GA and CHC. Two Evolutionary Algorithms to Solve the Root Identification Problem in Geometric Constraint Solving

M.V. Luzón[1], E. Barreiro[1], E. Yeguas, and R. Joan-Arinyo[2]

[1] Escuela Superior de Ingeniería Informática. Universidade de Vigo,
Av. As Lagoas s/n, E-32004 Ourense
{luzon,enrique}@uvigo.es
[2] Escola Técnica Superior d'Enginyeria Industrial de Barcelona.
Universitat Politècnica de Catalunya,
Av. Diagonal 647, 8^a, E-08028 Barcelona
robert@lsi.upc.es

Abstract. Geometric problems defined by constraints have an exponential number of solution instances 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.
Selecting a solution instance amounts to selecting a given root every time the geometric constraint solver needs to compute the zeros of a multi valuated function. The problem of selecting a given root is known as the *Root Identification Problem*.
In this paper we present a comparative study of a basic genetic algorithm against the CHC algorithm. Both techniques are based on an automatic search in the space of solutions driven by a set of extra constraints. A number of case studies illustrate the performance of the methods.

Keywords: Evolutionary algorithms, Constructive geometric constraint solving, Root identification problem, Solution selection.

1 Introduction

Modern computer aided design and manufacturing systems are built on top of parametric geometric modeling engines. The 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 and constraints are normally not yet satisfied. The core of those sketching systems is the geometric constraint *solver*.

Geometric problems defined by constraints have an exponential number of solution instances 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*.

Selecting a solution instance amounts to selecting one among a number of different roots of a nonlinear equation or system of equations. The problem of selecting a given root was named in [1] the *Root Identification Problem*.

Several approaches to solve the Root Identification Problem have been reported in the literature. Examples are: Selectively moving the geometric elements, conducting a dialogue with the constraint solver that identifies interactively the intended solution, and preserving the topology of the sketch input by the user. For a discussion of these approaches see, for example, references [1,7,20] and references therein.

In [16,15] we reported on a new technique to automatically solve the Root Identification Problem for constructive solvers, [1,9]. The technique over-constrains the geometric problem by defining two different categories of constraints. One category includes the set of constraints specifically needed to solve the geometric constraint problem. The other category includes a set of extra constraints or predicates on the geometric elements which identify the intended solution instance. Once the constructive solver has generated the space of solution instances, the extra constraints are used to drive an automatic search of the solution instances space performed by a genetic algorithm, [11,24]. The search outputs a solution instance that maximizes the number of extra constraints fulfilled.

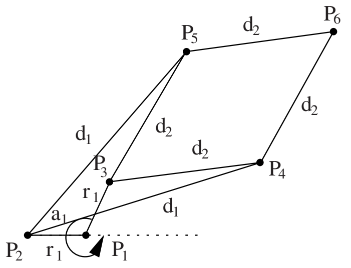
In this paper we study the performance of two evolutive algorithms applied to solve the Root Identification Problem: the basic genetic algorithm and the CHC algorithm. In both cases, the automatic search in the space of solutions is driven by the set of extra constraints.

2 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. A geometric constraint solver then checks whether the set of geometric constraints coherently defines the object and, if so, determines the position of the geometric elements. Figure 1 shows an example sketch of a constraint-based design.

Many techniques have been reported in the literature that provide powerful and efficient methods for solving systems of geometric constraints. For example, see [4] and references therein for an extensive analysis of work on constraint solving. Among all the geometric constraint solving techniques, our interest focuses on the one known as *constructive*.

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 which places each geometric element in such a way that all constraints are satisfied. This sequence is known



$distance(P_1, P_2)$	$= r_1$
$distance(P_1, P_3)$	$= r_1$
$distance(P_2, P_4)$	$= d_1$
$distance(P_2, P_5)$	$= d_1$
$distance(P_3, P_4)$	$= d_2$
$distance(P_3, P_5)$	$= d_2$
$distance(P_4, P_6)$	$= d_2$
$distance(P_5, P_6)$	$= d_2$
$angle(line(P_1, P_3), line(P_2, P_1))$	$= a_1$

Fig. 1. Geometric problem defined by constraints

as the *construction plan.* Figure 2 shows a construction plan generated by the ruler-and-compass geometric constraint solver reported in [14] for the problem depicted in 1. After assigning specific values to the parameters, the constructor interprets the construction plan and builds an object instance, provided that no numerical incompatibilities arise.

