ELSEVIER

Computer-Aided Design 31 (1999) 867–879

COMPUTER-AIDED DESIGN

www.elsevier.com/locate/cad

Geometric constraint satisfaction using optimization methods

Jian-Xin Ge[1], Shang-Ching Chou*, Xiao-Shan Gao[2]

Computer Science Department, Wichita State University, Wichita, KS 67260-0083, USA

Received 9 July 1999; received in revised form 21 August 1999; accepted 17 September 1999

Abstract

The numerical approach to solving geometric constraint problems is indispensable for building a practical CAD system. The most commonly-used numerical method is the Newton–Raphson method. It is fast, but has the instability problem: the method requires good initial values. To overcome this problem, recently the homotopy method has been proposed and experimented with. According to the report, the homotopy method generally works much better in terms of stability. In this paper we use the numerical optimization method to deal with the geometric constraint solving problem. The experimental results based on our implementation of the method show that this method is also much less sensitive to the initial value. Further, a distinctive advantage of the method is that under- and over-constrained problems can be handled naturally and efficiently. We also give many instructive examples to illustrate the above advantages. © Published by Elsevier Science Ltd.

Keywords: Parametric design; Optimization method; Variational geometry

1. Introduction

There are several approaches to solving the geometric constraint problem. The *symbolic approach* [7,14,21] translates geometric constraints into a system of polynomial equations and solves the system by computer algebra techniques such as the Wu–Ritt method or the Grobner basis method [8]. It is reliable and complete, but is too slow and space-consuming to solve practical problems.

The *propagation approach* [1,6,9,15,22,23,26,28,39,41] solves the constraint system by deriving unknown variables or geometric objects from already known ones using a set of predefined rules. Usually the propagation methods are implemented with expert systems or logic programming languages such as Prolog.

The *graph analysis approach* [5,12,13,25,35,37,40] translates a geometric constraint problem into a graph and finds the geometric construction sequence by analyzing the graph. Both the propagation approach and the graph analysis approach have their limitation in scope.

On the other hand, the *numerical approach* [2,4,19,24,27,29,31,36] is a general method for solving the geometric constraint problem. Like the symbolic method, the numerical method first translates the constraints into a system of nonlinear equations. Then this equation system is solved by iterative methods instead of exact symbolic computation.

The most commonly used method in the numerical approach is the Newton–Raphson method. It is fast, but has the instability problem: the method is sensitive to the initial values. A small deviation in the initial value can lead to an unexpected or unwanted solution, or to the iteration divergence. To overcome this problem, recently the homotopy method has been proposed and experimented with [24]. According to the report in Ref. [24], generally the homotopy method works much better in terms of stability. These two methods generally require the number of variables to be the same as the number of equations. If these two numbers are different, i.e. the constraint system is generally over- or under-constrained, some special techniques, such as linear least square and singular value decomposition of a matrix, are required.

In this paper, based on the optimization method we use a numerical method for solving geometric constraint problems. Our experiments with this method show that it is also quite stable. Further, the method can *naturally* deal with under- and over-constrained problems. We also give many instructive examples to illustrate the above advantages. Numerical approaches similar to the optimization method have been introduced and discussed in Refs. [2,4,19]. We will discuss the related work in Section 2.11.

* Corresponding author.

E-mail addresses: ge@cs.twsu.edu (J.-X. Ge); chou@cs.twsu.edu (S.-C. Chou); gao@cs.twsu.edu (X.-S. Gao)

[1] On leave from Zhejiang University, Hangzhou, 310027, People's Republic of China.

[2] On leave from Institute of Systems Sciences, Academia Sinica, Beijing 100080, People's Republic of China.

0010-4485/00/$ - see front matter © Published by Elsevier Science Ltd.
PII: S0010-4485(99)00074-3

2. The optimization method for solving geometric constraint problems

2.1. Using optimization method for solving a system of equations

Generally, a geometric constraint problem can be first translated into a system of equations:

$$\begin{aligned} &f_1(x_1,\ldots,x_n)=0\\ &f_2(x_1,\ldots,x_n)=0\\ &\qquad\ldots\\ &f_m(x_1,\ldots,x_n)=0 \end{aligned} \tag{1}$$

Then the problem is how to solve this system of equations $F(X)=0$, where $F=(f_1,f_2,\ldots,f_m)^{\mathrm{T}}:\mathscr{R}^n\rightarrow\mathscr{R}^m$ is the equation vector and $X=(x_1,x_2,\ldots,x_n)^{\mathrm{T}}$ is the vector of unknown variables. This system of equations can be solved iteratively by the Newton–Raphson method [29]. The iteration formula is $X_{k+1}=X_k-J(X_k)^{-1}F(X_k)$, where $J(X_k)$ is the Jacobi matrix of $F(X)$ at point X_k.

The Newton–Raphson method usually requires that the number of constraints and the number of variables are the same so that the inverse of the Jacobi matrix can be calculated. This requirement makes the method difficult to handle under- and over-constrained problems which frequently occur in real applications.

Unlike the other numerical methods, the optimization approach solves the system of equations $F(X)$ by converting it into finding X at which the sum of squares

$$\sigma(X)=\sum_{i=1}^{m} f_i(X)^2 \tag{2}$$

is minimal. It is obvious that $F(X)=0$ has a *real* solution X^* if and only if min $\sigma(X)$ is 0. The problem of solving a system of equations is thus converted into the problem of finding the minimum of a real multi-variate function. The problem now can be solved by various well-developed numerical optimization methods [10,32,33].

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 for this approach it is natural to deal with under- and over-constrained problems.

In this paper, we focus on the numerical aspects of the algorithm to solve the geometric constraint solving problem by the optimization method. We have tested two optimization methods: the modified Levenberg–Marquardt method and the BFGS method. Next we will briefly introduce these two methods.

2.2. The modified Levenberg–Marquardt method

By the optimality condition we have the fact that the derivatives $g(X)$ of $\sigma(X)$ at the minimum points equal zero. Using the chain rule of derivation we have

$$g(X)=2J(X)^{\mathrm{T}}F(X) \tag{3}$$

where $J(X)$ is the Jacobi matrix of the function vector $F(X)=(f_1,f_2,\ldots,f_m)$. Applying the Newton–Raphson iteration formula to Eq. (3) we have the following iteration formula

$$(J_k^{\mathrm{T}}J_k+S_k)\Delta X_k=-J_k^{\mathrm{T}}F_k \tag{4}$$

$$X_{k+1}=X_k+\Delta X_k$$

where $S(X)=\sum_{i=1}^{m} f_i(X)\nabla^2 f_i(X)$ is a function of second-order derivatives. If we ignore the second-order derivatives we have the Gauss–Newton iteration formula

$$\Delta X_k=-(J_k^{\mathrm{T}}J_k)^{-1}J_k^{\mathrm{T}}F_k \tag{5}$$

$$X_{k+1}=X_k+\Delta X_k$$

To ensure that $\sigma(X)$ decreases with each iteration and also to deal with the singularity of matrix $J_k^{\mathrm{T}}J_k$, the Levenberg–Marquardt method is usually used with the modified iteration formula

$$\Delta X_k=-(J_k^{\mathrm{T}}J_k+\lambda_k I)^{-1}J_k^{\mathrm{T}}F_k \tag{6}$$

$$X_{k+1}=X_k+\Delta X_k$$

where I is the unit matrix and λ_k is a small real number.

