Helping the Designer in Solution Selection: Applications in CAD

Caroline Essert-Villard

Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection
UMR CNRS 7005 – Université Louis Pasteur
Pôle API, Boulevard Sébastien Brant – F-67400 Illkirch, France
essert@dpt-info.u-strasbg.fr

Abstract. In CAD, symbolic geometric solvers allow to solve constraint systems, given under the form of a sketch and a set of constraints, by computing a symbolic construction plan describing how to build the required figure. But a construction plan does not usually define a unique figure, and the selection of the expected figure remains an important topic. This paper expose three methods, automatic or interactive, to help the designer in the exploration of the solution space. These methods guide him towards the expected solution, by basing the construction on the observation of the sketch. A set of examples from a large range of application domains illustrate the different methods.

1 Introduction

Over the past 20 years, many studies have been done about solving geometric problems. Geometric solvers offer the possibility to solve constraint systems graphically and interactively given by a designer under the form of a dimensioned sketch. Several kinds of methods, numeric, symbolic, or even combined, are used to solve such geometric constraint systems (see R. Anderl and R. Mendgen for a survey [1]). In the case of symbolic solvers, and if the dimensioned sketch is well constrained, the result is a symbolic *construction plan*. Its numerical interpretation enables to generate all the solutions of the system.

This approach has been chosen in our team because of its exactitude, and the ability to propose not only one solution satisfying the constraints, but the whole solution space. The solution space provided by the symbolic geometric solver *YAMS* [2], that was developed from this approach, can be represented as a tree. Indeed, as the construction plan is a list of definitions, in wich the function may generate several results (such as the intersection between a line and a circle), we can build a branching in the *solutions tree* at each time the function actually generates several results, becoming a *multifunction*.

Even in the well constrained case, a constraint system generally defines a large number of figures, whereas the designer usually wants the unique figure corresponding to his will. This sets the problem of how to select the "expected" solution ? We think that a designer does not draw a dimensioned sketch haphazardly, but has already in mind the kind of solution he wants. We take advantage of this idea to help him in the selection of the pertinent solution.

In this paper, we will illustrate our methods by presenting some results of experimentations in several applicative domains. These examples show at once the advantages of the methods, how they are limitated, and which answers to these limitations would be helpful. Note that all figures presented in this paper are snapshots from our prototype *SAMY*, extending *YAMS*.

2 Solutions Selection and Solution Space Browsing

Whatever the solving method, the standard response given by most of the solvers is to browse the whole solution space, and/or to use some simple pruning heuristics. But in the case of large systems, such an exploration may be both tedious and very long. Let us see what propositions have been made so far by various authors to the problem of the selection of a solution amongst many.

B. Brüderlin briefly observed in [3] the existence of ambiguous solutions, and proposed to introduce additional constraints to help clearing the ambiguity. But doing this may lead to an over-constrained system.

J. Owen exposed in [4] an idea, yet oulined by B. Aldefeld ([5]), to solve the ambiguities. He proposed to compare the relative placements of the elements of the solution figure with those of the given sketch. However, for this kind of method, the number of criteria to give is too important, and furthermore these criteria are, to our mind, very insufficient to find the expected solution (see [6]).

C. Hoffmann *et al.* were some of the firsts to be really interested in the multiple solutions problem. In [7], they proposed 3 ways to find the expected solution. The first one, quickly abandoned, was to move the misplaced elements of a figure. The second one was to over-constrain the system. The third one was to propose an "incremental" mode, allowing the user to modify an element at a "level" of the construction if it is not suitable. But this method didn't seem to be very intuitive yet. Moreover, after a first change, the figure had to be "regenerated" before another change could be done. From that point of view, the use of a symbolic solving method would be costless in time and processes.

In the end, the problem of selecting the most relevant solution amongst the whole solution space, instead of simply one solution complying with the constraints, has not been studied yet in a satisfactory way. That is why we propose to improve in some way the outset of an answer brought by C. Hoffmann *et al.*, with a deeper study of some automatic and interactive methods. For this work, we will take advantage of the properties of the solving method followed in the existing symbolic geometric solver of our team, *YAMS* [2], and stay in that framework.

3 Automatic and Interactive Tools

We propose two categories of methods and associated tools: the first one is automatic (selection with no intervention of the user) and the second and third ones are interactive (letting the user choose).

3.1 Branch Freezing

As we explained before, we think that the designer has in mind the kind of figure he wants to obtain when he draws his dimensioned sketch, and that in most cases he takes into account his idea to draw a sketch having likeness to his will. That is why we intend to use the sketch in order to find the solution that has the best likeness with it. Let us precise that we work on plans yet computed by *YAMS*.

