Tracking method for reparametrized geometrical constraint systems

Rémi Imbach, Pascal Mathis and Pascal Schreck
LSIIT
UMR ULP-CNRS 7005
Pôle API, boulevard Sébastien Brant, F67400 ILLKIRCH

***Abstract*—In CAD, constraint solvers allow a user to describe a figure or an object with a set of constraints like distances, angles, tangencies, incidences and so on. Geometric solvers proceed in two stages. First, a symbolic construction plan is provided from the set of constraints. Then, the dimensions of constraints are used in a numerical stage to evaluate the construction plan.**

However, construction plans can not be easily provided for many problems in 3D. A classic idea consists in removing and adding some constraints in order to make the problem solvable by a geometric method. This leads to a numerical problem in which numerical values for the added constraints have to be computed in order to find the values of the added dimensions that validate the removed dimensions. Finding these values is usually done by sampling which is very time-consuming when there are more than 2 variables to sample. In this paper we address the numerical stage by adapting a path-tracking method. This allows to find several solutions and this method is efficient when the number of values is greater than 2.

I. Introduction

Geometrical constraint solving is used in many fields such as CAD, robotics and molecular modeling. The aim is to find coordinates of entities such as points, lines, circles, spheres and planes subject to some constraints involving for instance distances, angles, tangencies, incidences. Usually geometrical constraint systems (*GCS*) are well-constrained in the sense that there exists a finite number of solutions. In CAD, the constraints are provided by the user under the form of a dimensioned sketch. As the number of solutions can grow exponentially, the sketch is sometimes used for capturing the user's intent and to propose only solutions close to the sketch.

Different approaches can be used to address the issue of finding solutions of a GCS. There are the symbolic algebraic approach [1], [2], the numerical approach [3], [4], [5], the geometrical approach which was performed with rule-based systems [6], [7], [8], [9] or graph analysis [10], [11], [12], [13].

In CAD, a GCS often contains dozens of primitives. A *divide and conquer* strategy is used to decompose the problem into sub-problems [14], [15], [16], [17], [18]. There are several reasons to do that. First, sub-problems can be solved by different methods suited to the involved constraints. Next, usual solving methods are implemented by polynomial time algorithms with order three or four. With sub-problems containing less than 20 primitives, this complexity is no more prohibitive. Finally, a method that fails to solve the whole problem can manage to offer solutions for the sub-systems. It is often the case with geometrical methods. In this article, we will not discuss aspects of decomposition. The aim is to describe a solving method that can also be used as a method for solving sub-problems.

The locus intersection method (*LIM*) is a fast geometric method that performs two stages : first a symbolic construction plan is provided, then numerical dimensions given by a user are taken into account and the construction plan is evaluated to give numerical solutions. LIM provides all the solutions with quadratic time complexity. Nevertheless, only few problems can be solved with this method, especially in 3D.

An idea taken up by [19] is to remove some constraints and to add new ones in order to make the problem solvable by LIM. The new constraints, that correspond to distances or angles, are *driving parameters* and a numerical stage seeks for values of driving parameters to meet the removed constraints. We call *reparametrization* this way of switching constraints by others. In [19], it is assumed that the problem is decomposed into *basic configurations* involving up to 6 primitives. All possible basic configurations are solved by reparametrization with a maximum of two driving parameters. With up to two parameters, a numerical solving that samples ranges could be sufficient. In this paper we do not make this assumption and systems can be reparameterized with more than two parameters. Thus, the numerical phase can be very time consuming. If a reparametrization leads to remove and add n constraints, then the numerical solving will have to find points in a n-box. [20] adapts the Newton method to work with LIM but this method encounters difficulties in controlling the passage from a solution to another. So, we propose here to use a method of path-tracking that is adapted for symbolic geometric solutions.

The paper is structured as follows. Section II gives some definitions about geometric constraint systems. Section III presents the symbolic part of the solving while section IV shows the numerical solver that performs paths tracking. Section V concludes.

II. Geometrical Constraint Systems and constructions

A. GCS

A *Geometrical Constraint System* (*GCS*) is a triple (C, X, A) where C is a set of constraints, X a set of unknowns, and A a set of parameters. A constraint can be seen as a first-order

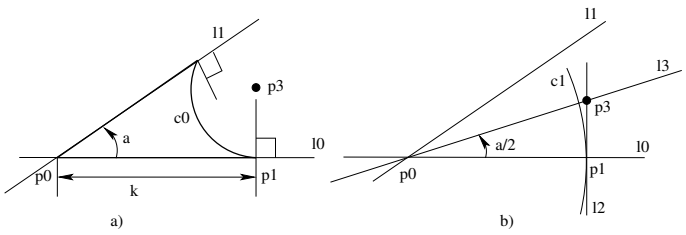

Fig. 1. Graphical representation of : a) a GCS, b) a construction plan

logic term based on the set of variables $X \cup A$. We will denote by $unk(C)$ the set of unknowns of constraints in C. Terms are built over a many-sorted signature. A signature for 2D GCS could be:

Signature
Sorts
Length, Angle, Point, Line, Circle
Functions
distance : Point Point → Length
angle : Line Line → Angle
bisector : Line Line → Line
intersection_cl : Circle Line → Point
...
Relations
on_line : Point Line
on_circle : Point Circle
radius_of : Point Circle
tangency : Circle Line Point
...