1. $P_1 = point(0, 0)$	5. $P_3 = rc(line(P_1, \alpha_2), circle(P_1, r_1))$
2. $P_2 = point(r_1, 0)$	6. $P_4 = cc(circle(P_2, d_1), circle(P_3, d_2))$
3. $\alpha_1 = direction(P_2, P_1)$	7. $P_5 = cc(circle(P_2, d_1), circle(P_3, d_2))$
4. $\alpha_2 = adif(\alpha_1, a_1)$	8. $P_6 = cc(circle(P_4, d_2), circle(P_5, d_2))$

Fig. 2. Construction plan for the problem in Fig. 1

Function names in the plan are self explanatory. For example function *adif* denotes subtracting the second angle from the first one and *asum* denotes the addition of two angles while *rc* and *cc* stand for the intersection of a straight line and a circle, and the intersection of two circles, respectively.

In general, a well constrained geometric constraint problem, [10,13,18], has an exponential number of solutions. 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 by two distances from two already placed points. In general, each point can be placed in two different locations corresponding to the intersection points of two circles. For n points, therefore, we could have up to 2^{n-2} solutions. Possible different locations of geometric elements corresponding to different roots of systems of nonlinear algebraic equations can be distinguished by enumerating the roots with an integer index. For a more formal definition see [7,22].

In what follows, we assume that the set of geometric constraints coherently defines the object under design, that is, the object is generically well constrained and that a ruler-and-compass constructive geometric constraint solver like that reported in [14] is available. In this solver, intersection operations where circles are involved, *rc* and *cc*, may lead to up to two different intersection points, depending on whether the second degree equation to be solved has no solution, one or two different solutions in the real domain. With each feasible *rc* and *cc* operation, the constructor in the solver associates an integer parameter $s_k \in \{-1, 1\}$, that characterizes each intersection point by the sign of the square root in the corresponding quadratic equation. For details on how to compute s_k, the reader is referred to [22].

3 The Root Identification as a Constraint Optimization Problem

We will solve the Root Identification Problem by over-constraining the geometric constraint problem: The intended solution instance to a well constrained problem is specified by defining a set of extra constraints or predicates on the geometric elements. As extra constraint, the user can apply the usual geometric constraints or specific topological constraints like $PointOnSide(P, line(P_i, P_j), side)$, which means that point P must be placed on one of the two open half spaces defined by the straight line through points P_i, P_j, oriented from P_i to P_j. Parameter *side* takes values in {*right, left*}.

Recall that we consider ruler-and-compass constructive geometric constraint solving. In this context, geometric operations correspond to quadratic equations, thus each constructive step has at most two different roots. Let s_j denote the integer parameter associated by the solver with the j-th intersection operation, either *rc* or *cc*, occurring in the construction plan. Since we are interested only in solution instances that actually are feasible, that is, solution instances where no numerical incompatibilities arise in the constructor, we only need to consider integer parameters s_j taking value in the set of signs $D_j = \{-1, 1\}$ that characterizes each intersection point.

Assume that n is the total number of *rc* plus *cc* intersection operations in the construction. We define the *index* associated with the construction plan as the ordered set $I = \{s_1, \ldots, s_j, \ldots, s_n\}$ with $s_j \in D_j, 1 \leq j \leq n$. Therefore the Cartesian product of sets $\mathcal{I} = D_1 \times \ldots \times D_n$ defines the space where the solution instances to the geometric constraint problem belong to.

A construction plan which is 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, [15]. Let $\Psi(I)$ denote this formula. Clearly, the set of indexes $\{I \in \mathcal{I} \mid \Psi(I)\}$ is the space of feasible indexes, that is the set of indexes each defining a solution to the geometric constraint problem. This set of indexes is the *allowable search space*, [5].

Let $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m\}$ be the set of extra constraints given to specify the intended solution instance and let $\Phi = \mathcal{C}_1 \wedge \mathcal{C}_2 \wedge \ldots \wedge \mathcal{C}_m$. Let f be a (possibly real-valued) function defined on $\Psi(I) \wedge \Phi$ which has to be optimized. Then, according to Eiben and Ruttkay, [5], the triple $< \mathcal{I}, f, \Psi(I) >$ defines a *constraint optimization problem* where finding a solution means finding an index I in the allowable search space with an optimal f value.

4 Evolutionary Algorithms

Evolutionary algorithms which model natural evolution processes were already proposed for optimization in the 1960s. The goal was to design powerful optimization methods, both in discrete and continuous domains, based on searching methods on a population, members of which are coded problem solutions, [2].

