http://www.zju.edu.cn/jzus; E-mail: jzus@zju.edu.cn
ISSN 1009-3095 *Journal of Zhejiang University SCIENCE* V.4, No.5, P.532-541, Sep.-Oct., 2003

# Solving geometric constraints with genetic simulated annealing algorithm*

LIU Sheng-Li(刘生礼)†1, TANG Min(唐敏)†2, DONG Jin-Xiang(董金祥)
(*Department of Computer Science, Zhejiang University, Hangzhou 310027, China*)
E-mail: †1jslsl75@yahoo.com.cn; †2tang_m@zju.edu.cn
Received Aug.24, 2002; revision accepted Jan.8, 2003

**Abstract:** This paper applies genetic simulated annealing algorithm (SAGA) to solving geometric constraint problems. This method makes full use of the advantages of SAGA and can handle under-/over- constraint problems naturally. It has advantages (due to its not being sensitive to the initial values) over the Newton-Raphson method, and its yielding of multiple solutions, is an advantage over other optimal methods for multi-solution constraint system. Our experiments have proved the robustness and efficiency of this method.

**Key words:** SAGA, Geometric constraint solving, Variational design
**Documents code:** A **CLC number:** TP391.72

## INTRODUCTION

A geometric constraint problem consists of a set of geometric elements, such as points, lines and planes, and constraints on them, such as constraints of distance, angle, coincidence, and so on. These elements can be positioned with respect to each other by computing a suitable set of coordinates such that the constraints are satisfied.

A geometric constraint problem is well-constrained if there are a finite number of solutions to the problem, while a problem with an infinite number of solutions is under-constrained. It is over-constrained if one constraint can be deleted while the constraint system still has a finite number of solutions. An over-constrained problem may have a solution when the additional constraints are consistent with previous constraints, but over-constrained problems often have no solution.

Geometric constraint solving has been applied widely in many fields, such as mechanical engineering, chemical molecular modeling, and surveying (Durand, 1998). In each of these communities the problem has been approached in a variety of ways and with differing levels of success. It can be classified into four kinds as follows.

### 1. Graph based constraint solving

Graph-based algorithms for solving geometric constraint problems have two phases: an analysis phase and a construction phase. The graph-based approach begins by first constructing a graph representation of the problem. After the constraint graph has been obtained from the given geometric parameter and constraints, the graph is analyzed to determine whether the problem is well-constrained. If the graph is well-constrained, this phase also determines a sequence of steps for solving the problem. The second phase of the graph method takes the construction sequence determined from the first phase and performs the necessary construction steps to actually position the geometric elements (Bouma *et al.*, 1994; Fudos, 1993a; Fudos *et al.*, 1993b; 1997; Hoffmann *et al.*, 1995a; 1995b; 1997; Song *et al.*, 2000).

### 2. Logical inference and term rewriting

This approach applies general logical reasoning techniques to the geometric problem of constraint solving. This approach was used by Aldefeld (1988), Bruderlin (1990), Verroust *et al.* (1992), Joan-Arinyo *et al.* (1997a; 1997b)

* Project (No. 6001107) supported by the National Science Foundation of Zhejiang Province

### 3. Symbolic algebraic approach

The constraints are formulated as a system of algebraic equations. However, instead of applying numerical techniques to determine a solution, general symbolic computations are used to find the solution to the system of equations. Methods such as those of Grobner basis (Bose, 1985; Wu, 1986) techniques can be applied to find symbolic expressions for the solutions. Kondo(1992) used this method to add constraints to the constraint system or delete from it. The details can be found in (Wang, 1998; Lazard, 1999; Sturmfels, 1997; Emiris *et al.*, 1997).

### 4. Numerical algebraic approach

Numerical constraint solvers function by first translating the constraints into a system of algebraic equations. This system is then solved using an iterative technique such as the Newton-Raphson method (Ivan, 1963; Borning, 1981). A positive feature of this approach is that it can handle over-constrained but consistent problems which other techniques may not be able to solve, assuming convergence. However, there are some serious drawbacks with the numerical approach. First is the problem that iterative methods can produce only a single solution. Also, the solution to which it converges depends strongly on the initial configuration.

To overcome the above methods' drawbacks and make use of their advantage, Durand *et al.* (2000) presented a systematic framework. This approach combines geometric reasoning, symbolic reduction, and homotopy continuation. But as pointed out by Hoffman *et al.* (2000), this method is not fast enough and different placement choices for the same template problem have a great effect on the efficiency of homotopy computation (Durand *et al.*, 1999). To overcome this drawback and provide an interactive speed, the constraint system must restrict geometric elements.

David *et al.* (2001) presented a new geometric constraint solving method that can solve two-dimension problems. They represented all the geometric constraints and elements with basic geometric constraints and basic elements. By filling matrix of distances and matrix of angles, the two-dimension constraint problems can be solved. This method can also handle over-constrained and under-constrained problems, but it can not deal with the three-dimension constraint system.