Figure 1a is a graphical representation of a simple problem. With the previous signature, the corresponding GCS is :

C = { distance(p0, p1) = k, angle(l1, l2) = a,
on_line(p0, l0), on_line(p1, l0),
on_line(p0, l1), on_line(p2, l1),
tangency(c0, l0, p1), tangency(c0, l1, p2)
radius_of(p3, c0) }
X = { p0, p1, p2, p3, l0, l1, c0 }
A = { k, a }

With the 2D Euclidean classical *interpretation*, (where usual functions and relations are assigned to symbols and where variables are elements of $\mathbb{R}^n$) figure 1a is also a graphic representation for a model of the previous GCS. A *solution* is a variable assignment that makes valid the constraints over the interpretation. In CAD, terms and variables are usually interpreted in 2D or 3D space with real values for coordinates. We make this assumption in the following. Notice that in dynamic geometry, it could be useful to consider other interpretations with complex or projective spaces [21]. In CAD, unknowns usually correspond to geometric *entities* and parameters correspond to *dimensions* (lengths, angles).

For a specific assignment of parameters, a GCS is said to be *well-constrained* if the set of all possible solutions is finite. It is *over-constrained* if there is no solution and *under-constrained* otherwise. In example of figure 1a, some values for parameters lead to an infinite set of solutions since each solution can be placed elsewhere in the euclidean plane. So in CAD, well/under/over-constrained has to be understood *apart from* rigid motions. In algebraic terms, the number of solutions is the number of orbits under the action of rigid motions group [18]. In example 1a, whatever the assignment for parameters, there is only one solution (one orbit) so the GCS is well-constrained as there is a finite number of orbits.

For any assignment, a GCS is said to be *generically* well-constrained if it is well-constrained for all possible assignments for parameters except for some specific ones that correspond to degenerate cases. The same goes to under and over constrained GCS. For instance, 4 points with 6 distances is generically over-constrained but there exists some values for the distances that give solutions. For these values the problem is then well-constrained.

Apart from specific frameworks like rigidity theory, determining whether a GCS is generically well-constrained is an undecidable problem. So in CAD, the weaker notion of *structurally* well-constrained is used. Intuitively, a system is structurally well-constrained if the system contains enough constraints for the number of unknowns and no part of the system is over-constrained. In 2D, a GCS is structurally well-constrained if it meets the characteristic of Laman [22]. Given an entity e (point, line, circle, etc.), *dof*(e) will denote the degree of freedom i.e. the number of coordinates. For a constraint c, $dor(c)$ will denote the degree of restriction of the constraint. It corresponds to the number of degrees of freedom the constraint removed. For a GCS interpreted in 2D, if we have

$$\sum_{x \in X} dof(x) - \sum_{c \in C} dor(c) = 3$$

and if for all subsets X' of X

$$\sum_{x \in X'} dof(x) - \sum_{c \in C} dor(c) \geq 3$$

then the GCS fulfills the characteristic of Laman and is structurally well-constrained. The constant 3 in these relations correspond to the number of degrees of freedom that have to be removed in order to choose a specific solution. In practice, in 2D, a point and the slope of an incident line are fixed. The Laman characteristic was not extended in 3D (see the famous counter example of the double banana [23]). And in CAD we simply impose the relationship

$$\sum_{p \in P} dof(p) - \sum_{c \in C} dor(c) = \begin{pmatrix} d+1 \\ 2 \end{pmatrix}$$

with d the dimension of space.

B. Construction plan

A *construction plan* is a list of terms where the i-th element has the form $x_i = f(x_1, ..., x_{i-1})$. It expresses the symbolic solution of a GCS. For the previous GCS, a construction plan can be :

p0 = fix_point(0, 0)
l0 = fix_line(p0, 0)
c1 = new_circle(p0, k)

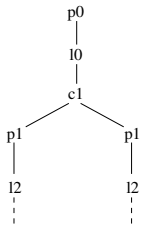

Fig. 2. Evaluation tree of a construction plan

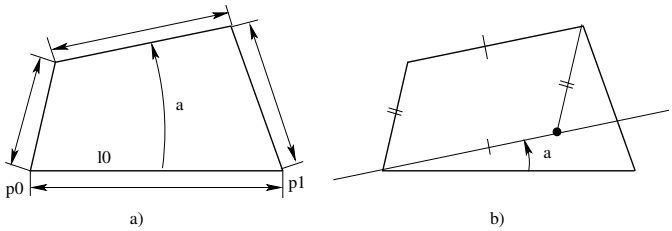

Fig. 3. Classical problem that LIM can not solve

p1 = intersection_cp(c1, l0)
l2 = perpendicular(l0, p1)
l1 = angle_lp(l0, p0)
l3 = bisector(l0, l1)
p3 = intersection_ll(l2, l3)
...