Likeness Let us first define our notion of likeness. Authors often use comparisons of relative placements of the elements of the figure to determine if two figures are similar or not. We only notice that most of these properties comparisons can be held in check by some simple examples (see[6]). So, we proposed a better criterion based on homotopy, called *S-Homotopy*. Homotopy as in [8] is too general for our case, whereas S-homotopy takes exactly into account geometric figures solutions of constraint systems, by including continuous deformation of constraint systems themselves to ensure homogeneity. It also includes a very important point that imposes the continuous deformation not to go through a degenerate case. If that occured, we could jump to another solution.

This led us to propose a numbering for multifunctions values, which is compatible with the continuous deformation of the geometric constraint systems, and which we defined as *continuous numbering*. This means that whatever the continuous deformation of the considered constraint system, the result numbered n will always keep the same geometric properties. The link between the notions of likeness and continuous numbering is that two figures e and f are S-homotopic if and only if they have the same number in their solutions tree. In addition, given a sketch e, there is at most a unique solution f such that e and f are S-homotopic, *i.e.* that have, in our sense, the best likeness.

Freezing Technique We first point out the fact that the sketch can be seen as a particular solution of the constraint system instantiated by the values read on the sketch. So, it corresponds to a particular branch of the solutions tree, that we call the *sketch branch*. Then, we suppose that the user gave a sketch and some constraints that look like his expectations, that is, in the sense we defined earlier, there is a S-homotopy between the sketch and the solution. Thus, if we can find a solution having the same number than the sketch, then we advance the idea that it is the intended solution. The operation consisting in storing the number of the sketch branch is called *freezing* of a branch. When this number is known, we can launch an interpretation with the given dimensions, guided by the number of the *frozen* branch.

One of the advantages of this method is its speed. Indeed, instead of potentially comparing some geometric properties of all the objects of the figure with each other, we only compare, at each junction of the tree, the objects that are brought into play in the concerned multifunction. Since the treatment is made as the interpretation goes along, this method reduces significantly the processing time in comparison with a systematic method. Another advantage is that this method is very general, thanks to its continuity justification by S-homotopy.

Limitations The method we exposed in previous section works fine when all constraints are metric. But another type of constraints is also used in *YAMS*: Boolean constraints. As examples, we can cite tangency, or equality of objects. In the case of Boolean constraints, some information is missing to find the intended solution. In that case, the freezing of a branch produces, instead of a unique branch, a small sub-tree of the initial solutions tree.

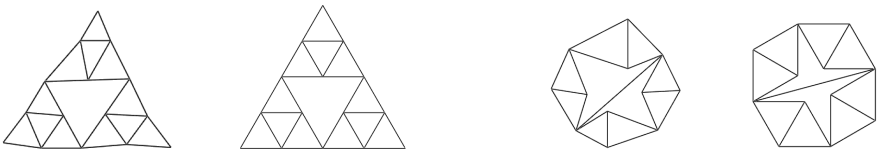

Fig. 1. Neveu *et al.* examples: sketches and solutions

Applications Geometric constructions under constraints can be used to represent objects in various domains. We present here several kinds of examples, illustrating the use of the branch freezing to find the expected solution in varied situations.

First, let us take an example from literature. In [9], B. Neveu *et al.* presented geometric figures made of triangles, that we recall on Fig.1. The constraint systems, containing only distance constraints, provide respectively 128 and 64 solutions. If we use our branch freezing technique, we obtain an instantaneous good answer for both (second and fourth parts of Fig.1), after an also instantaneous symbolic resolution. This technique avoids to compute all the solutions and review them. To have an idea of what reviewing them one by one would cost at worst, the authors cite the following approximative process times (not including viewing): 9 sec. and 1.4 sec. on a Pentium III 500, respectively.

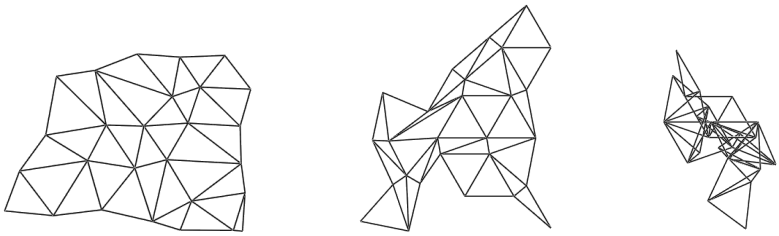

Fig. 2. Triangulation example: the sketch, the right solution, and one of the rejected solutions

The next example is a representative triangles problem, as we can meet in triangulation studies. The sketch of Fig.2 is composed of 34 adjacent triangles, with constrained sides. This kind of problem is often cited, for instance by Owen in [4], and well known to have 2^{p-2} solutions if p is the number of points, that is in our case 33554432 potential solutions. On the middle of Fig.2, we can see

the result found instantaneously with the branch freezing method. On the right of Fig.2 is presented one of the solutions rejected by our method.

