do not always reach an optimal solution, even for long computation times. In addition, it is often difficult to obtain an analytical prediction of either the achievable solution quality within a given computation time or the time taken to find a solution of a given quality. The assessment of these conflicting performance measures is critical for the evaluation of metaheuristics, in particular evolutionary algorithms, and their application to the RIP in geometric constraint solving. Given the lack of theoretical guidelines and the stochastic nature of most metaheuristics, such performance assessments are best carried out by experimentation. However, since measures of performance such as the solution-cost and run-time can be seen as random variables, the field of statistics provides the appropriate basis for supporting the research on metaheuristics and, in particular, evolutionary algorithms, [5].

In general, literature on metaheuristics has focused on experimental comparisons and performance evaluation of certain algorithms over a problem or problems. The idea of the existence of an underlying performance model for a pair problem instance-metaheuristic is rarely suggested and demonstrated. In particular, as far as we know, metaheuristics have been never applied before to the RIP. The availability of a performance model for metaheuristics and, in particular, for evolutionary algorithms could allow to predict the behavior of an algorithm before running it. An approximate idea of the algorithm efficiency needed to reach a given solution quality could be obtained. Likewise the solution quality that one could expected to achieve after allowing an algorithm to run for a given period of time could be assessed, [11]. For large search spaces in CAD systems, where real time interaction is a must, such a model could be even more useful.

In this paper we develop a model to predict the performance of PBIL and CHC when they are applied to solve the RIP as a function of the problem size and the quality of the sought solution. The model is validated using a benchmark including very large size geometric problems. Experiments have been conducted using the two-dimensional geometric constraint solver described in [21]. However, the methodology applies to any constructive constraint solving method where the space of solutions is represented as a construction plan.

The remainder of this work is organized as follows. Section 2 briefly describes the main concepts involved in the RIP and how it can be formally defined as an optimization problem. Section 3 briefly describes the evolutionary algorithms studied. Section 4 presents the parameters setting for the performance model. The performance model is developed in Section 5. In Section 6 we validate the model, leaving Section 7 to draw some conclusions and to suggest future work.

2 THE ROOT IDENTIFICATION PROBLEM

The RIP in parametric CAD can be seen as selecting one solution to a system of nonlinear equations among a potentially exponential number of solutions, that is, to select one root for each equation in the system. We first briefly describe the context where this problem arises. Then we give the criteria we use to define the intended solution instance, that is, the solution of interest we want to select. Finally, we show how the RIP can be formally defined as an optimization problem.

2.1 Constructive Geometric Constraint Solving

In two-dimensional constraint-based geometric design, the designer creates a rough sketch of an object made out of simple geometric elements like points, lines, circles and arcs of circle. Then the intended exact shape is specified by annotating the sketch with constraints like distance between two points, distance from a point to a line, angle between two lines, line-circle tangency and so on. Fig. 1. shows a mechanism as a geometric constraint problem. This mechanism, known as the Peaucellier's linkage, transforms the circular motion of point p_3, along the circle c_1, into the translation of point p_2 along the straight line l_3. As a geometric constraint problem, Peaucellier's linkage includes six points, $p_i, 1 \le i \le 6$, and two straight segments l_1, l_2. The set of geometric constraints are those listed on the right of Fig. 1. which includes point to point distance constraint, $dpp()$, coincidence, $on()$, and angle between two straight line segments, $angle()$.

Once the user has defined the sketch and the set of constraints, a geometric constraint solver checks whether the set of geometric constraints coherently defines the object and, if so, determines the position of the geometric elements.

Many techniques have been reported in the literature that provide powerful and efficient methods for solving systems of geometric constraints. See [10] and references therein for an extensive analysis of work on constraint solving. Among all the geometric constraint solving techniques, our interest focuses on the graph-based constructive approach, [13].

Constructive solvers have two major components: the analyzer and the constructor. The analyzer symbolically determines whether a geometric problem defined by constraints is solvable. If the problem is solvable, the output of the analyzer is a sequence of construction steps, known as the construction plan that describes how to place each geometric element in such a way that all constraints are satisfied. After assigning specific values to the parameters, the constructor interprets the construction plan and builds an object instance, provided that no numerical incompatibilities arise, for example, computing the square root of a negative value.

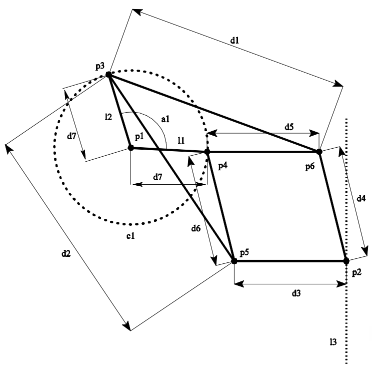
2.2 The Construction Plan

The specific construction plan generated by an analyzer depends on the underlying constructive technique and on how it is implemented. For example, the ruler-and-compass constructive approach is a well-known technique where each constructive step in the plan corresponds to a basic operation solvable with a ruler, a compass and a protractor. In practice, this simple approach solves most useful geometric problems. Fig. 2 shows a construction plan for the object in Fig. 1., generated by the ruler-and-compass geometric constraint solver reported in [13] and available in [21].