This piece of construction is illustrated figure 1b.

A geometrical solver provides a construction plan from a GCS. It can be implemented as a knowledge-based system where knowledge are rules of geometric construction. Once a construction plan is available, it is evaluated by replacing the parameters with numerical values defined by the user. The evaluation leads to a tree in which the number of leaves correspond to the number of solutions. Each branch from the root to a leaf represents a solution. Figure 2 shows the first levels of the tree for the plan above assuming that values for parameters do not correspond to a degenerate case (no intersection or congruent entities). This tree contains two leaves corresponding to two symmetric solutions. They come from the intersection of $c1$ and $l0$ that gives two solutions for $p1$.

III. Locus intersection method and reparametrization

A. Locus Intersection Method

The symbolic phase consists in moving from a set of terms describing a statement to a set of terms corresponding to a construction plan. With knowledge-based systems, this phase is based on a set of symbolic construction rules given by an expert.

The time complexity of geometrical solvers implementing knowledge-based systems is $O(n^3)$ or $O(n^4)$. But there exists a simple implementation of a geometrical solver that has a quadratic complexity.

The loci intersection method (called LIM as in [19] that gives a variant of the following algorithm) consists in translating constraints into loci. For instance, if there is a distance between two points $k = distance(p1, p2)$, then point $p2$ is on a circle with $p1$ as center and k as radius. If a circle is tangent to a line and to another circle, then its center is on a parabola. The entities (unknowns) are then defined by intersections of loci.

In 2D, the algorithm proceeds in the following way.

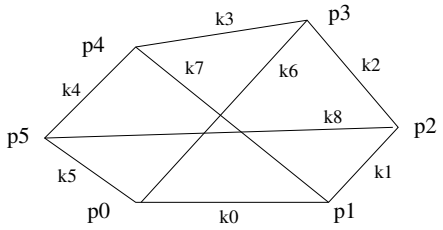

Fig. 4. $K_{3,3}$: 2d problem with 6 points p_i and 9 distances k_j

Algorithm LIM
Step 1 : all entities are tagged as non constructed
Step 2 : a point and an incident line are tagged as constructed
Step 3 : choose an entity x tagged as non constructed, build the set C of constraints such as x appears in each constraint of C and entities of $unk(C) - \{x\}$ are constructed. If $\sum_{c \in C} dor(c) = dof(x)$ then tag x as constructed. Actually, x can be determined by intersections of loci of constraints in C.
Step 4 : repeat step 3 until no more entities can be tagged.

If all entities are tagged, the solving is successful. Notice that the success of this algorithm could depend on the choice of fixed entities at step 2.

In step 3, this algorithm uses a rule of construction that translates a symbolic constraint in a set of terms that come to the current construction plan.

Many CAD problems are solved by decomposition of the constraint system into subsystems [17]. This loci method can solve most of the usual subsystems. Nevertheless, some small GCS are not resolved by this method. Figure 3a shows a GCS involving 4 distances and 1 angle constraints. A geometrical construction is outlined figure 3b. A knowledge-based system could provide a construction plan for this problem. However LIM can not successfully deal with this problem. After fixing point $p1$ and slope of $l0$, $p2$ is constructed and the process can not continue.

There is no known combinatorial characterization of solvable constraint systems by LIM. Although rather limited, this method is interesting because it can easily be adapted to solve more problems and even problems not solvable geometrically such as the one shown in figure 4 and known as $K_{3,3}$. In this figure, straight lines mean distance constraints.

B. Reparametrization

The reparametrization algorithm (called REPARAM) forces the GCS to be solvable with LIM. Whenever an entity does not share enough constraints with constructed entities, new constraints are added. Assuming that the GCS is structurally well-constrained, these new constraints make some entities over-constrained. So, during the process, some constraints are removed. For the sake of simplicity and without loss of generality, we consider that $dor(c) = 1$ for any constraint c. The reparametrization algorithm performs this way :

Algorithm REPARAM

Step 1 : Let $A = \emptyset$, the set of new functional constraints
let $R = \emptyset$, the set of removed functional constraints
Step 2 : all entities are tagged as non constructed
Step 3 : a point and an incident line are tagged as constructed
Step 4 : for an entity x tagged as non constructed, build the set C of contraints such as x appears in each constraint of C and entities of $unk(C) - \{x\}$ are constructed. Choose x such as
diff $= \sum_{c \in C} dor(c) -$ *dof*(x) is minimum.
if *diff* < 0 $A = A \cup$ NewConstraint($unk(C)$, x)
if *diff* > 0
$R = R -$ ChooseConstraintToRemove(C)
tag x as constructed.
Step 5 : repeat step 4 until no more entities can be tagged.

Algorithm `NewConstraint`(X, x) builds one new constraint involving x and elements of X. It must be a functional constraint (not incidence or tangency). It is either a distance constraint or an angle constraint according to the type of entities. If x is a line and X only contains one point, the new constraint will be point-line distance. These new dimensions, distance or angle, are *driving parameters*.