Methods with global convergence properties were proposed (Dennis *et al.*, 1983) based on the theory of nonlinear optimization. These methods are referred to as global, and converge to a solution from almost any initial guess. Convergence is achieved by defining an energy function that decreases as progress is made towards a solution, therefore assuring improvement at each iteration. However, this method can still occasionally fail by ending in a local minimum of the energy function.

An optimization method employing BFGS method is proposed by Ge *et al.* (2000). The distinct advantage of this method is that under- and over-constrained problems can be handled naturally and efficiently and be not sensitive to the initial values. However, for the well-constrained system with many solutions, this method, like the Newton-Raphson Method, can only yield one solution.

This paper introduces genetic simulated annealing algorithm (SAGA) into geometric constraint solving. To begin with, the geometric constraint system is formulated into a system of nonlinear equations. Then, equations are simplified to a certain degree with Gaussian elimination. Finally, SAGA is applied to solve the equations. The second step minimizes the searching space of SAGA through reducing the number of variables and increases solving speed. The SAGA step applies selection, crossover and mutation operators to the initial population to make them "evolve" until they are approximate to the analytic roots of equations. The whole process of SAGA is not concerned about the matrix inversion often encountered in the Newton-Raphson method, which always results in computation instability; so SAGA is robust. A constraint problem that is converted into a problem of optimization does not require that the number of variables be equal to that of the equations, so the under-constrained and over-constrained consistent problems can be handled naturally. In fact under-constrained problems have infinite solutions. Many methods can judge whether a constrained system is under-constrained or not, but can not give a solution. SAGA can present a solution to meet the designer's intent to the utmost, even when the constrained system has infi-

nite solutions. For well-constrained and multi-solution system, SAGA can get all the possible solutions at one time, but many other methods such as the Newton-Raphson method and that of Ge *et al*. (2000) only give a solution at one time and are often trapped by the initial values.

## GENETIC SIMULATED ANNEALING ALGORITHM

### 1. Genetic algorithm (GA)

The Genetic algorithm has become a hotspot in the area of AI and has been applied successfully in the fields of machine learning, engineering optimizing, job scheduling, image processing, etc.

The algorithm itself involves iterating generations in a population of potential answers at each stage selecting certain population elements, and using these selected elements to generate new elements according to the genetic operators. The key of this method is the selection process: in order to provide an intuitive simulation of the biological process of evolution by natural selection, measures must be taken to ensure that, on the whole, the "fittest" elements are chosen to reproduce (Goldberg, 1989).

GA has the benefit of global searching, and will not be trapped in the rapid descending direction introduced by local minima. And with its intrinsic parallelism, the calculation speed can be easily increased by distributive computation methods.

However, GA has a weak ability for local searching, so the "pure" genetic algorithm always has a lower computing efficiency.

### 2. Simulated annealing algorithm (SAA)

As a technique to solve massive optimization problem, SAA has received the attention of many researchers. Its main idea is simulating the growth of single crystals from a molten metal while it cools down slowly (Kirkpatrick *et al*., 1983). It can find the global minimum of object function with random searching. In mathematics, simulated annealing algorithm can be described with the ergodic theory of Markov chain. During searching for the optimal solution, SAA not only accepts optimal solutions, but also the degraded solutions to a certain degree with a random acceptance rule, such as Metropolis rule. The acceptance probability of degraded solutions gradually approaches zero, which makes it possible that SAA jumps over the local minima and makes it converge to the optimization.

However, because the algorithm has no knowledge about the global objective space that has been detected, it is hard to figure out which area contains optimal results more likely, resulting lower computing efficiency.

### 3. Genetic simulated annealing algorithm (SAGA)

The genetic simulated annealing algorithm is an optimization algorithm that integrates genetic algorithm with simulated annealing algorithm. Similar to the simple genetic algorithm, SAGA starts the global searching process from a set of solutions (initial population) produced randomly. Genetic operators such as selection, crossover and mutation are performed over the initial population to generate the new individuals that are to be computed further by simulated annealing algorithm. After this process, the new individuals constitute the next generation. This process is iterated until the solutions are found.

To sum up, SAGA makes full use of the advantage of genetic algorithm and simulated annealing algorithm, and improves the calculation efficiency greatly (Davis, 1991).