Function names in the plan in Fig. 2 are self explanatory. For example, function $line2P()$ defines a straight line through two points, $circleCR()$ defines a circle by its center and radius, and functions $ilc()$, $icc()$ denote line-circle intersection and circle-circle intersection respectively. Parameter s_i in the last two functions identifies one out of the two possible intersections. We will call it the sign.

2.3 The Root Identification as an Optimization Problem

In general, a well constrained geometric constraint problem has an exponential number of solution instances with respect to the number of geometric elements in the problem, [8]. For example, consider a geometric constraint problem that properly places n points with respect to each other. Assume that the points can be placed serially, each time determining the next point q by two distances from two already placed points, say p_1 and p_2. In general, each point can be placed in two different locations corresponding to the intersection points of two circles. Therefore, for n points, once the first two points have been placed, we could have up to 2^{n-2} solutions.



1.	dpp(p_3, p_6, d_1)	8.	dpp(p_3, p_1, d_7)
2.	dpp(p_3, p_5, d_2)	9.	on(p_1, l_1)
3.	dpp(p_2, p_5, d_3)	10.	on(p_4, l_1)
4.	dpp(p_2, p_6, d_4)	11.	on(p_1, l_2)
5.	dpp(p_4, p_6, d_5)	12.	on(p_3, l_2)
6.	dpp(p_4, p_5, d_6)	13.	angle(l_1, l_2, a_1)
7.	dpp(p_1, p_4, d_7)		

Fig. 1: Geometric problem defined by constraints. Left: problem, right: constraints.

1.	$p_1 = pointXY(0, 0)$	9.	$p_5 = icc(x_2, x_3, s_2)$
2.	$p_3 = pointXY(0, d_7)$	10.	$x_4 = circleCR(p_3, d_1)$
3.	$l_2 = line2P(p_1, p_3)$	11.	$x_5 = circleCR(p_4, d_5)$
4.	$l_1 = lineAP(l_2, p_1, -a_1)$	12.	$p_6 = icc(x_4, x_5, s_3)$
5.	$x_1 = circleCR(p_1, d_7)$	13.	$x_6 = circleCR(p_5, d_3)$
6.	$p_4 = ilc(l_1, x_1, s_1)$	14.	$x_7 = circleCR(p_6, d_4)$
7.	$x_2 = circleCR(p_3, d_2)$	15.	$p_2 = icc(x_6, x_7, s_4)$
8.	$x_3 = circleCR(p_4, d_6)$		

Fig. 2: Construction plan for Peaucellier's linkage.

The problem of finding a specific real solution to a geometric constraint problem has been classified as NP-hard in [8]. Since geometric constraint solving is at the heart of CAD and Manufacturing systems where user-system interactivity in real time is a must, devising techniques to efficiently solve the RIP would be a great accomplishment.

In the technique considered in this work, the RIP is solved by overconstraining the geometric constraint problem. The technique has two different steps. In the first step, the user defines the set of constraints. To avoid over-constrained situations, geometric constraints are defined as belonging to two different categories. One category includes the set of constraints needed to specifically solve the geometric constraint problem at hand. The other category includes a set of extra constraints or predicates on the geometric elements with which the user identifies the intended solution instance.

In the second step, the solver generates the space of solution instances, as a construction plan. Then, a search of the solution instances space is performed seeking for a solution instance which maximizes the number of extra constraints fulfilled. In this context, solving the RIP amounts to solving a general constraint-satisfaction problem expressed as a constraint optimization problem.

Evolutionary algorithms have proven to be an effective technology for solving general constraint-satisfaction problems, [6]. With the aim of applying evolutionary algorithms, we show how our RIP can be formalized as a constraint optimization problem. Recall from Section 2.2 that we consider ruler-and-compass constructive geometric constraint solving where geometric operations correspond to quadratic equations, thus each constructive step has at most two different roots.

Let s_j denote the sign associated by the solver with the j-th intersection operation, either line-circle intersection, ilc, or circle-circle intersection, icc, occurring in the construction plan. Since we are interested only in solution instances that are actually feasible, that is, solution instances where no numerical incompatibilities arise in the constructor, we only need to consider the integer parameter s_j taking value in the set $S_j = \{0,1\}$ that characterizes each intersection point.

Assume that L is the total number of ilc plus icc intersection operations in the construction plan. We define the index associated with the construction plan as the ordered set $I = \{s_1, \ldots, s_j, \ldots, s_L\}$ with

$s_j \in S_j; 1 \leq j \leq L$. Therefore the Cartesian product of sets $\Im = S_1 \times \ldots \times S_L$ defines the space where the solution instances to the geometric constraint problem belong to.