With the previous $K_{3,3}$ example, the algorithm proceeds as shown in figure 5. After constructing points $p0$ and $p1$, points $p2$ and $p5$ are linked to constructed points but one constraint is missing to define them. A constraint $k9 = distance(p0, p2)$ is added with $k9$ the driving parameter. Now, *diff* for points $p5$ is 0 and it can be constructed. Next, point $p4$ is tagged. For point 3, *diff* $= 3 - 2$, so one constraint must be removed. Procedure `ChooseConstraintToRemove` chooses arbitrarily the distance $p4p3$. The construction plan contains line-circle intersection for $p1$ and circle-circle intersections for points $p2, p3, p4$ and $p5$. So, the evaluation tree can have up to 32 branches. Finally all numerical values for $k9$ making constraint $k3 = distance(p3, p4)$ true, have to be found.

In this example, let S be a numerical solution for a particular assignment of $k1, ..., k8$. Let $v9$ be the value taken by $k9$ by reading distance $p0p2$ in S, then the interpretation of the construction plan yielded by REPARAM corresponding to this assignment for $k1, k2, k4 \ldots k8$, and $v9$ for $k9$ leads to a tree containing at least one branch corresponding to S. The other branches are not necessarily other solutions. And so is for all solutions of this system. Thus, the construction plan potentially allows to find all the solutions.

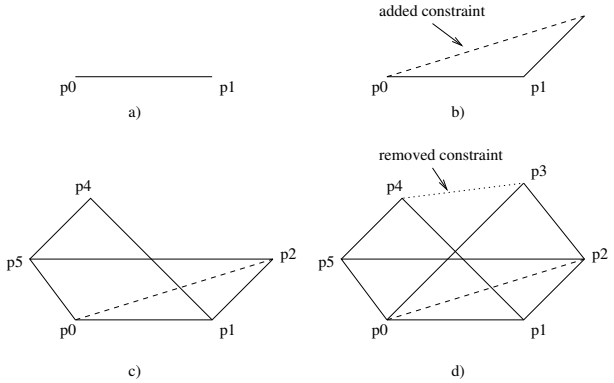

Fig. 5. Algorithm REPARAM on $K_{3,3}$

To find the solutions of initial GCS, we must find the numerical values of driving parameters that validate the constraints removed. To automatically determine the allowable ranges for a driving parameter the method presented in [24] is used. With only one driving parameter, sampling the ranges is efficient to find all the solutions. But if there are two or more parameters, the sampling is too time-consuming and we resort to a paths tracking method.

IV. Paths tracking

Once symbolic stage presented in III has been performed, giving a construction plan Cp depending on driving parameters, a numerical function is obtained as follows: let $k = (k_0, ..., k_{d-1})$ be the d-dimensional vector of the deleted constraint's parameter values. On a given branch, evaluating Cp with the vector $v = (v_0, ..., v_{d-1})$ of values for the driving parameters provides a figure (i.e. the values of the coordinates of geometric entities) that will be denoted by $Cp(v)$.

One can check on this figure the constraints that have been deleted in the symbolic stage. For instance, if the i-th constraint that has been removed is $k_i = distance(p0, p1)$, where $p0$ and $p1$ are two geometric entities, then the distance between $p0$ and $p1$ can be read on $Cp(v)$, giving a real number $f_i(Cp(v))$, where f_i is the functional term associated with the i-th deleted constraints, for $0 \leq i \leq d - 1$.

A numerical function $F : \mathbb{R}^d \rightarrow \mathbb{R}^d$ is now defined as: $F(v) = (F_0(v), ..., F_{d-1}(v))$, where $F_i(v) = f_i(Cp(v)) - k_i$. Figure 6 presents the graph of this function F for the $K_{3,3}$ problem depicted on figures 4 and 5, for all the branches of Cp. Recall that the added constraint is $k9 = distance(p0, p2)$ and that the removed constraint is $k3 = distance(p3, p4)$. With user values for $k1, ..., k8$ and with $k3 = 0.5$, the evaluation of Cp with $k9 = 1.0$ leads to two figures. On the first one, $f_O(Cp(v)) = 2.0$ i.e. $distance(p4, p3)$ on this figure. So the difference with the desired value is 1.5 (see the graph where values for $k9$ are on the abscissa). On the second figure, $f_O(Cp(v)) = 2.75$ and the difference with $k3$ is 2.25. There are only two numerical interpretations of Cp for $k9 = 1.0$ because many intersections failed during the evaluation of the construction plan.

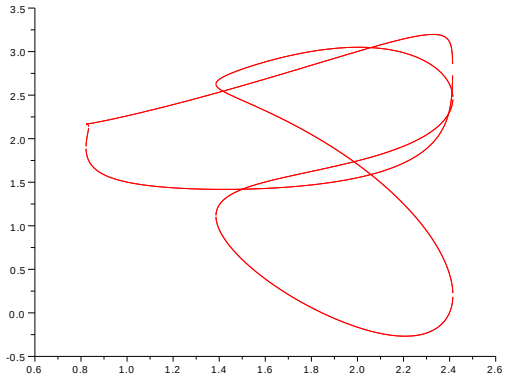

Fig. 6. The graph of the function F for $K_{3,3}$

Each branch carries a piece of a d-dimensional hypersurface, and the solutions of the problem are the zero-values of all these pieces.