A pseudo-code of SAGA can be described as follows:

(1) Deciding the solution phenotype and potential answer space;

(2) Deciding objective function, its mathematical expression and its quantification method;

(3) Deciding how to code and decode the chromosome of the feasible solution;

(4) Deciding how to express the individual fitness;

(5) Deciding the selection, crossover and mutation operator;

$t \leftarrow 0$;

Initializing the initial population $P(t) = \{X_1, X_2, \cdots X_N\}$;

Computing individual fitness in $P(t)$;

Do

{

Individual crossover $P'(t) \leftarrow Crosserver[P(t)]$;

Individual mutation $P''(t) \leftarrow Crosserver[P'(t)]$;

Individual simulated annealing $P'''(t) = SimulatedAnnealing[P''(t)]$;

Calculating the individual fitness of $P'''(t)$;

Individual selection $P(t) \leftarrow \text{Re}\,production[P(t) \cup P'''(t)]$;

$t \leftarrow t + 1$;

}while (terminate condition is not satisfied)

In the following section, we discuss in detail how to combine the new approach with constraint solving.

## CONSTRAINT SOLVING BASED ON SAGA

### 1. Decision variable, constraint condition and optimization model

A constraint problem can be formulated as $(E, C)$, $E: = (e_1, e_2, \ldots e_n)$, where $e_i$ is the geometric element, such as point, line, circle, etc. $C: = (c_1, c_2, \ldots c_m)$, where $c_i$ is the constraint on these elements. Generally the constraint can be formatted as an equation. A constraint system can be formulated as follows:

$$\begin{cases} f_1(x_0, x_1, x_2, \cdots x_n) = 0 \\ f_2(x_0, x_1, x_2, \cdots x_n) = 0 \\ f_3(x_0, x_1, x_2, \cdots x_n) = 0 \\ \quad \cdots \\ f_{m-1}(x_0, x_1, x_2, \cdots x_n) = 0 \\ f_m(x_0, x_1, x_2, \cdots x_n) = 0 \end{cases} \tag{1}$$

$\boldsymbol{X} = (x_0, x_1, \cdots x_n)$;

$x_i$ is the parameter of geometric element. For example, a two-dimension point can be represented as $(x_1, x_2)$, where $x_1$, $x_2$ are coordinates respectively. The aim of constraint solving is to find a vector $\boldsymbol{X}$ satisfying Eq. (1).

$F(X_j)$ is defined as follows:

$$F(X_j) = \sum_{1}^{m} |f_i| \tag{2}$$

It is obvious that Eq. (1) has a real solution $X_j$ if and only if min $F(X_j)$ is zero. So the constraint problem is converted to an optimization problem. One obvious fact for this approach is that the number of equations $m$ is not necessarily the same as the number of variables $n$. Thus it is natural to deal with under- and over-constrained problems.

Decision variable is $x_i$, $i \in [0, n]$, constraint condition is $x_j \in [L_l, L_u]$, $[L_l, L_u]$ is the real value range of variable $x_i$. Optimization model is Eq. (2).

### 2. Coding and decoding schema

In constraint system, an individual can be represented as $X(j) = (x_0, x_1, \cdots x_i, \cdots x_n)$, $x_i \in [L_l, L_u]$. To improve the computation speed, we employ a two-stage coding schema that combines float coding with Gray coding. The genotype of $X(j)$ is as follows:

$X(j): (x_0, x_1, \cdots\cdots, x_{n-1}, x_n)$

The code length of genotype is the number of decision variables. As an individual genotype is the same as its phenotype, some additional translations between genotype and phenotype can be omitted. The individual phenotype is $\boldsymbol{X}(j) = [x_0, x_1, \cdots x_{n-1}, x_n]$.

Because the variation range of some decision variables can not be decided precisely, it is estimated conservatively. As a result of such estimation, it is enlarged and the code is lengthened correspondingly. In order to avoid such disadvantage and improve computation efficiency and precision, we adopt float coding (Michalewicz *et al.*, 1990). After the gene $x_i$ is decided, on which crossover and mutation operations will be performed further, we further apply Gray coding to it. So Gray coding is integrated with float coding efficiently.

We adopt 20-bit Gray code that can represent 1 048 576 numerals ranging from 0 to 1 048 575. The domain $[L_l, L_u]$ of $x_i$ is divided into 1 048 575 well-proportioned intervals. After crossover and mutation is performed on $x_i$, $x_i$ is translated into real number from Gray code according to the domain. At first $x_i$ is translated into decimal $y_i$, then $x_i$ can be calculated as follows:

$$x_i = (L_l - L_u) \times \frac{y_i}{1\,048\,575} + L_l$$

From the above we can see that the two-stage coding schema has two advantages: (1) The float coding schema shortens the code length and improves the computation efficiency; (2) The local Gray coding schema makes crossover and mutation operations convenient.