The selection of λ_k should ensure that the matrix $(J_k^{\mathrm{T}}J_k+\lambda_k I)$ is positive definite. In practice the selection of λ_k has a great impact on the convergence domain and speed. Usually λ_k is initially set to 0.01 and later is doubled if $(J_k^{\mathrm{T}}J_k+\lambda_k I)$ is not positive definite. This method is a locally convergence optimization method and its behavior in our experiments is not satisfactory either in convergence speed and domain. The main problem is that the initial guess of the solution should be very near the solution to ensure that matrix $J_k^{\mathrm{T}}J_k$ is positive definite. This problem becomes more serious when processing under-constrained problems, since matrix $J_k^{\mathrm{T}}J_k$ always singular in this case. This makes us resort to the second optimization method to solve the problem.

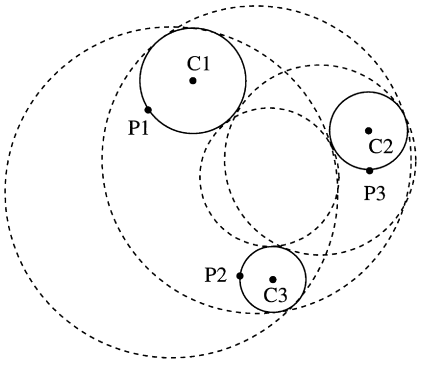
2.3. The BFGS method

The second method we have tried is the BFGS method which is also called the secant or quasi-Newton method [3,10,33]. It is a globally convergent method and thus is a more robust numerical optimization method. This method tries to construct a secant approximation H of the Hessian matrix of function $\sigma(X)$. The main algorithm of BFGS is described as follows:

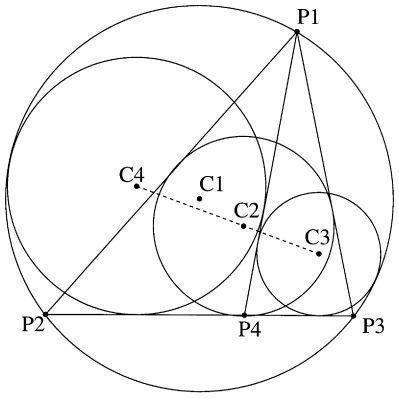
1. Initialization

   $H_0=$ a unit matrix; $X_0=$ the initial value; $k=0$;

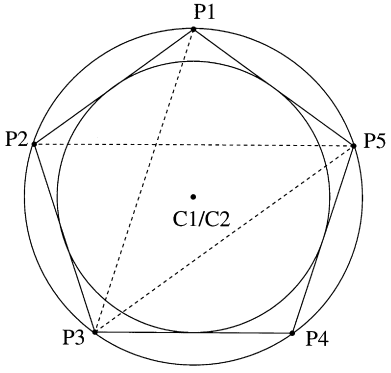
2. Compute the derivatives $g(X)$ and their difference y_k at X_k



(a)



(b)



(c)

Fig. 1. Some simple examples.

$g_k = \nabla\sigma(X_k)$; if $\|g_k\| < \varepsilon$ then X^*

$= X_k$ and stop the iteration

$y_k = g_k - g_{k-1}$;

3. Compute search direction p_k and the step to reach a minimum along the direction p_k

$p_k = -H_k g_k$

$\lambda_k = \min_\lambda \sigma(X_k + \lambda p_k)$

$\Delta X_k = \lambda_k p_k$

$X_{k+1} = X_k + \Delta X_k$

4. Adjust matrix H

$$H_{k+1} = H_k + \left(1 + \frac{y_k^{\mathrm{T}} H_k y_k}{\Delta X_k^{\mathrm{T}} y_k}\right)\frac{\Delta X_k \Delta X_k^{\mathrm{T}}}{\Delta X_k^{\mathrm{T}} y_k} - \frac{H_k y_k \Delta X_k^{\mathrm{T}} + \Delta X_k y_k^{\mathrm{T}} H_k}{\Delta X_k^{\mathrm{T}} y_k}$$

5. $k + 1$ goto (2)

More details of this algorithm can be found in Refs. [3,10,33]. There are several ways to improve this algorithm. One improvement is to use the trust region technique in this algorithm to promote convergence from poor starting point guesses [16]. Another improvement is to use a different updating formula to avoid storing the secant matrix H to save memory [34].

Compared with the Newton–Raphson method for geometric constraint solving, the optimization method has the following desirable features:

- This method is quite stable as demonstrated by our experimental results. The success of finding a desired solution by this method is rather insensitive to the initial guess of the solution to the geometric constraint solving problem. Also the solution found by this method is more predictable in contrast to the drastic solution jumping caused by even small changes of the initial value in the Newton–Raphson method.
- Under- and over-constrained problems are naturally handled by this method. In particular, for under-constrained problems, this method will find a "visually less changed" solution which is reasonably near the initially sketched diagram. For over-constrained but consistent problems, this method will generally still find a solution. This feature will be shown in Section 2.5.

The concept of "visually less changed" is not rigorously defined. However, we can give a formal definition to a related concept. A solution X is *visually least changed* if X is a solution to a geometric constraint problem and $\|X - X_0\|_2$ is minimal, where X_0 is the initial guess of the solution. From our experiments, we have found that the optimization method often finds a solution near the visually least changed solution for under-constrained problems.

In the next section we will show some experimental results of this method.

2.4. Experimental results with the BFGS method

We have developed a system AGP (Associative

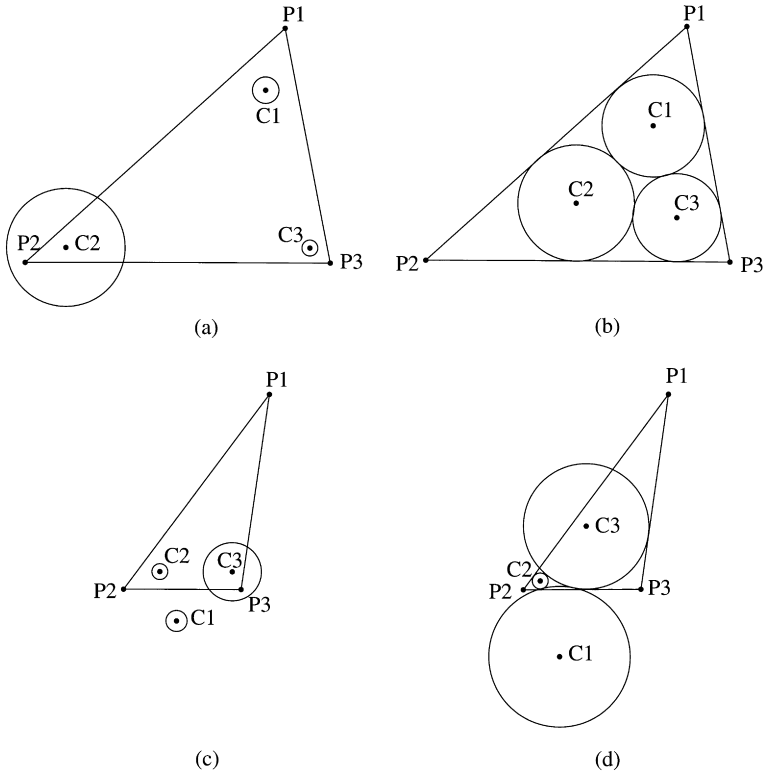


Fig. 2. The example to demonstrate the insensitivity to the inexact initial guesses of the solutions: (a) and (c) are the initial diagrams constructed by the user; (b) and (d) are the diagrams after computation.