The purpose of the numerical stage is the search of the d-dimensional vectors v_* satisfying $F(v_*) = 0$.

Considering that $Cp(v)$ is computed from a sequence of ruler and compass constructions, we do not have an analytic expression of F. There is a simple brute force method to approximate the zero-values of such a function in a given box of $\mathbb{R}^d$: sample the box accurately enough and evaluate F on each sample, and on each branch of the evaluation tree of Cp for a complete research of solutions. Observing that the number of branches of a construction plan produced by REPARAM grows exponentially with the number of locus intersections that have more than one solution, this method leads to a high computational cost when d and the depth of the evaluation tree are increasing. Another (classical) problem comes from the precision of the sampling which could make miss some solutions.

However, one can assume that on a given branch of the construction plan, F is almost everywhere infinitely differentiable: indeed, one can associate with each primitive produced by Cp an analytic expression in a Cartesian reference, and construct a system of equations such as the coordinates of the primitives are a solution of this system. These functions are infinitely derivable except on their poles, consisting in a finite set of parameter values, because they are combination of rational fractions and analytic functions. F consists then in evaluating some distances or angles on primitives, that are analytic functions. Then F is almost everywhere smooth, and zero-values of F can be found with usual numerical methods, such as Newton's method or homotopy, that avoids the extra computational cost of the sampling.

A. Finding first solution

Assuming that Cp is also parametrized by the parameters of the constraints that have to be satisfied, F can be seen as a parametrized function. Let h be the parameters of constraints of the GCS, and h_* the values for these parameters that have to be fulfilled. Let us note $F(h, v)$ the function F for the vector of parameters h. Assume now that the sketch given by the user is a solution of the GCS with another valuation of the parameters h, say h_{sk}, the ones that are satisfied by this sketch. So by reading values of driving parameters on it, the sketch provides a solution v_{sk} of $F(h_{sk}, v) = 0$.

Let us consider the $d+1$-dimensional function

$$H : [0,1] \times \mathbb{R}^d \to \mathbb{R}^d \text{ defined by}$$
$$H(t, v) = F((1-t)h_{sk} + th_*, v).$$

Notice that for $t = 0$, a solution of $H(0, v) = 0$ is known since the values v_{sk} of the driving parameters read on the sketch fulfill this system. So one branch must lead to the sketch. As shown in [25], that presents how to establish an homotopy in the parameter's space, $H(t, v) = 0$ consists in curves, called *homotopic paths*, climbing from the solutions $H(0, v) = 0$ to the solutions of $H(1, v) = 0$. These paths have to be followed from the known solutions to the guessed solutions. A survey on continuation and path-tracking methods can be found in [26], and IV-C is devoted to the path-tracking method that has been implemented.

Since only one solution of $H(0, v) = 0$ is known (v_{sk} that is provided by the sketch), we can obtain at most one solution v_* of $H(1, v) = 0$ using this method.

The following part explains how to obtain more solutions.

B. Finding other solutions

Consider now the system $\mathcal{F}$ defined by $F(h, v) = 0$ for a given set of parameters h, and let us define, for $0 \leq i \leq d-1$, its i-th partial system $\mathcal{F}^i$ by:

$$\begin{cases} F_0(v) = 0 \\ \dots \\ F_{i-1}(v) = 0 \\ F_{i+1}(v) = 0 \\ \dots \\ F_{d-1}(v) = 0 \end{cases}$$

where $v = (v_0, ..., v_{d-1})$.

The solution set of such a system of $d-1$ independent equations of d real variables is a curve, that is smooth almost everywhere since the functions F_i are analytic. However, this curve may have several connected components.

Considering a point of this curve is known (v_{sk} if $h = h_{sk}$ or v_* if $h = h_*$), this curve can thus be followed from this known point by a path-tracking method (see IV-C).

A sequence of points is thus obtained, and if one of this points, say v_c, satisfies $|F_i(v_c)| < \epsilon$, for a given $\epsilon \in \mathbb{R}$, v_c is close to a solution of $\mathcal{F}$. (v_c is to be corrected into a solution, for instance with Newton's method applied on the system $\mathcal{F}$).

This operation is performed for each system $\mathcal{F}^i$ and each found solution.

Such curves are presented in figures 10 and 12.