**3. Fitness variable**

Objective function is $F(\boldsymbol{X}_j) = \sum_{1}^{m} | f_i |$. As the object of optimization is to get the minimum of $F(\boldsymbol{X}_j)$, we define fitness variable $Fit(\boldsymbol{X}_j)$ as follows:

$Fit(\boldsymbol{X}_j)) = 1/e^{\beta F(\boldsymbol{X}_j)}$

$0 \leqslant F(\boldsymbol{X}_j) \leqslant \max, 1/e^{\beta \max} \leqslant Fit(\boldsymbol{X}_j) \leqslant 1$

Theoretically, $F(\boldsymbol{X}_j) \to \infty, 0 < Fit(\boldsymbol{X}_j) \leqslant 1$

Where $\beta$ is an adjustive parameter of fitness. During the computing process, the selection probability of individual is changed to avoid premature convergence by modifying the parameter $\beta$. Generally, $\beta$ is assigned a large value at the earlier stages and a small value at later stages. In our program, $\beta$ is set to 1.25 at the early 25 generations and 0.85 afterwards.

**4. Design of genetic operators**

(1) Selection operator

We adopt simple Proportional Model (De Jong, 1975). Its main idea is that the individual selection probability is in direct proportion to its fitness. Although the randomness of the pure model slows the convergence process, this shortcoming can be overcome by incorporating it with simulated annealing algorithm, which is good at local fast converging.

(2) Crossover operator

One-point crossover is adopted in our algorithm. However, it is different from the general one-point crossover because of the two-stage coding schema.

For a pair of crossover individuals, the position of crossover is generated randomly. Note that the position has $n \times 20$ possibilities, not $n$. For example, if given the crossover position $k$, the other variables can be calculated as follows:

int $p = k/20$;

int $q = k \bmod 20$;

$(p + 1)$ is the position of the crossover gene and $q$ is the crossover position that is relative to $x_{p+1}$ (Fig.1).

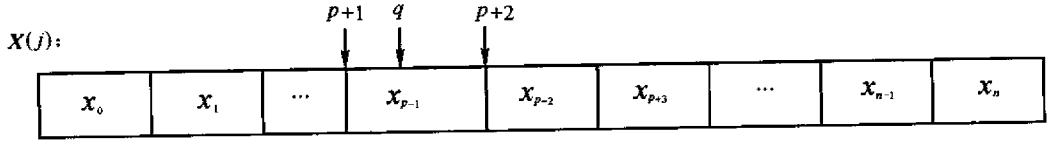


**Fig.1 The crossover schematic plan of two-stage coding**

The crossover process details: all the decision genes $(x_{p+2}, \cdots x_n)$ behind crossover position $q$ in the two individuals are exchanged directly. The genes that include the crossover position $q$ are further coded into binary bits with Gray code schema. The binary codes behind crossover position $q$ in the two genes are exchanged with each other. After exchanging, the values of these two genes are calculated again.

(3) Mutation operator

Simple Mutation is used here and made a little change. After the mutation gene is positioned, a random value ranging between interval $[L_l, L_u]$ substitutes the gene. Although it suffers randomness, this drawback can be overcome by the integration of simulated annealing.

**5. Running parameters**

(1) Population ($P$) size: $M = 200$. Population is initialized with individuals whose genes are generated randomly in the range of $[L_l, L_u]$. The individual whose value is input by user in the sketch is also added into the initial population. The variation range of the gene is different, which will be discussed in detail in the next section.

(2) Termination generation: $T = 100$.

(3) Crossover probability: $P_c = 0.5$.

(4) Mutation probability: $P_m = 0.0005$.

**6. Termination condition**

**Condition 1**: The D-value $\delta$ of adjacent two generations $(j, j + 1)$ is minimum for well-constrained condition, namely $\delta < \varepsilon$, where $\varepsilon$ is a threshold. $\delta$ can be defined as follows:

$\delta = \left| \sum_{k=1}^{M} F(\boldsymbol{X}_{j,k}) - \sum_{k=1}^{M} F(\boldsymbol{X}_{j+1,k}) \right|$. Where $M$ is the population size.

**Condition 2**: $| Fit(\boldsymbol{X}_j) - 1 | \leqslant \varepsilon$ for under-constrained condition, where $\varepsilon$ is a threshold.

Because there are infinite solutions for under-constrained condition, any solution satisfying the constraints is ok.

**Condition 3:** Termination generation $< T$. Therefore the ultimate condition is (Condition 1 ∪ Condition 2 ∪ Condition 3).

### 7. Output result

The individual $\boldsymbol{X}_i$ whose fitness is the largest among the population is selected as the result of the constraint system. Hence it is $fit(\boldsymbol{X}_i) = \max[fit(\boldsymbol{X}_1), \cdots, fit(\boldsymbol{X}_n)]$.

## ADDITIONAL OPTIMIZATION

### 1. Compress the searching space

An individual can be represented as $\boldsymbol{X}(j) = (x_0, x_1 \cdots x_n)$. The variables $(x_0, x_1 \cdots x_n)$ generated from the constraint system are correlated and not like those of other systems' variables, which are independent of each other. So the constraint equations can be simplified by reducing the number of variables and equations according to their correlation, which can shorten the length of the gene code and minimize searching space.