A construction plan which is a solution to a geometric constraint problem can be seen as a function of the index I. Moreover, the construction plan can be expressed as a first order logic formula, see [16]. Let $\Psi(I)$ denote this first order logic formula. Clearly, the set of indices $I_F = \{I \in \Im \mid \Psi(I) = true\}$ is the space of feasible indices, that is the set of indices each selecting one solution to the geometric constraint problem. This set of indices is the allowable search space, [6].

Let Φ denote the first order logic formula defined by the conjunction of the extra constraints given to specify the intended solution instance. Let f be a (possibly real-valued) function defined on $\Psi(I) \wedge \Phi$ which has to be optimized. Then, according to [6], the triple $< \Im, f, \Psi(I) >$ defines a constraint optimization problem where finding a solution means finding an index I in the allowable search space I_F with an optimal f value. In simple terms, the problem we are studying can be stated as:

> Let G be a geometric problem defined by a set of constraints C. Let $P(I)$ be a construction plan generated by a constructive solver that defines the space of instance solutions to G as a function of the set of indices I. Let C' be the set of extra constraints that characterizes the intended solution instance. Find an index $I \in I_F$ such that the number of constraints in C' fulfilled by the solution instance $P(I)$ is maximum.

3 THE EVOLUTIONARY ALGORITHMS STUDIED

In previous works, [12], we conducted a preliminary study to asses the potential behavior of a number of metaheuristics applied to solve the RIP. The study considered single-point and population-based metaheuristics according to the classification given in [2]. The single-point search methods applied were local search algorithms, simulated annealing, tabu search, and multistart local search. The population-based search methods applied were genetic algorithms, estimation of distribution algorithms and ant colony optimization.

The results showed that the most promising algorithms were clearly in the population-based category, specifically PBIL and CHC. In this section, we first define both the coding scheme and the fitness function. Then, for the sake of completeness, we give a brief description for each algorithm.

3.1 Coding Scheme and Fitness Function

When dealing with evolutionary algorithms, two general concepts need to be defined according to the specific problem at hand: the coding scheme and the fitness function.

Recall from Section 2.3 that the set of signs that identifies each intersection point is $S = \{0,1\}$. Clearly, each solution instance to the geometric constraint problem can be coded as a binary string I of length L. Therefore the Cartesian product $I = S^L$, defines the space where the solution instances to the geometric constraint problem belong to.

We solve the RIP by overconstraining the geometric constraint problem, that is, the intended solution instance to a well constrained problem is specified by defining a set C' of predicates on the geometric elements. As extra constraints, the user can apply the usual geometric constraints or specific constraints. Examples of specific constraints we apply are $PointOnSide(p, line(p_i, p_j), side)$, $CwOrientation(p_i, p_j, p_k)$, and $CcwOrientation(p_i, p_j, p_k)$. Considering the straight line through points p_i, p_j, oriented from p_i to p_j, the first fixes the half space, say the side, where point p must lay. The second and third fix the usual clock-wise and counterclockwise orientations of three given points.

Since the goal is to select a solution instance such that the number of extra constraints fulfilled is maximal, with each solution instance I we associate the fitness function

$$f(I)=\sum_{k=1}^{|C'|}P_k(I) \tag{3.1}$$

where $|C'|$ is the number of extra constraints defined, and $P_k(I)$ evaluates the solution instance identified by the index I with respect to the extra constraint $P_k \in C'$ and returns value 1 if the solution I fulfills the constraint and 0 otherwise.

The computational cost of evaluating this fitness function depends quadratically on the number of signs in the index, $|I|$, see [10], and linearly on the number of extra constraints, $|C'|$, see [16]. Therefore, the computational requirements are high.

3.2 CHC and PBIL

The CHC algorithm, [7], is a nontraditional genetic algorithm which combines a conservative selection strategy that always preserves the best individuals found so far with a radical, highly disruptive recombination operator that produces offsprings that are maximally different from both parents.

The parameters affecting the evolution of the CHC algorithm for which the user must provide a value are:

- Population size (PS): number of individuals in the population evolving in the search process.
- Divergence rate (DR): percentage of the best global solution found used as a pattern to construct the new population in the restart stage. Values are in $[0,1]$.
- Difference threshold (D): maximum similarity degree allowed between two individuals in the population in the crossover stage. Values range in $[0, L/2]$, where L is the length of the string that encodes the individuals in the population.
- Best individuals (M): best individuals considered when the population is reinitialized.

The PBIL algorithm, [14], is a general heuristic search algorithm inspired by the standard genetic algorithm. PBIL is a kind of evolutionary algorithm where the genotype of an entire population is evolved rather than individual members. Contrary to the standard genetic algorithm it does not maintain a population of individuals but instead PBIL contains a Probability Vector (PV). The mechanisms used in PBIL are derived from those used in competitive learning.