Geometric Pad) using the BFGS method.[3] It is an experimental sketching system implemented in C++ under the Linux and X/Motif environment.

In AGP, four types of geometric objects are supported: points, lines, circles and arcs. However, only points are internally maintained on which constraints are specified. This is made possible by introducing some auxiliary points under some circumstances. For example when specifying that two circles with centers C_1 and C_2 are tangent, a new auxiliary tangent point T is introduced and the tangent constraint is internally converted into three constraints: (collinear; $C_1\,T\,C_2$), (onCircle $T\,C_1$) and (onCircle $T\,C_2$). Finally constraints in the form of (onCircle $P_i\,C$) $i = 1,\ldots$ are converted into (equalDistance $P_iC\,P_{i+1}C$) $i = 1,\ldots,k-1$.

In this section we show some examples solved by AGP to demonstrate some features of the optimization method.

Fig. 1 shows some simple examples made by AGP. Fig. 1a is one of the *Apollonius construction problems*. The problem is to construct a circle tangent to three given circles which are specified to have fixed centers C_i $(i = 1, 2, 3)$ and to pass through fixed points P_i $(i = 1, 2, 3)$, respectively. Actually the problem has eight solutions. In the figure AGP generated four essentially different circles (in dashed lines) corresponding the four initially roughly sketched circles.

Fig. 1b is the diagram of the Thebault–Taylor theorem proposed as a conjecture in 1938 and proved in 1983. The theorem states that given a triangle $P_1\,P_2\,P_3$ and a point P_4 on P_2P_3; circle C_1 is the circumscribed circle of the triangle; circle C_2 is the inscribed circle of the triangle; circles C_3 and C_4 are tangent to line $P_2\,P_3$, line $P_1\,P_4$, and circle C_1, then the centers C_2, C_3 and C_4 are collinear. The construction of this diagram is well constrained, but it has 256 solutions.

In Fig. 1c the regular pentagon is specified such that all its five sides are equal and three of its diagonals in dashed lines are equal; the circumscribed circle is specified to pass through five vertices of the pentagon; and the inscribed circle is specified to be tangent to the five sides of the pentagon. Note that this problem is over-constrained. Table 1 shows the running statistics of these three examples.[4]

Since the BFGS method is a globally convergence numerical optimization method, the success of finding a solution is not very sensitive to the initial value. This preferable feature is demonstrated in Fig. 2 where the three circles are specified to be mutually tangent and to be tangent to two neighboring sides of a triangle whose three vertices are specified

[3] This software can be accessed at ftp://henry.cs.twsu.edu/pub/agp/agp.tgz.

[4] All the running time in this paper is collected by averaging ten consecutive execution time on a Pentium Pro 200 machine with the Linux operating system.

Table 1
Running statistics for Fig. 1

	# Equations	# Variables	Time (s)
Fig. 1a	44	44	0.623
Fig. 1b	34	34	0.172
Fig. 1c	28	24	0.142

to be fixed. Fig. 2a and c are the initial diagrams sketched by the user. It is easy to see that the differences between the initial guesses and the exact solutions of this problem, respectively, in Fig. 2b and d are rather large. Also this figure demonstrates how different initial values lead to different branches of the solutions.

Fig. 3 is the simplest case for the problem we call *the tangent packing problem*. The problem is to pack $n(n+1)/2$ circles (n rows of circles) tangent to adjacent circles and/or the adjacent neighboring sides of a given triangle. Fig. 3 is the case of $n = 6$, i.e. we need to pack 21 circles in the triangle. This difficult problem contains 174 variables (since we introduce auxiliary tangent points) and 6 linear equations and 168 quadratic equations which could not be block triangularized. Table 2 shows the running results for different n of this problem.[5]

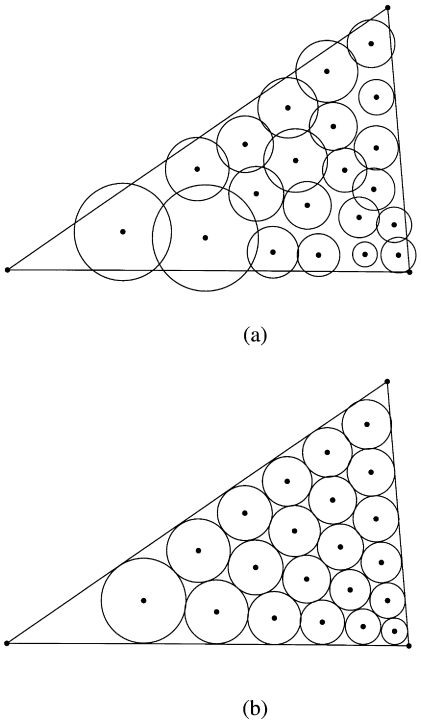


Fig. 3. A difficult problem. (a) The initial diagram drawn by the user. (b) Diagram generated after all tangent constraints are added.

2.5. Under- or over-constrained problems

Under- and over-constrained problems are very common in real applications. For example, 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 all the dimensions specified at the beginning. Even for some finished engineering drawings under-constrained cases can still occur, since some unimportant dimensions are ignored by the users. Over-constrained problems are not common in engineering drawings. 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.

The optimization method is capable of handling under- and over-constrained problems in a very natural way. The following examples demonstrate this capability. Fig. 4 shows the process to draw a regular 11-polygon. First the user sketches the diagram as in Fig. 4a. Then he/she specifies that the bottom side to be fixed and all the sides of the polygon are equal. At this moment, this problem is under-constrained since the polygon is not totally fixed. However, the diagram can still be computed satisfying all the constraints in the system as shown in Fig. 4b. Finally, the user specifies that all the diagonals connecting the two next adjacent vertices in the figure are equal. After computation we get the regular 11-polygon as shown in Fig. 4c. This actually becomes an over-constrained problem, since the user needs only to specify 8 of the 11 diagonals to be equal. Of course, this over-constrained problem is consistent, meaning that introducing redundant constraints will not conflict with the existing constraints. Effectively handling under- and over-constrained problems is a preferable feature of a geometric constraint solving system. In this problem, it is rather difficult to decide which eight diagonals should be selected to be equal. It is more convenient for the user to specify all the diagonals to be equal.