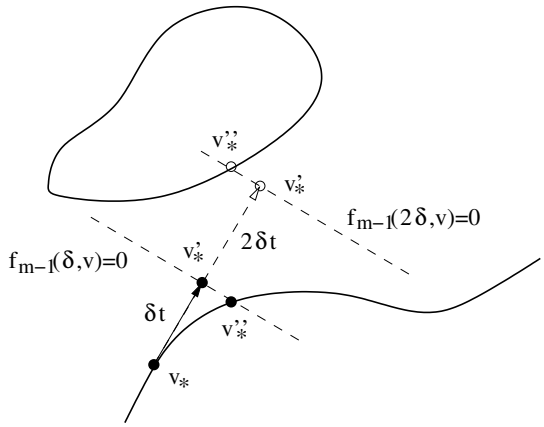

Fig. 7. Two components of a followed curve, the known point v_*, and the tangent t to the curve in v_*. Applying a too large step δ can lead the corrected point v''_* of the prediction v'_* to lie on another component of the curve. Normal lines to the prediction vectors are the solution set of the added equations $f_{m-1}(\delta, v) = 0$.

C. Curve Tracking

The path-tracking method that has been implemented in order to follow the curves defined in IV-A and IV-B is the so-called *predictor-corrector* method: knowing a point on the followed curve, translating it in the direction given by the tangent to the curve at this point, provides a prediction that is corrected into a point lying on the curve.

Let $f(v) = 0$ be a system of $m-1$ independent equations $f_i(v) = 0$ of m real variables $v = (v_0, ..., v_{m-1})$, and v_* a point satisfying $f(v_*) = 0$ (in other words, v_* is a point of the curve defined by $f(v) = 0$). The tangent $t = (t_0, ..., t_{m-1})$ of the curve in v_* is the solution of the equation $J(v_*)t = 0$ such as $max_{0 \leq i \leq m-1} t_i = 1$, where $J(v_*)$ is the Jacobian matrix of f in v_*. Since no analytical expression of f is known, $J(v_*)$ is computed in practice with finite differences.

Then a new point v'_* on the curve is predicted such as $v'_* = v_* + \delta t$, where δ is a real parameter.

In order to correct v'_* by the Newton's method, an additional equation $f_{m-1}(\delta, v)$ parametrized by δ that measures the progression along the curve has to be added to the system $f(v) = 0$. This new equation is defined as $f_m(\delta, v) = (v - v_* - \delta t)t^t$.

The Newton's method is then applied to the system $(f_0(v) = 0, ..., f_{m-2}(v) = 0, f_{m-1}(\delta, v) = 0)$ with initial vector v'_*, to obtain a new solution v''_*. (See figure 7).

The parameter δ can either be chosen small enough, hoping the Newton's iterations will converge, or be adapted (i.e. halved) each times they do not. Such a method is presented in [27].

[28] uses interval arithmetic in combination with this method to avoid the risk of jumping from one component of the curve to another. (See figure 7).

D. Branch Swapping

As said, the Cp provided by the symbolic stage has several branches: it means that during the evaluation, some choices have to be done. For instance, if two circles have two intersections, one of these intersections has to be selected in order to continue the evaluation. Only considering the function F and the systems $\mathcal{F}^i$ on one branch leads to obtain only pieces of the curves. Other pieces are obtained by swapping branches when the curve is no more defined on the current branch. Such cases happen when two geometric entities (for instance two circles), that have two intersections, move away from each other such as they have no more intersection. When these two entities have only one intersection, the new branch that has to be considered in order to stay on the same hypersurface, is the one that makes the movement of geometric entities to be continuous. See figure 8.

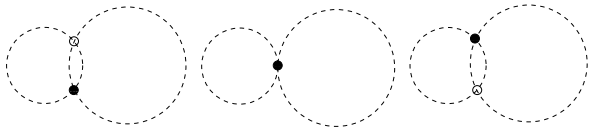

Fig. 8. Two circles constructed by a construction plan, and their intersections. In the left part, the considered intersection is the one marked by a filled point. As the circles are moving away from each other with the variations of driving parameters, the two intersection becomes one (at the middle). In the right part, as circles comes back closely, the other intersection is considered.

E. Numerical Results

The presented method has been implemented, and we will now expose its test on three geometric problems. Table 9 shows, for each of these problems, the time of computation (symbolic and numeric steps) of our method and the number of found solutions, and compares it with the naive sampling method (some properties of the problems are also given, as number of primitives, number of constraints, and the number of driving parameters).

The research of solutions has been restricted to one branch of the construction plan.

	Sampling	Tracking
$K_{3,3}$ 6 primitives 9 constraints 1 added/removed	2 solutions, $\simeq 0.02sec$	2 solutions, $\simeq 0.4sec$
Dodecagon 12 primitives 21 constraints 2 added/removed	2 solutions, $\simeq 10sec$	2 solutions, $\simeq 1.5sec$
Icosahedron 12 primitives 30 constraints 3 added/removed	1 solution, $\simeq 767sec$	2 solutions, $\simeq 17.4sec$

Fig. 9. Number of found solutions and execution times on a PC with processor at 2.2GHz for three geometric problems.

1) $K_{3,3}$: The problem illustrated in figure 4 involves six points in a plane and nine constraints of distance. During the

symbolic stage, one constraint is deleted and one is added providing a problem solvable by LIM.

The system to solve (see IV-A) consists thus in one equation in one unknown. One solution is found by homotopy, and one more by the method of IV-B; in this case, the followed curve is the graph of the function F, presented in figure 6.