Morgan (1992) gave many examples where symbolic reduction degrades the numerical stability of a system, especially when the reduction is carried out beyond a certain point. Therefore, a balance should be struck between seeking to reduce the number of variables, by symbolic algebraic computation and maintaining stable system. In order to avoid degrading the system stability, we only adopt Gaussian elimination and invariable elimination.

The well-known "Four Point One Line" problem (Hoffmann *et al.*, 2000) can be formulated as follows:

$$x^2 + y^2 + z^2 - r_1^2 = 0 \tag{3}$$

$$(a_1 - x)^2 + y^2 + z^2 - (a_2 u)^2 - r_2^2 = 0 \tag{4}$$

$$(a_3 - x)^2 + (b_3 - y)^2 + z^2 - (a_3 u + b_3 v)^2 - r_3^2 = 0 \tag{5}$$

$$(a_4 - x)^2 + (b_4 - y)^2 + (c_4 - z)^2 - (a_4 u + b_4 v + c_4 w)^2 - r_4^2 = 0 \tag{6}$$

$$xu + yv + zw = 0 \tag{7}$$

$$u^2 + v^2 + w^2 - 1 = 0 \tag{8}$$

The three equations yielded respectively by (4) − (3), (5) − (3), (6) − (3) can be solved simultaneously, with the results having the following forms.

$$x = f_1(u)$$
$$y = f_2(u, v)$$
$$z = f_3(u, v, w)$$

Therefore, variables $x$, $y$, $z$ can be eliminated from equations (3), (7), (8) and the constraints are simplified to make the constraint system includes only three variables $u$, $v$, $w$.

### 2. Determination of domain $[L_l, L_u]$

Geometric elements, such as circles, points, lines and planes, are represented as follows in two-and three-dimension respectively (Table 1).

Geometry elements are classified as: (1) vector such as normal $(n_x, n_y, n_z)$; (2) coordinate, distance and radii. For class (1), the variables $(n_x, n_y, n_z)$ satisfy equation $n_x^2 + n_y^2 + n_z^2 = 1$, so the variation range of variables is $[L_l, L_u] = [-1, 1]$. For class (2), the range

**Table 1 Representation of geometry elements in 2D and 3D**

| | Two-dimension | Three-dimension | Notes |
|---|---|---|---|
| Point | $(x_i, y_i)$ | $(x_i, y_i, z_i)$ | |
| Line | $(n_x, n_y, d)$ | $(p_i, t_i)$ | $(n_x, n_y)$ is the unit normal of line, $d$ is the distance of the line from the origin. $p_i$ is the point on the line closest to the origin and $t_i$ is the unit tangent of line. |
| Circle | $(x_i, y_i, r)$ | $(x_i, y_i, z_i, r)$ | $r$ is the radii of circle. |
| Plane | $(n_x, n_y, n_z, d)$ | | $(n_x, n_y, n_z)$ is the unit normal of plane and $d$ is the distance from the origin to the plane. |

can be estimated conservatively as follows.

Given the input distance constraint or radii $d_i$, $i = 0, 1 \cdots n$ and the input angle constraint $a_i$, $i = 0, 1 \cdots n$.

$$L = \sum_{0}^{n} d_i$$

We can calculate the equivalent length caused by the input angle $a_i$ (excluding $\pi/2, 3\pi/2$) as follows.

$L' = L/\cos a_i$, $L = L + |L'|$. Therefore, the gross variable range for class (2) is $[L_l, L_u] = [-L, L]$.

## ANALYSIS OF EXAMPLES

Compared with the Newton-Raphson method, genetic simulated annealing algorithm has the following outstandingly superior features:

SAGA has the advantages of its intrinsic implicit parallelism and robustness. Our experimental system demonstrates this. As SAGA searches the optimal solutions globally in an initial population and is not concerned about matrix inversion, it is not sensitive to the initial values and can get all the possible solutions. When the constraint system is converted to optimization problem, the number of the equations $m$ is not required to equal that of variables $n$, which makes the over- and under-constraint problems handled easily.

We have developed a system Tsketch that is an experimental system implemented in C++ under the Win 2000 and VC6.0 environment. Tsketch supports four types of geometric objects: points, lines, circles and arcs. The constraints in Tsketch include distance between two points, distance between a point and a line, distance between two parallel lines, angle between two lines, tangency between a circle and a line, parallelism, perpendicularity, incidence, symmetry, equation of angles and distances.

### 1. Under-constraint

Engineering drawings are usually under constrained, especially in their early design stages. It is quite inconvenient to require the user to draw the diagram with accurate dimensions at the beginning. Even for some finished engineering drawings, under-constrained cases can still occur, because the user always ignores some unimportant dimensions. Most constraint solving algorithm can judge whether the constraint system is under-constrained or not, but cannot continue to present a solution. In contrast to that, SAGA can try its best to give a solution to meet the designer's intent.