Sometimes over-constraining can be used to select a solution and exclude others from a finite set of solutions to a geometric constraint problem. The next example demonstrates such usage in drawing a regular pentagon. Fig. 5a is the initial configuration of the diagram with two points P_1 and P_2 fixed. After specifying that five sides of the pentagon are equal and three diagonals P_1P_3, P_2P_5 and P_3P_5 are equal, the diagram becomes the one in Fig. 5b which is a solution we do not want. Actually the problem is well constrained and there are eight solutions to this problem. Next we specify in Fig. 5c that points P_1 and P_2 are fixed; five

Table 2
Running statistics for Fig. 3 with different number of circles

# Circles (# rows)	# Equations	# Variables	Time (s)
3 (2)	30	30	0.228
6 (3)	54	54	0.965
10 (4)	86	86	3.379
15 (5)	126	126	11.587
21 (6)	174	174	23.751

[5] The case when $n = 4$ was given in Ref. [24] which inspires us to consider other cases.

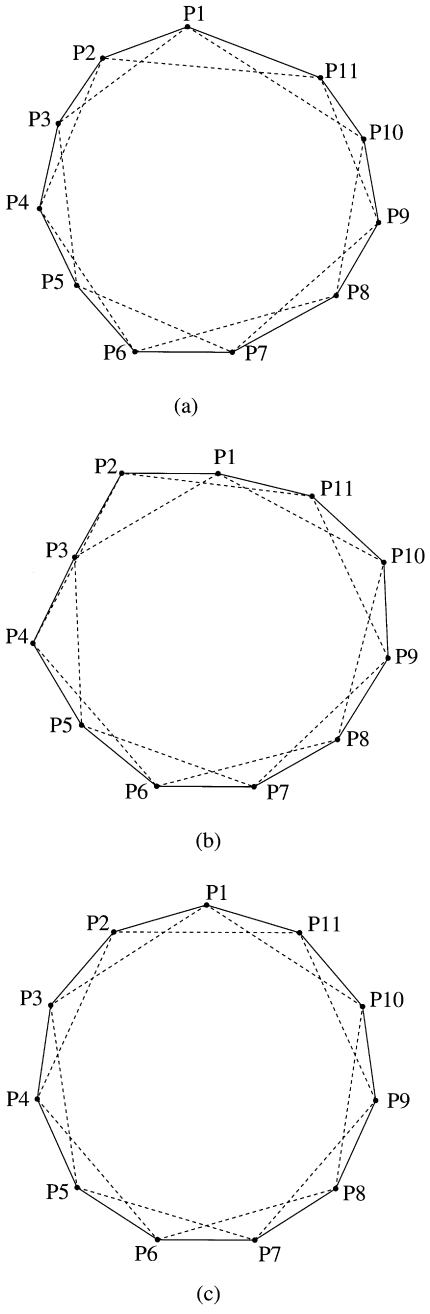


Fig. 4. Under- and over-constrained examples. (a) The initial diagram drawn by the user. (b) Diagram generated after specifying that the bottom side is fixed and the lengths of eleven sides are equal. (c) Diagram generated after specifying that all the diagonals in the figure are equal.

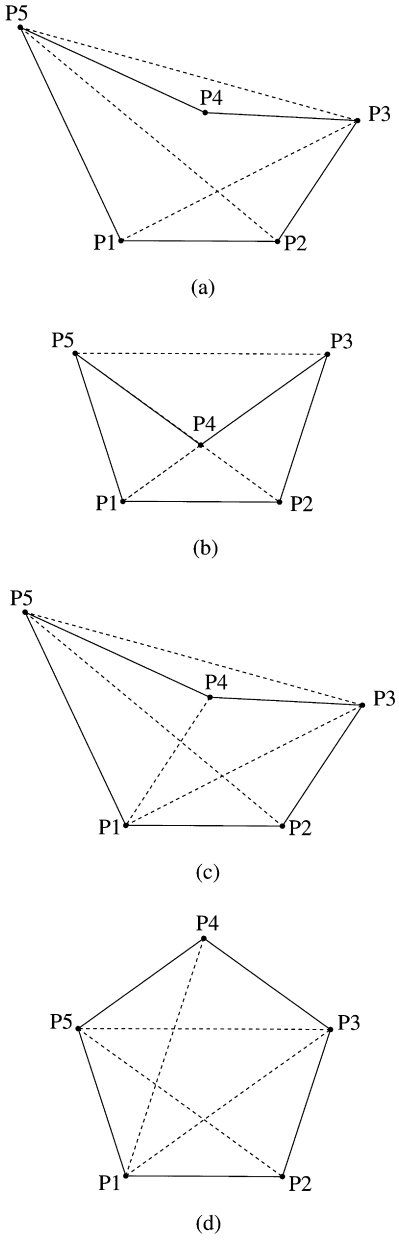


Fig. 5. The example to use redundant constraints to select different branches of solutions. (a) Initial diagram drawn by the user. (b) Diagram generated after specifying that five sides of the pentagon are equal and three diagonals are equal. (c) Initial diagram drawn by the user. (d) Diagram generated after specifying that five sides of the pentagon are equal and four diagonals are equal.

sides of the pentagon are equal; four diagonals P_1P_3, P_2P_5, P_3P_5 and P_1P_4 are equal. Obviously it becomes an over-constrained problem. After computation we get the desired regular pentagon in Fig. 5d.

In the general case under-, well-, and over-constrained problems could be very complicated. It could happen that a seemingly well-constrained diagram actually has infinite solutions. For example, to specify a regular pentagon we give seven constraints: all the five sides of the pentagon are equal and the four diagonals in Fig. 5c are equal. These constraints are independent in the sense that each polynomial corresponding to one constraint is not in the radical ideal of the polynomials corresponding to the other constraints. Since the problem has five points and thus needs seven independent constraints, it seems to be well constrained. But the problem actually has infinite solutions since the length of each side of the pentagon is not fixed. This phenomenon occurs for problems with many branches of solutions. Some of the constraints are used to eliminate branches, but not to reduce the degree of freedom of the

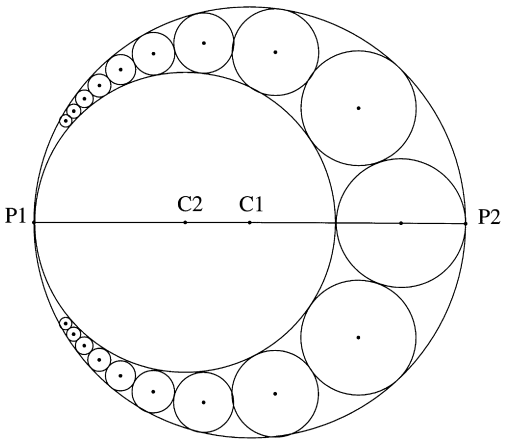


Fig. 6. A sequence of Apollonius' construction problems.

diagram. Generally, this kind of problem cannot be solved with the propagation method or the graph analysis method. Obviously the ability of the optimization approach to handle such problems is an advantage over other methods.

2.6. Practical considerations

Our experiments with the optimization method show that this method is quite efficient for medium sized geometric constraint solving problems with less than one hundred equations. However, for larger problems the performance is not satisfactory. For example, the problem in Fig. 6 is to pack 19 tangent circles in the area separated by two given tangent circles C_1 and C_2. It contains more than 170 equations, and we solve the problem as a whole in nearly 39 s.