2) Dodecagon: This problem consists in constructing 12 points, knowing 21 distances between these points (see figure 10). Solving this problem with our method leads to replace two of the initial problem constraints.

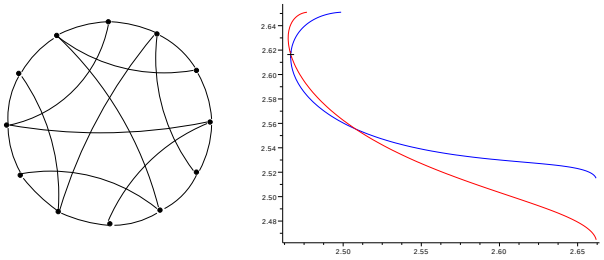

Fig. 10. Left: the constraint graph of a dodecagon. Right: the two curves followed during the search of new solutions, from the solution localised by a plus sign.

The two curves, solution set of the two systems $\mathcal{F}^i$ for $i = \{0, 1\}$, that are followed after computing an initial solution by homotopy, are shown in the right part of figure 10. The points of their intersections are solutions of the problem: in this case, a new solution is found during this stage.

3) Icosahedron: The last problem consists in the construction of an icosahedron: a polyhedron of 12 vertices and 20 faces in a 3-dimensional space. Since each vertex is connected with 5 distance constraints, at least three constraints have to be replaced.

During the step described in IV-C, three systems are considered, and three curves are followed, and one new solution is obtained as shown in figure 12, where the three curves are crossing.

In this case, only one solution is found by sampling (see table of figure 9). This lack is due to the impossibility to have a sufficiently high sampling frequency, because of the computational cost increase.

V. Conclusion

Whatever the constraint system is, the reparametrization provides a construction plan. For this, some constraints could be replaced by others making the system solvable. The construction plan is a symbolic solution that can potentially lead to all numerical solutions.

The fact of using a geometric method driving a numeric one, offers the possibility of having all solutions and allows to reduce the number of equations and variables that are involved. For instance, applying a classic numeric method in order to solve the icosahedron problem (see IV-E3) leads to solve a system of 30 equations in 30 unknowns, whereas our method deals with systems of 3 equations in 3 unknowns.

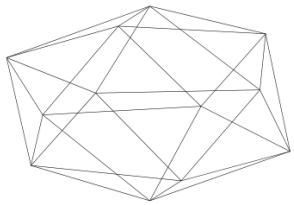

Fig. 11. An icosahedron: vertices represents points in a 3 dimensional space, and edges are distance constraints between these points.

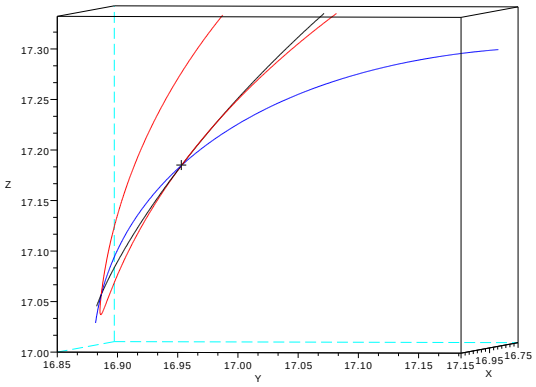

Fig. 12. The three curves followed during the search of new solutions, from the solution localised by a plus sign.

Even if the idea of reparametrization is not new, the sampling method that has been proposed to find numeric values of driving parameters avoids its use for systems that lead to change more than two constraints. Moreover, the number of solutions provided by this method depends on the acuteness of the sampling grid.

Using an efficient numeric method such as tracking paths defined by partial systems allows us to apply this solver to larger problems while producing all of its solutions, which is the issue of this paper.

However, the numeric stage needs to be improved by using geometric informations such as relative positions of the geometric entities, especially to set the parameters in path-tracking, and to manage the branch changing. Allowing the path-tracker to deal with complex numbers and working in a projective space, could improve the numerical processing. Indeed, multiplicity in intersections (e.g. two solutions in circle-circle intersection) causes branch changing and remains numerically sensitive.

References

[1] W. Wu, "Basic principles of mechanical theorem proving in elementary geometries," *Journal of Symbolic Computation*, vol. 4, pp. 207–235, 1984.

[2] S. Chou, *Mechanical geometry theorem proving*, ser. Mathematics and its Applications. D. Reidel, Dordrecht, 1988.

[3] V. C. Lin, D. C. Gossard, and R. A. Light, "Variational geometry in computer-aided design," *SIGGRAPH Comput. Graph.*, vol. 15, pp. 171–177, August 1981. [Online]. Available: http://doi.acm.org/10.1145/965161.806803

[4] H. Lamure and D. Michelucci, "Solving geometric constraints by homotopy," pp. 263–269, 1995. [Online]. Available: http://doi.acm.org/10.1145/218013.218071

[5] D. Michelucci, "Using cayley menger determinants," in *In Proceedings of the 2004 ACM symposium on Solid modeling*, 2004, pp. 285–290.