Fig. 2a shows a well-constrained triangle. Then, a circle is added to the sketch and constraints are added which make the circle tangent to two sides of the triangle respectively: AC, AB. Now this configuration is under-constrained because the circle has three freedoms but is only subject to two constraints. Fig. 2b is the result after regenerating. From the result we can see that the position of the under-constrained circle is still recomputed and repositioned to satisfy the two tangent constraints. At the same time the triangle ABC remains unchanged. Therefore, the result can satisfy the designer's intent to a great degree.

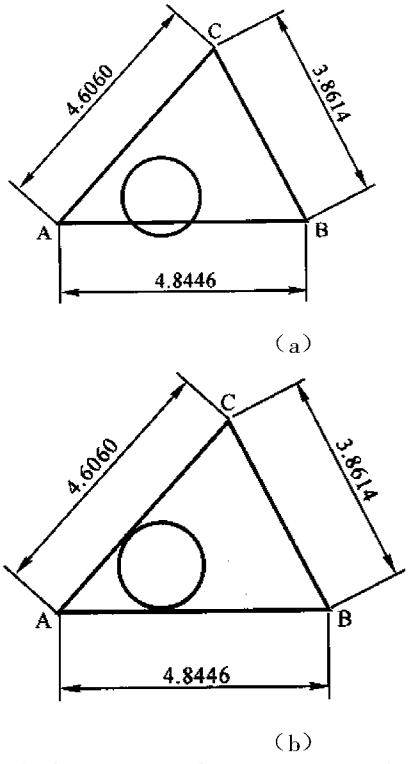


**Fig.2 Example of an under-constrained configuration**
(a) before being regenerated; (b) after being regenerated

### 2. Over-constraint

Sometimes engineering drawings are over-constrained. One case is the close dimension

chain because of the additional constraints added unintentionally (Fig.3). This always occurs especially when the rough draft has too many dimensions. Obviously this case does not conflict with the user's intent. So the constraint solver is required to handle it.

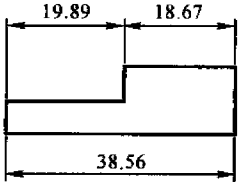


**Fig.3 The schematic plan of close dimension chain**

The other case is how to select a wanted solution among all the solutions for well-constrained system. Many solvers select the most approximate one by comparing the results with the initial configuration (David *et al.*, 2001). Of course, it is correct and reasonable only if the user designs and sketches strictly enough. However, the over-constraining technique can be used as a tool to aid the user to draw some complicated figures conveniently or to exclude some unwanted solutions. Fig.4 demonstrates such usage in drawing a regular pentagon. Fig.4a is the initial configuration of the diagram with two points A and B fixed. After specifying that five sides of the pentagon are equal and three diagonals BE, AC and CE are equal, the diagram is well-constrained. However, the diagram becomes the one in Fig.4b against our intent. In fact eight solutions are obtained from Tsketch and the one (Fig.4b) that is most similar to the initial configuration is selected. Next we continue our work by specifying that side AD equals side BE in Fig.4b. Obviously it becomes an over-constrained problem although we can still get the desired regular pentagon in Fig.4c.

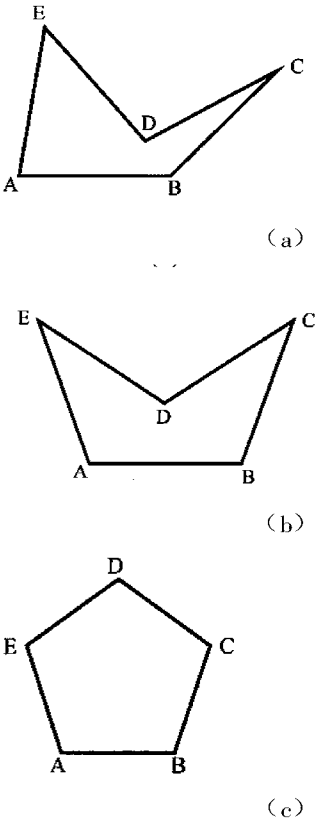


**Fig.4 Example of an over-constraint configuration**
(a) initial sketch; (b) configuration before constraint AD = BE is added in; (c) configuration after constraint AD = BE is added in

## 3. Well-constrained

SAGA integrates the guiding global searching ability of genetic algorithm with the local fast converging ability of simulated annealing algorithm well, which improves this algorithm's efficiency greatly. In consideration of the special characters of constraint problems, we take many specific measures mentioned above to optimize computation. Fig.5 shows a sectional drawing of a connecting rod. We compare the results between the case in which the searching space is compressed and that not compressed as in Table 2.