The main parameters affecting the PBIL algorithm evolution are:

- Population size (PS): number of samples in the population to be generated per generation.
- Mutation probability (MP): probability of mutation in each PV position. Values are in $[0,1]$.
- Mutation shift (MS): amount for mutation to affect the PV shifting. Values are defined in $[0,1]$.
- Learning rate (LR): learning rate that regulates the speed with which the PV approaches to the best found solution. Values are in $[0,1]$.

4 PARAMETERS SETTING FOR CHC AND PBIL

In previous works, [12], we conducted a preliminary study to tune CHC and PBIL algorithms in order to obtain the best performance when they are used to solve the RIP. There, we considered problems with search spaces of $2^{18}, 2^{19}$ and 2^{20} instances. We applied an empirical methodology along with a statistical ANalysis Of VAriance (ANOVA), [5].

In this section, a parameter setting for each problem size and evolutionary algorithm, CHC and PBIL, is estimated by means of a simple linear regression method, [22].

4.1 Preliminaries

To study the behaviour of PBIL and CHC algorithms as a function of their parameters we have applied an empirical methodology along with an ANOVA statistical analysis of the experimental results. To avoid doing a complete factorial design of experiments, we have taken into account the statistical nature of the samples obtained from our experiments and the limitations of the different proposals: racing algorithms, sequential parameter optimization, etc., [15].

Since investigating the behavior of the evolutionary algorithms with all parameters variable would be hard to accomplish, we will focus on those parameters whose influence on the algorithms are considered as fundamental which are listed in Section 3.2. These parameters are called factors. The factor levels are the set of discrete different values assigned to a factor in an experiment. For each factor studied, several levels have been considered. First, for each parameter, a central value has been selected among those suggested by the specific literature. Then we defined a number of additional levels (either four or six) evenly distributed with respect to the central value, half of them smaller and half of them greater than the central value. A specific assignment of factors levels to the parameters is called a treatment.

We have considered a representative benchmark of problems with sizes of 2^{18}, 2^{19} and 2^{20}, [23]. For each treatment and run we recorded the actual number of fitness evaluations performed, the run-length. The maximum number of evaluations allowed before declaring a run as a failure is the maximum run-length, $rl_{\max}$. Since our context requires real time interaction, [13], $rl_{\max}$ was set to 30000 evaluations.

An observation is the mean run-length estimated by

$$\hat{E}(RL) = \frac{1}{k}\sum_{i=1}^{k} rl_i + \frac{(n_r - k)}{k} rl_{\max} \tag{4.1}$$

where k is the number of successful runs (95% of the extra constraints fulfilled) and rl_i is the run-length of the i-th successful run, [11]. The total number of runs was $n_r = 50$, each triggered with an initial random seed.

For the safe usage of ANOVA (or, as a matter of fact, any parametric test) observations must fulfill independence, normality and homocedasticity, [20]. Independence was guaranteed by selecting a different random seed for each CHC and PBIL run. Since the sample size is very large, the Central Limit Theorem guarantees normality and homocedasticity was checked applying Levene's test, [5]. However, normality and homocedasticity were checked by performing 50 observations for each treatment and each problem. This experimental setup yielded, for each problem instance, 875 series for PBIL and 735 series for CHC (one series for each treatment considered) of run-length values, $\hat{E}(RL)$.

To elucidate the influence of the parameters on the algorithms performance, we have conducted a comprehensive statistical analysis of the empirical results by using ANOVA. The independent variables are the algorithm parameters and the dependent variable is the mean run-length, $\hat{E}(RL)$, required to find a solution. We have conducted unifactorial analysis, multifactorial analysis, post hoc tests and best parameter level selection. First, one-way ANOVA has been used to determine if the effect of each individual parameter on the algorithm result is significant. Second, multiple factor ANOVA has been applied to elucidate how all the possible combinations of factors affect the algorithm performance and the importance of that influence. Third, post hoc tests have allowed to select the best values for each parameter differing among the results yielded by each factor level.

4.2 Parameters Setting Models

Given a problem size and an evolutionary algorithm, specific values must be assigned to the set of parameters which determine the evolution of the algorithm. Those values are commonly chosen in practice by trial and error, tuned by hand, taken from other fields or by adaptation and self-adaptation mechanisms, [15].

We have identified linear regression models to figure out the best parameters settings, for CHC and PBIL algorithms, as a function of the index length L for problems with search space bounded by 2^L, [12]. Problems in the benchmark were randomly obtained using Henneberg sequences, [3], an easy and common way to generate useful classes of constraint graphs. The resulting benchmark is available at [23]. The identified models for CHC and PBIL algorithms are given in Tab. 1. The independent variable has been transformed to guarantee linearity, normality and homocedasticity in the model,[22].