[6] B. Aldefeld, "Variations of geometries based on a geometric-reasoning method," *Computer-Aided Design*, vol. 20, no. 3, pp. 117–126, 1988.

[7] A. Verroust, F. Schonek, and D. Roller, "Oriented method for parametrized computer-aided design," *Computer-Aided Design*, vol. 24, no. 10, pp. 531–535, 1992.

[8] B. Brüderlin, "Using geometric rewrite rules for solving geometric problems symbolically," *Theoritical Computer Science*, pp. 291–303, 1993.

[9] R. Joan-Arinyo and A. Soto, "A correct rule-based geometric constraint solver," *Computer and Graphics*, vol. 5, no. 21, pp. 599–609, 1997.

[10] J. Owen, "Algebraic solution for geometry from dimensional constraints," in *Proceedings of the 1th ACM Symposium of Solid Modeling and CAD/CAM Applications*. ACM Press, 1991, pp. 397–407.

[11] R. S. Latham and A. E. Middleditch, "Connectivity analysis: a tool for processing geometric constraints," *Computer-Aided Design*, vol. 28, no. 11, pp. 917–928, 1996.

[12] C. Hoffmann, A. Lomonosov, and M. Sitharam, "Decomposition plans for geometric constraint problems, part ii : New algorithms," *J. Symbolic Computation*, vol. 31, pp. 409–427, 2001.

[13] C. M. Hoffmann, A. Lomonosov, and M. Sitharam, "Finding solvable subsets of constraint graphs," in *CP*, 1997, pp. 463–477.

[14] G. Sunde, "Specification of shape by dimensions and other geometric constraints," in *Geometric modeling for CAD applications*. Wozny, M. J., McLaughlin H. W., and Encarnacao, Eds., 1988, pp. 199–213.

[15] W. Bouma, I. Fudos, C. Hoffmann, J. Cai, and R. Paige, "A geometric constraint solver," *Computer-Aided Design*, vol. 27, no. 6, pp. 487–501, 1995.

[16] Q. L. Xiao-Shan Gao and G.-F. Zhang, "A c-tree decomposition algorithm for 2d and 3d geometric constraint solving," *Computer-Aided Design*, vol. 38, no. 1, pp. 1–13, january 2006.

[17] C. Jermann, G. Trombettoni, B. Neveu, and P. Mathis, "Decomposition of geometric constraint systems: a survey," *Int. J. Comput. Geometry Appl.*, vol. 16, no. 5-6, pp. 379–414, 2006.

[18] P. Mathis and S. E. Thierry, "A formalization of geometric constraint systems and their decomposition," *Formal Aspects of Computing*, vol. 22, no. 2, pp. 129–151, 2010.

[19] X.-S. Gao, C. M. Hoffmann, and W.-Q. Yang, "Solving spatial basic geometric constraint configurations with locus intersection," in *Proceedings of the seventh ACM symposium on Solid modeling and applications*, ser. SMA '02. New York, NY, USA: ACM, 2002, pp. 95–104. [Online]. Available: http://doi.acm.org/10.1145/566282.566299

[20] A. Fabre and P. Schreck, "Combining symbolic and numerical solvers to simplify indecomposable systems solving," in *ACM Symposium on Applied Computing SAC 2008*, H. H. Roger L. Wainwright, Ed., ACM. ACM Press, Mar 2008, pp. 1838–1842, iSBN 978-1-59593-753-7. [Online]. Available: http://lsiit-cnrs.unistra.fr/Publications/2008/4-FS08

[21] B. Denner-Broser, "On the decidability of tracing problems in dynamic geometry," *Lecture Notes in Computer Science, Springer Verlag*, no. LNCS 3763/2006, pp. 111–129, 2006.

[22] G. Laman, "On graphs and rigidity of plane skeletal structures," *Journal of Engineering Mathematics*, vol. 4, pp. 331–340, 1970.

[23] J. Graver, B. Servatius, and H. Servatius, *Combinatorial rigidity.*, ser. Graduate Studies in Mathematics, Providence, RI, 1993.

[24] H. A. van der Meiden and W. F. Bronsvoort, "A constructive approach to calculate parameter ranges for systems of geometric constraints," *Comput. Aided Des.*, vol. 38, pp. 275–283, April 2006.

[25] A. Morgan and A. Sommese, "Coefficient-parameter polynomial continuation," *Applied Mathematics and Computation*, vol. 29, no. 2, pp. 123–160, 1989.

[26] E. Allgower and K. Georg, "Continuation and path following," *Acta Numerica*, vol. 2, no. -1, pp. 1–64, 1993.

[27] C. Durand and C. Hoffmann, "Continuum: A Homotopy Continuation Solver for Systems of Algebraic Equations," Technical Report TR 98-028, Department of Computer Sciences, Purdue University, Tech. Rep., 1998.

[28] D. Faudot and D. Michelucci, "A new robust algorithm to trace curves," *Reliable computing*, vol. 13, no. 4, pp. 309–324, 2007.