One solution is to use various storage saving techniques to improve the performance [20]. Among these methods we tested the method in Ref. [30]. The result is quite satisfactory. For the six tangent circle packing problem, the method takes only 0.84 s instead of 23.75 s (see Table 2). However, this method is less unstable that our original method equipped with the trust region technique. In any case this is a good starting point. On one side we could try improve the stability of those very efficient but less stable numerical methods by using some techniques, such as the trust region method and stable line search methods. On the other side, we could provide the system with two numerical solvers. These solver are selected so that one is efficient but less stable and the other is stable but less efficient. When solving a specific problem, the system first tries the efficient solver. If the efficient solver could not solve the problem, the stable solver is invoked.

The other solution is to use the decomposition techniques discussed below. Actually the diagram in Fig. 6 could be constructed sequentially in an order of the size of the circle to be constructed. The construction of each circle is a special case of the Apollonius construction problems.

For those kinds of problems we can use two decomposition techniques to improve the computation performance. One is called *equation oriented decomposition* technique which turns a system of equations into *block triangular form*. A system of equations is said to be in block triangular form if it can be divided into subsets of equations:

$$S_1(x_1,\ldots,x_{n_1}), S_2(x_1,\ldots,x_{n_1+n_2}),\ldots, S_t(x_1,\ldots,x_{n_1+\cdots+n_t}).$$

where n_i is the number of equations in S_i. Such a system of equations can be solved sequentially. Each time S_i is solved by the optimization method and the solution is substituted into subsequent sub-systems S_k ($k > i$). The block triangularization process can be implemented by the algorithms proposed in Refs. [25,37].

Another technique is called *cluster oriented decomposition* which solves the constraint problems by first dividing the diagram into a set of small sub-diagrams called clusters and then assembling these clusters again. A cluster is essentially assumed to be a rigid body only having translational and rotational degrees of freedom. This implies that the inner-cluster constraints which only concern geometric objects in the same cluster are invariant with respect to the rigid body transformation, e.g. if an inner-constraint in a cluster is originally satisfied it will still be satisfied after the cluster is transformed by any translational and rotational transformation.

First suppose clusters $C_1,\ldots,C_p$ in the diagram are recognized and solved with some methods [1,5,13,35,41]. For each cluster C_i we introduce a transformation matrix T_i

$$T_i = \begin{pmatrix} a_i & -b_i & c \\ b_i & a_i & d_i \\ 0 & 0 & 1 \end{pmatrix}$$

where a_i, b_i, c_i and d_i are unknown variables satisfying $a_i^2 + b_i^2 = 1$.

Like the 3D cases [27,36], we first translate the inter-cluster constraints, which constrain geometric objects in different clusters, into a system of equations with a_i, b_i, c_i and d_i ($i = 1,\ldots,p$) as unknown variables. For example, an inter-cluster constraint requiring that two clusters C_i and C_j share a common point P can be written as

$$T_i \begin{pmatrix} x_{pi} \\ y_{pi} \\ 1 \end{pmatrix} = T_j \begin{pmatrix} x_{pj} \\ y_{pj} \\ 1 \end{pmatrix}$$

which essentially represents two linear equations with elements in T_i and T_j as unknown variables, where (x_{pi}, y_{pi}) and (x_{pj}, y_{pj}) are the numerical coordinate values for point P in clusters C_i and C_j, respectively. The equations obtained in this way, called assembly equations, are then solved by the optimization method. The solution to the system of assembly equations, if it exists, ensures that the inter-clusters will be satisfied after the transformations are performed on the clusters. Since the inner-cluster constraints are invariant with respect to the transformation, all the constraints in the system are now satisfied.

Generally, these two decomposition techniques will improve the performance of numerical calculation greatly. It will make the optimization method attractive in some real applications, such as engineering drawing systems in which nearly all problems can be solved sequentially. However, this decomposition technique will not be helpful if the structure of the problem is very "bad". The diagram in Fig. 3 is such an example. It is obvious that the system of equations for the problem cannot be structurally block-triangularized and the diagram itself also cannot be clustered.

2.7. Problem of inequalities

One of the problems of the optimization method is that the general numerical optimization methods only find local minima of a multi-variate function. It is possible for the function $\sigma(X)$ in Eq. (2) to have a nonzero local minimum, although there exists a solution X^* to Eq. (1). Furthermore, the general optimization method usually cannot properly handle the critical point (the point at which the gradient vector of the goal function is zero) which is not an minimal point. From our experience, this problem rarely happens for under- or well-constrained problems (actually, we have encountered no such cases so far). However, for over-constrained problems we did encounter such cases. In particular it becomes serious when the constraint problem involves inequalities.

For example, the diagram in Fig. 5d is generated from the sketch in Fig. 5c by specifying that $P_1P_3 = P_1P_4 = P_2P_5 = P_3P_5$. This is an over-constrained problem. However, if the initial sketch is Fig. 5b with the same constraints, then the optimization iteration will finally reach a critical point which is not a solution.

A natural remedy for this problem seems to use inequality constraints. For this example we tried to add an inequality constraint which specifies that point P_4 is above line P_3P_5 in Fig. 5b. In theory it should get the desired result. Each inequality $f(X) \geq 0$ can be transformed into an equality $f(X) - My^2 = 0$, where y is a newly introduced variable and M is a positive number. It is obvious that we have $\forall X(f(X) \geq 0 \Leftrightarrow \exists y\, f(X) - My^2 = 0)$. However, this attempt failed in our experiments. The reason is that the introduction of inequalities may lead to some critical points near the solution we want.

Here is a very simple example to demonstrate this situation. Suppose we have an equation $x^2 - 1 = 0$ and an inequality $x \geq 0$, following the above discussion we construct a function $f(x, y)$ to be minimized

$$f(x, y) = (x^2 - 1)^2 + M(x - y^2)^2$$

For $M = 1$ there are five critical points for this function:$(1, \pm 1)$, $(\pm\sqrt{2}/2, 0)$ and (0,0) (see Fig. 7a–c). But only the first two critical points are the minimum points we want. The other critical points are essentially saddle points which are not local minima. The introduction of these saddle points make the problem especially difficult. The success rate of our numerical solver for this problem is only 33.3% for the initial values of $x, y \in [-5, 5)$ (see Fig. 7d). Although we can choose $M \geq 2$ so that the critical points $(\pm\sqrt{2}/2.0)$ are eliminated, the critical point (0,0) cannot be removed.

2.8. Constraint hierarchy

The constraint hierarchy can be introduced to geometric constraint problems. In mechanical designs, some constraints, especially those related to structural configuration or performance of the part, are of the highest priority and should be satisfied unconditionally. Others, such as those for esthetical purposes, are of lower priority and are not necessarily to be satisfied. Freeman-Benson [11] solves constraint hierarchy problems with an efficient local propagation method. However, his method cannot handle geometric constraint problems.

We solve this problem with multi-objective optimization methods. As in Ref. [11] we partition all the constraints in the system into a constraint hierarchy with different priority levels. We assume that constraints in lower constraint hierarchy levels have higher priority, and level zero has the highest priority. Following the way in Section 2.1, for each constraint hierarchy level i we could construct an objective function $G_i(X)$

$$G_i(X) = \sum_{j=0}^{m_i} \mu_{ij} f_{ij}^2(X)$$

where μ_{ij} can be used to specify the relative importance of each constraint in the same hierarchy level.