The goodness of these models, [9], was validated by computing the Pearson correlation coefficients, whose values are close to 100%. Besides, the T-test, the ANOVA and residuals analysis confirmed the model validity, [20]. Therefore our models are well-supported by the experimental samples.

CHC		*PBIL*	
Parameter	***Model***	***Parameter***	***Model***
Population Size	$\widehat{PS} = \lfloor 19.7807 + 0.3805L \rfloor$	*Population Size*	$\widehat{PS} = \lfloor 7.7391 + 1.6505L \rfloor$
Difference Threshold	$\widehat{D} = \lfloor 0.8414 + 0.2115L \rfloor$	*Learning Rate*	$\widehat{LR} = e^{-1.6339-0.0029L}$
Divergence Rate	$\widehat{DR} = \min(0.1355 + 0.0627\ln L, 1)$	*Mutation Probability*	$\widehat{MP} = \min(0.0118 + 0.0032\sqrt{L}, 1)$
Best Individuals	$\widehat{M} = \lfloor 1.3609 + 0.0136L \rfloor$	*Mutation Shift*	$\widehat{MS} = \min(0.0177 + 0.0013\sqrt{L}, 1)$

Tab. 1: Parameters setting models for CHC and PBIL.

5 PBIL AND CHC PERFORMANCE MEASUREMENT

We evaluate PBIL and CHC algorithms performance when they are applied to solve the RIP, following the ideas reported in [11]. First we identify empirical run-time distributions. Then, a theoretical statistical distribution which describes this behavior is identified. Finally, the properties which determine the algorithm behavior and performance are described using the model identified.

5.1 Run-Length Distributions

The run-time behavior of evolutionary algorithms, in general, is characterized by random variables. Therefore, the detailed knowledge of run-time distributions provides key information for the analysis of these algorithms.

Instead of actually measuring algorithm runs in terms of CPU-time, it is often preferable to use representative operation counts as a more machine independent measure of an algorithm's performance. In this way, using a suitable cost model for the algorithm operations facilitates the comparison of different algorithms.

In our study, the computational cost for both PBIL and CHC algorithms corresponds, basically, to computing the number of extra predicates fulfilled by the set of solution instances in the population, see Section 4.1. Therefore, as run-length unit or step, we choose computing the number of extra predicates fulfilled by a solution instance. In these conditions, instead of run-time distributions we get run-length distributions, RLDs in what follows. In the scenario we are working, user-system interaction is a must. Therefore, we fix a time limit, say $rl_{\max}$, for the run-length.

To actually measure RLDs, we measure empirical RLDs by running PBIL and CHC algorithms for n_r times on a given problem instance up to the limit $rl_{\max}$. A run is declared successful if a solution instance that fulfills all the extra constraints that select the desired solution is found before reaching the cutoff. Then we record for each successful run the number of steps actually performed.

Since the run-time depends on the problem instance and the search space is different for each instance, a run-time distribution must be estimated for each instance and algorithm. From them, we will identify the run-time distribution type corresponding to the global problem, [11].

The empirical run-length distribution is the cumulative distribution associated with the observations. Let $rl(j)$ denote the run length for the j-th successful run over an instance I, then the cumulative empirical RLD is defined as

$$\hat{P}_I(rl \le i) = \left|\{ j : rl(j) \le i \}\right| / n_r \tag{5.1}$$

The averaged empirical RLD for an algorithm over a set T including n_T instances can be defined as

$$\hat{P}_{n_T}(rl \le i) = \left| \left\{ j : \frac{1}{n_T} \sum_{I=1}^{n_T} rl_I(j) \le i \right\} \right| \Big/ n_r \tag{5.2}$$

where $rl_I(j)$ denotes the run length for the j -th successful run over the instance I .

The averaged empirical RLD will be used to summarize the performance of an algorithm on a specific 2^L size problem represented by a significant set T of instances. The averaged empirical RLD samples are obtained by averaging the run-lengths corresponding to the same probability in individual empirical RLDs.

Empirical RLDs were determined for a set of problems in our benchmark, see [23]. The problem size and the number of random instances included in each of them are given in Tab. 2. To guarantee the normality in the samples, the number of runs for each algorithm and instance was $n_r = 50$, [5].

Problem size	2^{18}	2^{19}	2^{20}	2^{25}	2^{30}	2^{35}
# Instances	29	12	12	10	10	10

Tab. 2: Features of the benchmark used for finding a statistical model.

5.2 A Statistical Model for Empirical RLDs

To find a statistical model for the empirical RLDs, a set of statistical continuous distributions were selected, taking into account the nature of the random variable analyzed. The set included the most common distributions to model run-time versus solution quality in non-deterministic search algorithms, [11], that is, Exponential, Weibull, Gamma, Erlang, Normal, Laplace and Cauchy.