In previous works [16,15] we shown that evolutionary algorithms are a feasible technique to solve the Root Identification Problem. In this work we compare the performance of two kinds of evolutionary algorithms: Genetic Algorithms and CHC algorithms.

4.1 The Genetic Algorithm

Genetic algorithms (GA) are search algorithms that model sexual reproduction. Sexual reproduction is characterized by recombining two parent strings into an offspring. This recombination is called *crossover*. Crossover is the recombination of traits of the selected members in the hope of producing a child with better fitness levels than its parents. Crossover is accomplished by swapping parts of strings representing two members in the population. GAs were invented by Holland, [12]. Recent surveys can be found in [8] and [11].

4.2 The CHC Algorithm

The main drawback of GAs is the premature convergence: After a few generations, the current population reaches an state where the goal function is not optimal and no longer improves.

The heterogeneous recombination and cataclysmic mutation algorithm (CHC) is an evolutive algorithm with binary coding which tries to avoid the premature convergence by getting a suitable balance between the ability to explore the search space for diversity and the ability to explote the local properties of the search for an appropriate selection. Instead the mutation operation of GAs, CHC includes a restart process that offers many of the benefits of a great size population without the cost of a slower search, [6].

5 Experimental Study

To assess and compare the performance of GA and CHC algorithms when applied to solve the Root Identification Problem, we considered eleven different problems. For each problem a number of extra constraints to select the intended solution instance were defined. The goal was to select one index such that the number extra constraints fulfilled by the associated solution instance was maximum.

The number of indexes in the initial population was always 20 and the maximum number of generations allowed was 30. For GAs the crossover and mutation probabilities were 0.3 and 0.1 respectively. In the search reinicialization, the CHC

Table 1. T test results

Problem	*Algorithm*	*Mean*	*StDev*	*SE mean*	*t*	*Sig. level*
1	CHC GA	32.70 32.32	1.251 1.588	0.125 0.159	-1.879	0.062
2	CHC GA	28.38 27.84	1.324 1.461	0.132 0.146	-2.738	0.007
3	CHC GA	38.48 37.96	1.467 1.825	0.147 0.183	-2.221	0.028
4	CHC GA	34.17 33.94	1.484 1.530	0.148 0.153	-1.079	0.282
5	CHC GA	39.12 37.87	3.059 5.076	0.306 0.508	-2.109	0.036
6	CHC GA	37.14 36.53	1.826 2.129	0.183 0.213	-2.175	0.031
7	CHC GA	46.74 45.71	2.473 3.220	0.247 0.322	-2.537	0.012
8	CHC GA	47.92 47.18	1.857 2.236	0.186 0.224	-2.546	0.012
9	CHC GA	26.91 26.83	0.944 0.954	0.094 0.095	-0.596	0.552
10	CHC GA	31.73 30.60	2.178 4.110	0.218 0.411	-2.429	0.016
11	CHC GA	23.16 22.53	1.779 2.952	0.178 0.295	-1.828	0.069

algorithm kept the 3 best fitting indexes. The remaining 17 indexes in the population were generated using these 3 individuals as templates and randomly changing the 35% of the binary signs, [6].

GA and CHC algorithms were applied to each problem one hundred times. The initial population in each run was the same for both algorithms and the value of the goal function at the end of the run was recorded.

Table 1 summarizes the experimental results. The fourth and fifth column give respectively the mean and the standard deviation of the number of extra constraints fulfilled in the set of one hundred runs. In all cases, the mean for the CHC algorithm was higher that for the GA while the standard deviation for CHC was smaller that for GA. Therefore, CHC showed a better performance in finding the intended solution instance.

To assess whether the mean of the goal functions yielded by each evolutive algorithm are statistically different from each other, we applied a t-test, [23]. Columns labeled *t* and *Sig. level* in Table 1 give the t value for the t-test and the significance level associated respectively. Problems 1, 4, 9 and 11 show a significance level higher than 0.05, the usually accepted level value. Notice, however, that only in problems 4 and 9 the significance level is clearly higher than 0.05. Therefore, we conclude that, in average, the instance solution selected by the CHC algorithm fulfills more extra constraints than that selected by the GA.

6 Conclusions and Future Work

The Root Identification Problem can be solved using GA and CHC evolutive algorithms. The idea is to over-constrain the problem and use the extra constraints to drive the algorithm search.

Experimental results from a significative benchmark show that performance of CHC is better than GA's. On the one hand, CHC algorithms do no show premature convergence. On the other hand, in the average and with a significance level higher than 0.05, the instance solution selected by the CHC algorithm shows a better fitting than that selected by the GA.