Now the constraint problem has been transformed into a classic nonlinear multi-objective unconstrained optimization problem

$$\min\{G_0(X), G_1(X), \ldots, G_k(X)\}.$$

Several numerical methods can be used to solve the above problem [33,38]. We convert this problem into a constrained optimization problem. For simplicity, we only consider two levels of constraints. Constraints in level zero are compulsory and constraints in level one are preferable. First, we solve the problem in level zero by solving following minimization problem

$$G_0^* = \min_{x \in \chi^n} G_0(X).$$

Since the constraints in this level are mandatory, we must check that G_0^* is less than a prescribed small number ε_0. If this condition is satisfied then we proceed to the next level of constraints. One natural condition to satisfy this level of constraint is that no constraints in the previous level (level zero) are violated. So we turn this problem into a constrained optimization problem with the following goal function

$$G_1^* = \min_{x \in \mathscr{D}} G_1(X)$$

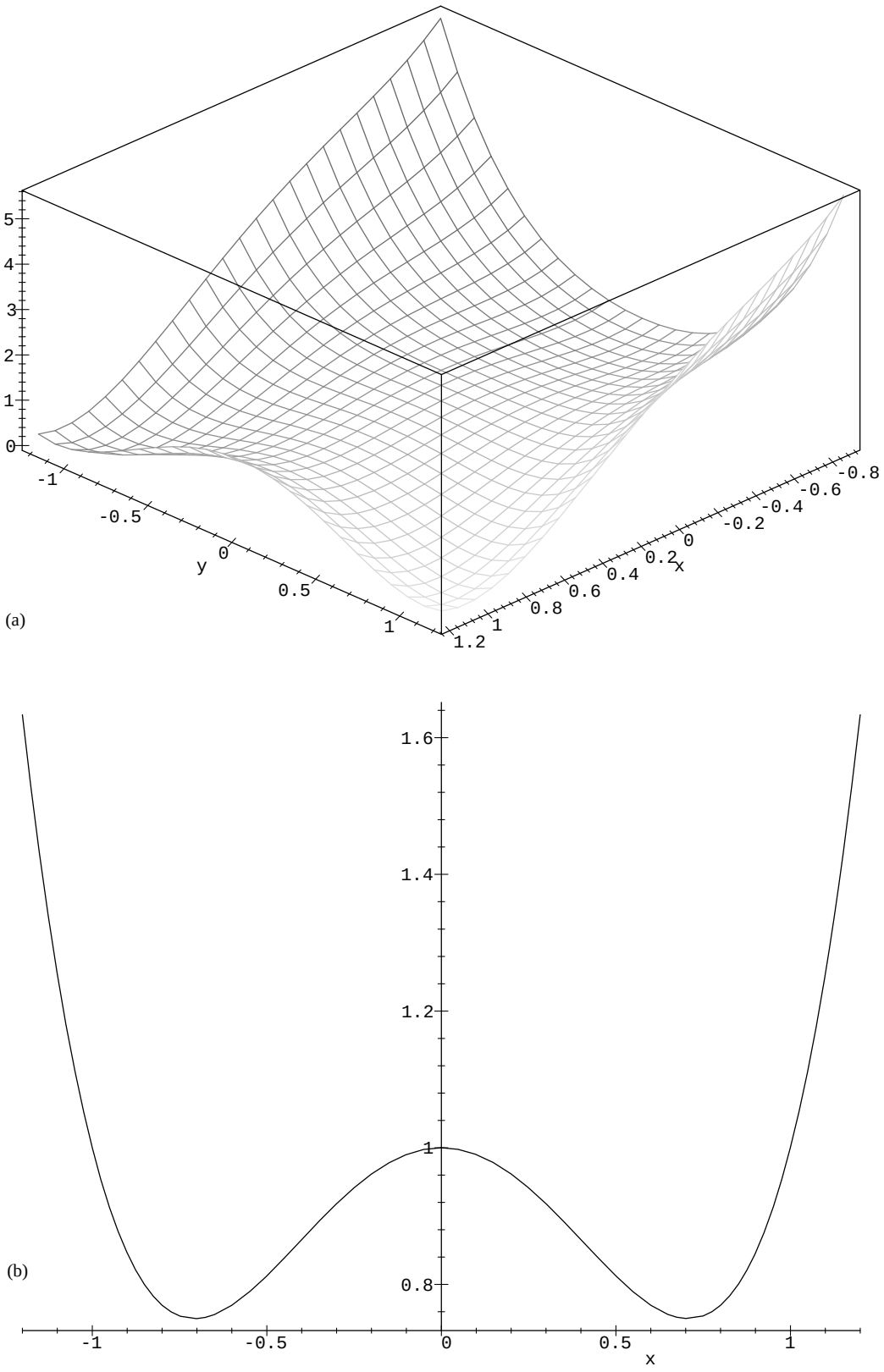
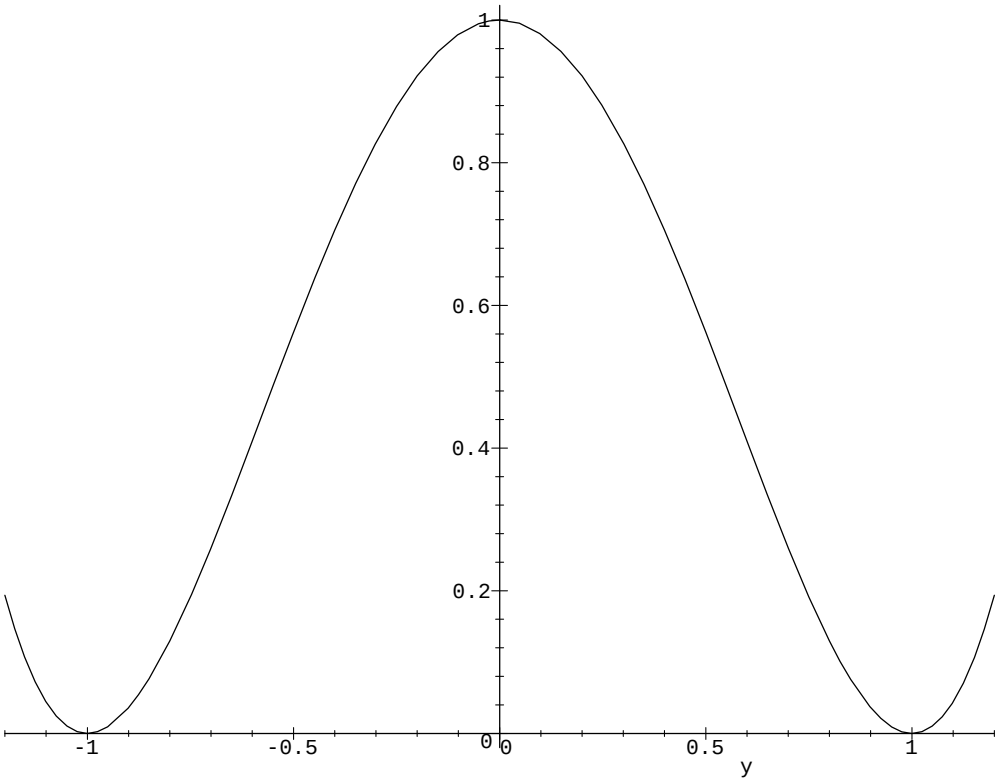


Fig. 7. Problem of unwanted critical point. (a) The graph of function $f(x, y)$ in interval $(-0.2, 1.2) \times (-1.2, 1.2)$. (b) The graph of the function for $y = 0$ with x in interval $(-1.2,1.2)$. (c) The graph of the function for $x = 1$ with y in interval $(-1.2,1.2)$. (d) The convergence diagram for $x, y \in [-5, 5)$. The black area consists of the set of the initial points which will lead the numerical solver to the solutions $(1, \pm 1)$.