Each empirical RLD was fit to each different theoretical statistical continuous distribution, [1]. Then, Percentile-Percentile, Chi-squared, Kolmogorov-Smirnov and Anderson-Darling goodness-of-fit tests were applied representing the most useful and used graphical and quantitative goodness-of-fit tests, see [9].

Among the theoretical continuous distributions essayed, only the Gamma distribution does not allow to reject the Null Hypothesis, i.e. we have not enough evidence to reject that the data follow the specified distribution, with a 95% confidence level, for all the problem instances. Therefore the Gamma distribution was chosen to model the RLDs for both CHC and PBIL.

Then an averaged empirical RLD, see Eqn. 5.2., for each problem size and algorithm was computed. The goodness-of-fit for each averaged empirical RLD to the theoretical statistical distributions listed above was tested applying the same procedure as in the individual empirical RLDs.

Again, the distribution which models the averaged empirical RLDs was the Gamma distribution. Fig. 3 depicts for each problem size in the benchmark the averaged Gamma RLD.

5.3 The Gamma Model and the Problem Size

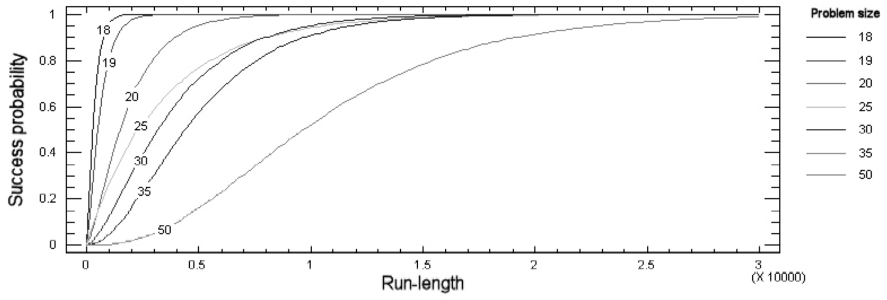
Experimental values for the mean and the standard deviation for RLDs suggest a linear relationship with the problem size, 2^L . Assuming linear behavior, we have

$$\overline{RL} = \hat{\alpha}_{0,\overline{RL}} + \hat{\alpha}_{1,\overline{RL}} L; \quad \sigma_{RL} = \hat{\alpha}_{0,\sigma_{RL}} + \hat{\alpha}_{1,\sigma_{RL}} L \tag{5.3}$$

To figure out linear coefficients, $\hat{\alpha}_0$ and $\hat{\alpha}_1$, we applied the maximum likelihood method, [1], in a simple linear regression framework. The resulting values are given in Tab. 3.

The Gamma distribution is defined in terms of two parameters: a scale factor, β , and a shape factor, k , [19]. Given an empirical RLD with mean run-length $\overline{RL}$ and standard deviation σ_{RL} , that fits into a Gamma distribution, these relationships hold, [5].

$$\overline{RL} = \frac{k}{\beta} \quad , \quad \sigma_{RL} = \frac{\sqrt{k}}{\beta} \tag{5.4}$$



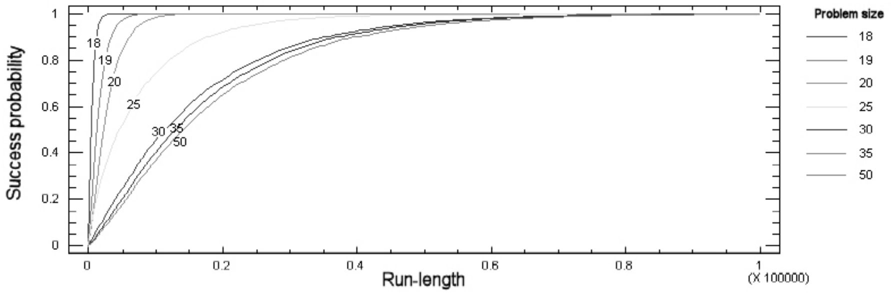


Fig. 3: Fitting between empirical RLD and Gamma RLD. Top: CHC, bottom: PBIL.

Therefore, once $\overline{RL}$ and σ_{RL} have been computed for a problem size 2^L, we will be able to figure out values for β and k which identify the specific Gamma distribution that applies.

Algorithm	*Parameter*	$\hat{\alpha}_{0,P}$	$\hat{\alpha}_{1,P}$
CHC	$\overline{RL}$	-4936.36	308.173
	σ_{RL}	-2107.86	169.024
PBIL	$\overline{RL}$	-8135.06	612.462
	σ_{RL}	-6767.08	538.613

Tab. 3: Estimated linear regression coefficients for statistical measures in Gamma distribution.

6 VALIDATING THE GAMMA MODEL

In the previous section, we have empirically and statistically shown that the Gamma model fits both the empirical RLDs and the averaged empirical RLDs corresponding to problems with size of up to 2^{50} solutions. Now we shall validate the Gamma model by considering problems with sizes up to 2^{100} and two questions:

- Question 1: If the quality of the desired solution is fixed, give an estimate for the run-length the algorithm needs to find an appropriate solution instance.