**Table 2 Some parameters under two conditions**

| | Number of equations | Number of variables | Running time (s) | Termination generations | Average fitness of the last generation $\left(\left\|\sum_{i=1}^{len} Fit(X_i)\right\|/len\right)$ |
|---|---|---|---|---|---|
| Not compressed | 41 | 41 | 1.32 | 95 | 0.99977 |
| Compressed | 29 | 29 | 0.83 | 82 | 0.99997 |

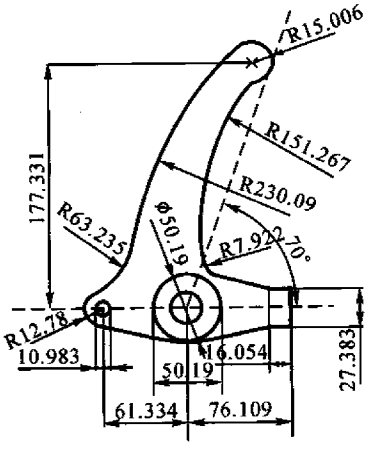


**Fig.5 Cross section of a connecting rod**

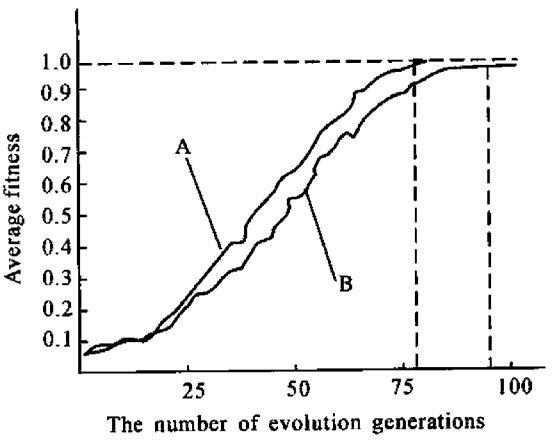


**Fig.6 Evolution process and running results**

Fig.6 shows how the evolution process proceeds. Curve A is the compressed case and curve B not the compressed. As can be seen from Fig.6, the two lines both rise slowly during the early 25 generations because the adjustive parameter of fitness $\beta$ is assigned a large value 1.25. After twenty-five generations, $\beta$ is assigned a small value 0.85, so the two lines rise steeply. However curve A is different from curve B in that the rate of grade of curve A is larger than that of curve B since the searching space of curve A is diminished, which improves the computation greatly.

## CONCLUSIONS

Genetic simulated annealing algorithm itself has many merits, such as implicit parallelism, stability of numerical computation and the global searching ability together with the local fast converging ability, etc. To the best of our knowledge, there are no literature reports regarding application of SAGA in constraint solving. This paper takes the special characters of constraint solving into consideration and introduces SAGA into constraint solving. Our experiment showed the validity of SAGA in constraint solving.

Multiple solutions cases often occurred in spatial constraint solving. Durand *et al*. (2000) presented a systematic framework that introduces homotopy method. Hoffmann *et al*. (1995a; 1995b) solves them by geometric constructing methods. Their common aspect is to solve the template problem, so they are difficult to be extended. SAGA is only concerned about the constraint equations and can be applied widely and can get all the possible solutions from the initial solutions. Therefore, it is natural that SAGA can solve the multiple solution problems.

Nevertheless, there is also plenty of room for further improvement. In particular, when dealing with geometric problems in higher dimensions, we need to improve the computational efficiency. It is also difficult to find an accurate searching space.

## ACKNOWLEDGMENT

I am very grateful doctor OU-YANG Yin-Xiu and doctor LIN Jun-Chen who help me correct the grammar errors in this paper.

## References

Aldefeld, B., 1988. Variation of geometries based on a geometric-reasoning method. *CAD*, **20**(3):117 – 126.

Borning, A.H., 1981. The programming language aspects of ThingLab, a constraint oriented simulation laboratory. *ACM TOPLAS*, **3**(4):353 – 387.

Bose, N.K., 1985. Multidimensional Systems Theory. D. Reidel Publishing Co., p.184 – 232.

Bouma, W., Fudos, I., Hoffmann, C.M., Cai, J. and Paige, J., 1994. A geometric contraint solver. *CAD*, **27**:487 – 501.

Bruderlin, B., 1990. Symbolic Computer Geometry for Computer Aided Geometric Design. *In*: Advances in Design and Manufacturing Systems, NSF conference, AZ, USA.

Davis, L., 1991. Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York.

David, P. and Borut, Z., 2001. A Geometric Constraint Solver With Decomposable Constraint Set. Skala, V.

Eds., The 9th International Conference in Central Europe on Computer Graphics and Visualisation' 01 - WSCG'01, Plzen, Czech Republic, University of West Bohemia, **2**:222 – 229.

De Jong, K., 1975, An Analysis of the Behavior of a Class of Genetic Adaptive Systems. PhD Dissertation, University of Michigan. Dissertation Abstract International, 36(10), 5140B.(University Microlms No.76 – 9381).