(c)



(d)

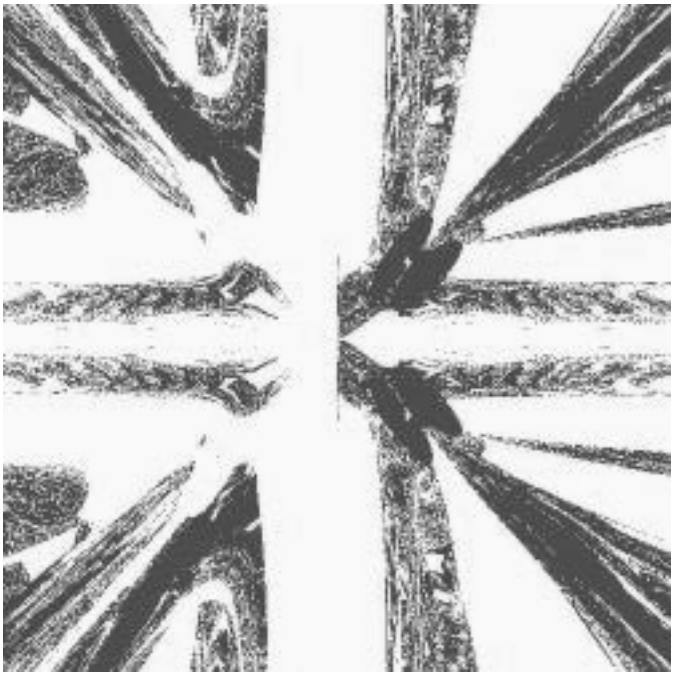


Fig. 7. (*continued*)

where $\mathscr{D} = \{X|(G_0(X) \leq G_0^* + \varepsilon_1\}$ and ε_1 is a small real number.

Using current general nonlinear constrained optimization techniques [3,10,33], it is not difficult to find a solution for the above problem. Since constraints in this level are not necessarily to be satisfied, G_1^* does not have to be zero.

It is worth pointing out that this technique can be easily adapted to find a visually least changed solution discussed in Section 2.3.

2.9. Linear constraints

In some applications, such as the user interface design and the architectural layout design, only linear constraints are involved. In this case, we can model the geometric constraint problems with much simpler optimization models. One such model translates the constraints into a linear programming problem

$$\min_{\mathbf{x}} \mathbf{c}^{\mathrm{T}}\mathbf{x}$$

$$\text{s.t. } \mathbf{A}\mathbf{x} = 0\ \mathbf{B}\mathbf{x} \geq 0$$

where the conditions correspond to the geometric constraints in the system. For this problem, we can simply set the goal function to be a constant. This problem can be solved by commonly used linear programming algorithms [18,22]. The other model is the quadratic programming model where the goal function is the sum of the squares of linear functions induced by linear geometric constraints, thus could be written as

$$\min_{\mathbf{x}} \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}$$

$$\text{s.t. } \mathbf{B}\mathbf{x} \geq 0.$$

This problem can be solved by a very efficient algorithm called successive quadratic programming (SQP) approach [3,33]. We believe that these methods can solve these specific problems much more efficiently and robustly.

2.10. Global optimization

The limitation of the optimization method discussed in Section 2.7 can be overcome with the global optimization method. The global optimization method can be used to determine global minima for a goal function. It is obvious that if there exists a solution X to Eq. (1), the goal function $\sigma(X)$ in Eq. (2) will always reach its global minimal value of zero at a point which is assumed to be found by the global optimization method. The global optimization will solve the problem by assuring that the minimization process will not trap into a local minimal point or an unwanted critical point.

Besides, the global optimization method can be used to find as many solutions to a set of equations as possible by enumerating local minima of the goal function in Eq. (2).

The global optimization method is a current research topic in the numerical optimization. Many global optimization methods have been developed, including the deterministic method, the stochastic method, the simulated annealing method, the interval method, and the genetic algorithm [17,32,33]. Some of these, especially the deterministic method, are so well developed that they could be used to solve many difficult problems in real applications.

2.11. Comparison to some related work

Numerical approaches similar to the optimization method have been introduced and discussed in Refs. [2,4,19]. While these papers focused on the mathematical model, the main purpose of our paper is to address the numerical experiment with the optimization method. For the numerical method to solve the geometric constraint problems, one of the most challenging problem is how to find a numerical method which is both efficient and robust.

In Ref. [4] the authors convert the constraints to an energy function and solve the constraint problem by minimizing the energy function. However, the steepest descent method used in their work suffers from slow convergence because of the zigzagging problem [3], which makes it hard to be used to solve real application problems. In Refs. [2,19] the authors independently proposed a method to convert the constraint problem into a constrained optimization problem. However, both papers gives no detailed information about the numerical behavior of the algorithm.

Also the distinctive advantage of the optimization method in handling with under- and over-constrained problems is one of the main focuses of our paper as our experiments with various regular n-polygons have shown (Section 2.5). We have made extensive experiments with under- and over-constrained problems and have found that our approach is very natural to deal with these problems.

Furthermore, we observe that with this method redundant constraints can be naturally used in practice to select a solution branch from a finite set of solutions of a well constrained problem (Section 2.5).

3. Conclusions

In this paper we use the optimization method to solve geometric constraint problems. Many experimental results show that this method is stable and effective in solving difficult geometric constraint problems. In particular, it can be used to solve under- and over-constrained problems naturally. Further more, this method can be extended to a general frame work to cover more general geometric constraint problems in many applications such as CAD.

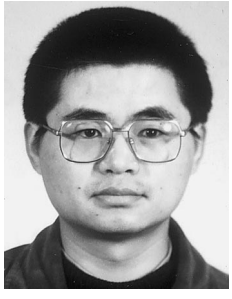
Acknowledgements

This work was supported in part by the NSF Grants CCR-9420857 and CCR-9901062 and was accomplished at Wichita State University. Ge and Gao were also supported in part by the Chinese National Science Foundation. The

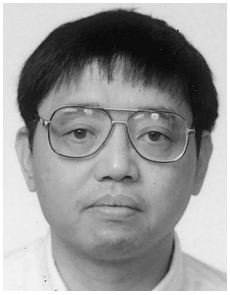
authors wish to thank the reviewers and the editor for their helpful suggestions and advice.