- Question 2: Given a limit run-length, estimate the quality of the solution instance found by the evolutionary algorithm.

We will compare the results predicted by the model with the figures yielded by actually running the algorithms on the benchmark.

6.1 Experimental Setting

To validate the Gamma model, [18], we have considered a benchmark with four different problem sizes 2^{60}, 2^{70}, 2^{80} and 2^{100}. Each problem size contained $n_T = 10$ different problem instances randomly generated following the properties of the instances analyzed in previous sections. The benchmark is available at [23].

Parameter settings for each algorithm and problem size were defined using the model developed in Section 5. They are given in Tab. 4. The number of runs of each algorithm on each instance was $n_r = 50$ in order to keep the normality in the results, [5].

	CHC		*PBIL*	
Problem	$\hat{k}$	$\hat{\beta}$	$\hat{k}$	$\hat{\beta}$
2^{60}	2.8465	2.1001E-04	1.2541	4.3831E-05
2^{70}	2.9269	1.7594E-04	1.2608	3.6297E-05
2^{80}	2.9841	1.5134E-04	1.2656	3.0972E-05
2^{100}	3.0602	1.1824E-04	1.2718	2.3947E-05

Tab. 4: Gamma parameters settings predicted with the model developed.

6.2 Question 1

To answer this question, the solution quality, measured as the probability of finding a solution instance which fulfils the set of extra constraints, was assigned the value $p_{suc} = 0.9$ to guarantee a good solution quality, [16]. The corresponding run lengths needed for the algorithms to find solutions with this quality were computed according to the model developed in Section 5. Predicted and experimental values are given in Tab. 5. Values predicted by the Gamma distribution are in the column labeled $\overline{RL}_p$. Experimental values are given as 95% confidence intervals bounded by values in columns labeled $\overline{RL}_{\min}$ and $\overline{RL}_{\max}$.

Experimental run-length values always define an interval which includes the predicted value. Moreover, experimental confidence interval bounds and predicted values are within the same order of magnitude. Therefore, the Gamma model correctly describes the behavior of CHC and PBIL algorithms concerning the run-length they need to find a solution with a given quality.

	CHC			*PBIL*		
Problem	$\overline{RL}_p$	$\overline{RL}_{\min}$	$\overline{RL}_{\max}$	$\overline{RL}_p$	$\overline{RL}_{\min}$	$\overline{RL}_{\max}$
2^{60}	24324	22002	26387	62301	59419	67981
2^{70}	29673	27093	31998	75537	67377	78302
2^{80}	35021	33111	37895	88772	80123	95166
2^{100}	45718	42288	47712	115244	108791	122541

Tab. 5: Estimated and experimental run-length needed for 90% of success.

6.3 Question 2

Now the run-length limit was fixed as $rl_{\max} = rl_{suc}$, where rl_{suc} is the run-length estimated by the Gamma model for each problem size with probability of success $p_{suc} = 0.9$. The corresponding level of success for the answers yielded by the algorithms were predicted by the model developed in Section 5. Predicted values are presented in Tab. 6 under the column labeled $\%suc_p$. Experimental quality was obtained as a 95% confidence interval bounded by $\%suc_{\min}$ and $\%suc_{\max}$.

As in the run-length values, the experimental success ratio defines an interval that always includes the predicted success. In all the cases studied, the algorithms find solutions with qualities higher than 0.8. Notice that the differences between the predicted and experimental values are always smaller than 10%. Therefore, the Gamma model describes the behavior of CHC and PBIL algorithms concerning the quality of the solutions found.

	CHC			*PBIL*		
Problem	$\%suc_p$	$\%suc_{\min}$	$\%suc_{\max}$	$\%suc_p$	$\%suc_{\min}$	$\%suc_{\max}$
2^{60}	0.900	0.878	0.979	0.900	0.839	0.951
2^{70}	0.900	0.837	0.988	0.900	0.892	0.976
2^{80}	0.900	0.828	0.929	0.900	0.848	0.966
2^{100}	0.900	0.819	0.941	0.900	0.872	0.988

Tab. 6: Estimated and experimental success for a rl_{suc} run-length.

7 CONCLUSIONS AND FUTURE WORK

We have shown that the performance of CHC and PBIL algorithms can be properly modeled by a Gamma distribution assuming that performance is measured in terms of a platform-independent Run-Length Distribution. The model has been validated by comparing performances predicted by the model for problems with a search space up to 2^{100} solutions with experimental performances yielded by actually running the algorithms on the benchmark. Performances yielded by CHC and PBIL evolutionary algorithms prove that they are both effective and efficient in solving the RIP in Geometric Constraint Solving.