Dennis, J. E. and Schnabel, R. B., 1983. Numerical Methods for Unconstrainted Optimization and Nonlinear Equations. Prentice-Hall Inc., New Jersey.

Durand, C., 1998. Symbolic and Numerical Techniques for Constraint Solving. PhD thesis, Department of Computer Sciences, Purdue University, IN, USA.

Durand, C. and Hoffmann, C.M., 1999. Variational Constraints in 3D. Proc. Intl Conf on shape Modeling and Appl, Aizu, Japan, p.90 – 97.

Durand, C. and Hoffmann, C.M., 2000. A systematic framework for solving geometric constraint analytically. *J. of Symbolic Computing*, **30**:483 – 520.

Emiris, I.Z. and Verschelde, J., 1997. How to count efficiently all affine roots of a polynomial system. *Discrete Applied Mathematics*, **93**(1).

Fudos, I., 1993a. Editable Representations for 2D Geometric Design, Master's Thesis, Purdue University, Dept. of Computer Science.

Fudos, I. and Hoffmann, C.M., 1993b. Correctness proof of a geometric constraint solver. *Intl. J. Comp Geometry and Applic*, **6**: 405 – 420.

Fudos, I. and Hoffmann, C.M., 1997. A graph-constructive approach to solving systems of geometric constraints. *ACM Transactions on Graphics*, **16**:179 – 216.

Ge, J.X., Gao, X.S. and Chou, S.C., 2000. Geometric constraint satifaction using optimization methods. *Computer Aided Design*, **31**:867 – 879.

Goldberg, D.E., 1989. Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley Publishing Company, New York, p.1 – 88.

Hoffmann, C.M. and Vermeer, P., 1995a. Geometric Constraint Solving in $R^2$ and $R^3$. *In*: Computing in Eulidean Geometry Second Edition. World Scientific Publishing, Singapore, p.266 – 298.

Hoffmann, C.M. and Vermeer, P., 1995b. A Spatial Constraint Problem. *In*: Computational Kenematics's 95, J. P. Merlet and B. Ravani, Eds., Kluwer Academic Publ., p.83 – 92.

Hoffmann, C. M. and Joan-arinyo, R., 1997. Symbolic constraints in constructive geometry. *Journal of Symbolic Computation*, **23**:287 – 300.

Hoffmann, C. M. and Yuan, B., 2000. On Spatial Constraint Solving Approaches. Proc. ADC 2000, Springer verlag, ETH Zurich.

Ivan, S. E., 1963. Sketchpad: A Man-Machine Graphical Communication System. *In*: Proceedings-Spring Joint Computer Conference, Michigan, **23**:329 – 346.

Joan-Arinyo, R. and Soto, A., 1997a. A correct rule-based geometric constraint solver. *Computer and Graphics*, **21**(5):599 – 609.

Joan-Arinyo, R. and Soto, A., 1997b. A Ruler-and-Compass Geometric Constraint Solver. *In*: M.J., Pratt, R. D., Sriram and M.J., Wozny, editors, Product Modeling for Computer Integrated Design and Manufacture, Chapman and Hall, London, p.384 – 393.

Kondo, K., 1992. Algebraic method for manipulation of dimensional relationships in geometric models. *CAD*, **24**(3):141 – 147.

Kirkpatrick, S., Gelatt Jr., C. D. and Vecchi, M. P., 1983. Optimization by simulated annealing. *Science*, **220**:671 – 680.

Lazard, D., 1999. On theories of triangular sets. *Journal of Symbolic Computation*, **28**:105 – 124.

Michalewicz, Z., Krawczyk, J., Kazemi, M. and Janikow, C., 1990. Genetic Algorithms and Optimal Control Problem. *In*: Proc. of 29th IEEE Conf. On Decision and Control, p.1664 – 1666.

Morgan, A., 1992. Polynomail Continuation and its Relatoinship to the Symbolic Reduction of Polynomial Systems. *In*: B.Donald, D.Kapur, and J.Mundy, editors, Symbolic and Numerical Computation for Artificial Inteligence. Academic Press.

Song, P., Tang, M. and Dong, J.X., 2000. A Constraint Solving Approach Based on Graph Decomposition. China Graph'2000, Hangzhou.

Sturmfels, B., 1997. Introduction to Resultants. Lecture Notes at the AMS Short Course on Applications of Computational Algebraic Geometry, San Diego.

Verroust, A., Schonek, F. and Roller, D., 1992. Rule-oriented method for parametrized computer-aided design. *Computer Aided Design*, **24**(10):531 – 540.

Wang, D., 1998. Decomposing polynomial systems into simple systems. *Journal of Symbolic Computation*, **25**: 295 – 314.

Wu, W.T., 1986. Principles of mechanical theorem proving. *J. of Automated Reasoning*, **2**:221 – 252.