References

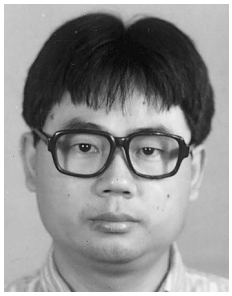
[1] Aldefeld B. Variation of geometries based on a geometric-reasoning method. Computer Aided Design 1988;20(3):117–26.
[2] Barford LA. A graphical, language-based editor for generic solid models represented by constraints, PhD dissertation, Department of Computer Science, Cornell University, 1987.
[3] Bazarra MS, Sherali HD, Shetty CM. Nonlinear programming: theory and algorithms, 2. New York: Wiley, 1993.
[4] Witkin A, Fleischer K, Barr A. Constraints on parametrized models. Computer Graphics 1987;21(4):225–32.
[5] Bouma W, Hoffmann CM, Fudos I, Cai J, Paige R. A geometric constraint solver. Computer Aided Design 1995;27(6):487–501.
[6] Bruderlin B. Rule-based geometric modeling, PhD dissertation, Swiss Federal Institute of Technology (ETH) Zürich, 1988.
[7] Buchanan SA, de Pennington A. Constraint definition system: a computer algebra based approach to solving geometric problems. Computer Aided Design 1993;25(12):740–50.
[8] Chou S-C. Mechanical geometry theorem proving, Dordrecht: Kluwer, 1987.
[9] Chou S-C, Gao X-S, Zhang J-Z. A fixpoint approach to automated geometry theorem proving, WSUCS-95-2, Computer Science Department, Wichita State University, 1995.
[10] Elster KH, editor. Modern mathematical methods of optimization Berlin: Akademie, 1993.
[11] Freeman-Benson B, Maloney J. An incremental constraint solver. Communications of the ACM 1990;33(1):54–63.
[12] Fudos I, Hoffmann CM. Correctness proof of a geometric constraint solver, Technical Report CSD93-076, Department of Computer Science of Purdue University, 1993.
[13] Fudos I, Hoffmann CM. A graph-constructive approach to solving systems of geometric constraints. ACM Transactions on Graphics 1997;16(2):179–216.
[14] Gao X-S, Chou S-C. Solving geometric constraint problems. II. A symbolic approach and decision of rc-constructibility. Computer Aided Design 1998;30(2):115–22.
[15] Gao X-S, Chou S-C. Solving geometric constraint problems, I. A global propagation approach. Computer Aided Design 1998;30(1): 47–54.
[16] Gay DM. Algorithm 611—subroutines for unconstrained minimization using a model/trust-regin approach. ACM Transactions on Mathematical Software 1983;9:503–24.
[17] Horst R, Pardalos PM, Thai NV. Introduction to global optimization, Dordrecht: Kluwer, 1995.
[18] Ignizio JP. Linear programming, Englewood Cliffs, NJ: Prentice-Hall, 1994.
[19] Kalra D, Barr A. Constraint-based figure-maker. Eurographics'90 1990:413–24.
[20] Kelley CT. Iterative methods for optimization, Department of Mathematics, North Carolina State University, http://www4.ncsu.edu/eos/users/c/ctkelley/www/darts.html, to be published by SIAM in 1999, Draft of May 4, 1998.
[21] Kondo K. Algebraic method for manipulation of dimensional relationships in geometric models. Geometric Aided Design 1992;24(3):141–7.
[22] Kramer G. Solving geometric constraint systems, Cambridge, MA: MIT Press, 1992.
[23] Kramer G. Geometric constraint engine. Artificial Intelligence 1992;58:327–60.
[24] Lamure H, Michelucci D. Solving geometric constraints by homotopy. IEEE Transactions on Visualization and Computer Graphics 1996;2(1):28–34.
[25] Latheam RS, Middleditch AE. Connectivity analysis: a tool for processing geometric constraints. Computer Aided Design 1994;28(11):917–28.
[26] Lee JY, Kim K. Geometric reasoning for knowledge-based parametric design using graph representation. Computer Aided Design 1996;28(10):831–41.
[27] Lee K, Andrews G. Inference of the positions of components in an assembly: part 2. Computer Aided Design 1985;17(1):20–4.
[28] Leler W. Constraint programming languages, Reading, MA: Addison-Wesley, 1988.
[29] Light R, Gossard D. Modification of geometric models through variational geometry. Geometric Aided Design 1982;14:208–14.
[30] Lin DC, Nocedal J. On the limited memory bfgs method for large scale optimization. Mathematical Programming 1989;45:503–628.
[31] Lin VC, Gossard DC, Light RA. Variational geometry in computer-aided design. Computer Graphics 1981;15(3):171–7.
[32] More JJ, Wright SJ. Optimization software guide, SIAM, 1993.
[33] Nemhauser GL, Rinnooy Kan AHG, Todd MJ, editors. Optimization Amsterdam: Elsevier, 1989.
[34] Nocedal J. Updating quasi-Newton matrices with limited storage, Mathematics of Computation, 1980;35, 773–82.
[35] Owen J. Algebraic solution for geometry from dimensional constraints, ACM Symp. Found. of Solid Modeling, Austin TX, 1991, p. 397–407.
[36] Rocheleau DN, Lee K. System for interactive assembly modeling. Computer Aided Design 1987;19(1):65–72.
[37] Serrano D, Gossard D. Constraint management in MCAE. In: Gero J, editor. Artificial intelligence in engineering: design, Amsterdam: Elsevier, 1988.
[38] Schnierderjans MJ. Goal programming: methodology and applications, Dordrecht: Kluwer, 1995.
[39] Suzuki H, Ando H, Kimura F. Geometric constraints and reasoning for geometrical CAD systems. Computer and Graphics 1990;14(2): 211–24.
[40] Todd P. A *k*-tree generalization that characterizes consistency of dimensioned engineering drawings. SIAM Journal of Discussions in Mathematics 1989;2:255–61.
[41] Verroust A, Schonek F, Roller D. Rule-oriented method for parametrized computer-aided design. Computer Aided Design 1992;24(3): 531–40.



Jianxin Ge is an associate professor in Applied Mathematics Department at Zhejiang University, China, He received his BS in 1988, MS in 1990, and Ph.D. in 1993, all in Computer Science and Engineering at Zhejiang University. His research interests are in the computer aided design (CAD) and include 3D surface and solid modeling, computer graphics, geometric constraint satisfaction, and numerical computation.



Shang-Ching Chou, currently a Professor at the CS Department of Wichita State University, received a Ph.D. at University of Texas at Austin in 1985. Since then he has been supported by NSF for 15 consecutive years for his research in automated geometric reasoning.



Xiaoshan Gao was born in Hebei province of China in 1963. He got his Ph.D. from the Chinese Academy of Sciences in 1988. His research interests include automated reasoning, symbolic computation, computer graphics and intelligence CAD, and computer aided education. He has published one monograph and more than 40 research papers. He has won a first class award in natural sciences and an Outstanding Young Scientist Award of 1997 from the Chinese Academy of Sciences. He is a recipient of the Excellent Youth Grant from the Chinese Academy of Sciences. He is a recipient of the Excellent Youth Grant from the Chinese NSF for 1998–2000.