Currently we are working in two different directions. One focuses on applying new evolutive algorithms to solve the Root Identification Problem and to study the relative performance. The goal of the other line of research is to define strategies to automatically define values for evolutive parameters (population size, crossover and mutation probabilities, etc) as a function of the geometric constraint problem at hand.

Acknowledgements. This research has been partially supported by FEDER and CICYT under the project TIC2001-2099-C03-01.

References

1. W. Bouma, I. Fudos, C. Hoffmann, J. Cai, and R. Paige. Geometric constraint solver. *Computer-Aided Design*, 27(6):487–501, June 1995.
2. H.J. Bremermann, J. Roghson, and S. Salaff. Global properties of evolution processes. In H.H. Pattee, E.A. Edelsack, L. Fein, and A.B. Callahan, editors, *Natural Automata and Useful Simulations*, pages 3–42. Macmillan, 1966.
3. B.D. Brüderlin. *Rule-Based Geometric Modelling.* PhD thesis, Institut für Informatik der ETH Zürich, 1988.
4. C. Durand. *Symbolic and Numerical Techniques for Constraint Solving.* PhD thesis, Purdue University, Department of Computer Sciences, December 1998.
5. A.E. Eiben and Zs. Ruttkay. Constraint-satisfaction problems. In T. Bäck, D.B. Fogel, and Z. Michalewicz, 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.
6. L.J. Eshelman. The CHC adaptive search algorithm: How to safe search when engaging in nontraditional genetic recombination. *Foundations of Genetic Algorithms*, pages 265–283, 1991.
7. C. Essert-Villard, P. Schreck, and J.-F. Dufourd. Sketch-based pruning of a solution space within a formal geometric constraint solver. *Artificial Intelligence*, 124:139–159, 2000.
8. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, San Mateo, CA, 1993. Morgan Kaufmann.
9. I. Fudos and C.M. Hoffmann. Correctness proof of a geometric constraint solver. *International Journal of Computational Geometry & Applications*, 6(4):405–420, 1996.

10. I. Fudos and C.M. Hoffmann. A graph-constructive approach to solving systems of geometric constraints. *ACM Transactions on Graphics*, 16(2):179–216, April 1997.
11. D.E. Goldberg. *Genetic Algorithms in Search, Optimization , and Machine Learning.* Addison Wesley, 1989.
12. J. H. Holland. *Adaptation in Natural and Artificial Systems.* Ann Arbor: The University of Michigan Press, 1975.
13. R. Joan-Arinyo and A. Soto-Riera. Combining geometric and equational geometric constraint solving techniques. In *VII Congreso Español de Informática Gráfica*, pages 309–324. Eurographics, June 1997.
14. R. Joan-Arinyo and A. Soto-Riera. Combining constructive and equational geometric constraint solving techniques. *ACM Transactions on Graphics*, 18(1):35–55, January 1999.
15. R. Joan-Arinyo and M.V. Luzón and A. Soto. Constructive geometric constraint solving: a new application of genetic algorithms. In *Parallel Problem Solving from Nature-PPSN VII*, volume 1, pages 759–768, 2002.
16. R. Joan-Arinyo and M.V. Luzón and A. Soto. Genetic algorithms for root multiselection in constructive geometric constraint solving. *Computer & Graphics*, 27:51–60, 2003.
17. S.C. Kleene. *Mathematical Logic.* John Wiley and Sons, New York, 1967.
18. G. Laman. On graphs and rigidity of plane skeletal structures. *Journal of Engineering Mathematics*, 4(4):331–340, October 1970.
19. L. Lovász and Y. Yemini. On generic rigidity in the plane. *SIAM Journal on Algebraic and Discrete Methods*, 3(1):91–98, March 1982.
20. M.V. Luzón. *Resolución de Restricciones geométricas. Selección de la solución deseada.* PhD thesis, Dpto. de Informática. Universidade de Vigo., Septiembre 2001.
21. N. Mata. Solving incidence and tangency constraints in 2D. Technical Report LSI-97-3R, Department LiSI, Universitat Politècnica de Catalunya, 1997.
22. N. Mata. *Constructible Geometric Problems with Interval Parameters.* PhD thesis, Dept. LSI, Universitat Politècnica de Catalunya, Barcelona, Catalonia, Spain, 2000.
23. W. Mendenhall and T. Sincich. *Statistics for engineering and the sciences*, 4th Edition. Prentice-Hall, 199.
24. Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, 1996.