We plan to explore three different lanes. First, we will include CHC and PBIL in the kernel of the geometric system for parametric design developed by our group. Then we will try to check whether our model can be extended to new evolutionary algorithms of interest for the community. Finally, in our study two different sources of randomness are present: the nondeterministic nature of the algorithm itself, and the random selection of problem instances. We plan to conduct a study to assess the contribution of each randomness source to the algorithms' final behavior. Knowing the effect of each randomness source would allow us to fit evolutionary algorithms to specific requirements in general constraint-satisfaction problems and, in particular, in parametric CAD.

ACKNOWLEDGEMENTS

This research has been supported by FEDER and by CICYT of the Spanish Ministerio de Educación y Ciencia under the project TIN2007-67474-C03-01.

REFERENCES

[1] Aldrich, J.: R. A. Fisher and the making of maximum likelihood 1912-1922, Statistical Science, 12(3), 1997, 162–176.

[2] Birattari, M.; Zlochin, M.; Dorigo, M.: Towards a theory of practice in metaheuristics design: A machine learning perspective, Theoretical Informatics and Applications, 40(2), 2006, 353–369.

[3] Borcea, C.; Streinu, I.: The number of embeddings of minimally rigid graphs, Discrete and Computational Geometry, 31(2), 2004, 287–303.
[4] Bouma, W.; Fudos, I.; Hoffman, C.; Cai, J.; Paige, R.: Geometric constraint solver, Computer-Aided Design, 27(6), 1995, 487–501.
[5] Devore, J. L.: Probability and Statistics for Engineering and the Sciences, Duxburg and Brooks Cole, Pacific Grove, CA, 6th edition, 2004.
[6] Eiben, A.; Ruttkay, Z.: Constraint-satisfaction problems, In Bäck, T., Fogel, D., and Michalewicz, Z., editors, Handbook of Evolutionary Computation, chapter C5.7, pages C5.7:1–C5.7:5. Institute of Physics Publishing Ltd and Oxford University Press, 1997.
[7] Eshelman, L.-J.: The CHC adaptative search algorithm: How to safe search when engaging in nontraditional genetic recombination, Foundations of Genetic Algorithms I, 1991, 265–283.
[8] Fudos, I.; Hoffmann, C.: A graph-constructive approach to solving systems of geometric constraints, ACM Transactions on Graphics, 16(2), 1997, 179–216.
[9] Hill, T.; Lewicki, P.: Statistics: Methods and Applications, Statsoft, Inc., Tulsa, OK, 2006.
[10] Hoffmann, C.; Joan-Arinyo, R.: A brief on constraint solving, Computer-Aided Design and Applications, 2(5), 2005, 655–663.
[11] Hoos, H.; Stützle, T.: Stochastic Local Search: Foundations & Applications, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.
[12] Joan-Arinyo, R.; Luzón, M.-V.; Yeguas, E.: Parameter Tuning of PBIL and CHC Evolutionary Algorithms Applied to Solve the Root Identification Problem, Applied Soft Computing, doi: 10.1016/j.asoc.2009.12.037, 2010.
[13] Joan-Arinyo, R.; Soto-Riera, A.: Combining constructive and equational geometric constraint solving techniques, ACM Transactions on Graphics, 18(1), 1999, 35–55.
[14] Larrañaga, P.; Lozano, J.-A.: Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation, Springer, (2002).
[15] Lobo, F.-G.; Lima, C.-F.; Michalewicz, Z.: Parameter Setting in Evolutionary Algorithms, volume 54 of Studies in Computational Intelligence, Springer, (2007).
[16] Luzón, M.; Soto, A.; Gálvez, J.; Joan-Arinyo, R.: Searching the solution space in constructive geometric constraint solving with genetic algorithms, Applied Intelligence, 22, 2005, 109–124.
[17] Narayan, K.-L.; Rao, K.-M.; Sarcar, M.-M.-M.: Computer Aided Design and Manufacturing. Prentice Hall of India, 1st edition, 2008.
[18] Press, W.-H.; Teukolsky, S.-A.; Vetterling, W.-T.; Flannery, B. P.: Numerical Recipes: The Art of Scientific Computing, Cambridge University Press, New York, NY, 3rd edition, 2007.
[19] Schmelzer, T.; Trefethen, L.-N.: Computing the gamma function using contour integrals and rational approximations, SIAM Journal on Numerical Analysis, 45(2), 2007, 558–571.
[20] Sheskin, D.: Handbook of parametric and nonparametric statistical procedures, Chapman and Hall/CRC, Boca Raton, Florida, 4th edition, 2007.
[21] SolBCN: http://floss.lsi.upc.edu, 2005.
[22] Weisberg, S.: Applied Linear Regression, John Wiley and Sons, Inc., New York, NY, 3rd ed., (2005).
[23] Yeguas, E.: Root identification problem in geometric constraint solving. http://www.uco.es/~in1yeboe/benchmark.html, 2009.