Let us take now an example describing a more concrete object, and that will illustrate the limitation of this method. The sketch on top of Fig.3 represents a lever, as it could be drawn for manufacturing, given with the appropriate constraints on distances, angles, and two tangency constraints involving two of the arcs on the right. As all the constraints are not metric, the branch freezing can't reduce the solution space to one single branch, but reduces it to a small sub-tree with only 4 branches representing the 4 remaining solutions (see Fig.3).

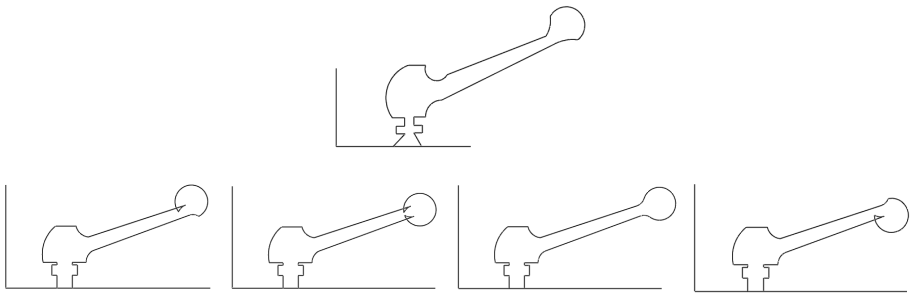

Fig. 3. The lever: the sketch and 4 remaining solutions

If the remaining sub-tree is, like this one, very small, it is quite easy to find the good solution by viewing all of them one by one. But if the sub-tree is too large to be reviewed this way, or if none of them correspond to the user's intend, either because the sketch was too close to a limit case, or because he wants to see other solutions, it becomes useful to provide other tools to help him. That is why we propose next two interactive tools that can constitute a complement for this first method, or be used alone as well.

3.2 Step by Step Interpretation

First, let us recall that the whole solution space may be very wide, and that exporing it may be quite tedious. In order to help the user exploring efficiently the solution space, a first proposal is to let the user take the choices in hand. This is possible because our approach is formal and we have a construction plan, generating a solutions space structured as a tree labeled with the numerical results of multifunctions. Thanks to that, it is easy to do a step by step evaluation, allowing the user to choose, at each fork of the tree, a value among the available results. This technique is quite similar to the one explained in [7], but our symbolic approach allow us a more efficient use because the figure does not have to be regenerated between each step.

Preconditions The solver may not provide automatically the construction plan in the appropriate form, and it may be necessary to make a *topological sort*

before performing the step by step interpretation. The topological sort is made by placing first the definitions corresponding to visible elements, called *sketch definitions*, according to the current order, and then interleaving the definitions of auxiliary construction elements just before the first sketch definition that needs it (i.e. that contains it as an argument). That way, if a backtracking has to be done, it remains located within a layer delimited by two sketch definitions. So, if none of the results satisfy the user, then we are sure that a previous sketch definition has to be thrown back into question.

Technique When a construction plan is sorted as we explained above, the only backtracking to be done in the solutions tree is located in the subtree between to sketch definitions $d1$ and $d2$, excluding the highest one (if the root is on top), say $d1$. If the user is not satisfied with any of the numerical interpretations proposed for the lowest one ($d2$), and wishes to see other possible solutions, then we are sure that some of the other sketch definitions have to be thrown back into question.

In such a case, we browse the sketch definitions that have been defined earlier, and on which $d2$ depends. We suggest to the user to reconsider some of the values he had chosen for these previous sketch definitions. First, we propose him to review only a few of them, those that are placed closer in the tree. Then, progressively we put into consideration more definitions, including those that were defined a longer time ago.

Limitations This method is the most precise and allows a total control over the construction. However, if the example is complex, the construction plan is long and contains a lot of multifunctions, and the solutions tree is very large, then it may be quite long to examine each step of the construction one by one to guide it from the beginning to the end. In addition, if the construction plan is the result of the assembling process of several sub-figures solved in various auxiliary coordinate systems, then it may be difficult to have an idea of the final result in the course of the process.

The technique can be enhanced with several kinds of breakpoint tools, for instance to offer the opportunity to freeze a part of a solution tree between two breakpoints, and then to skip this part that has become a big step. But this implies knowledge on the construction plan configuration and on the definitions on the user's part. That led us to introduce a more intuitive tool, that we will present in next Section.

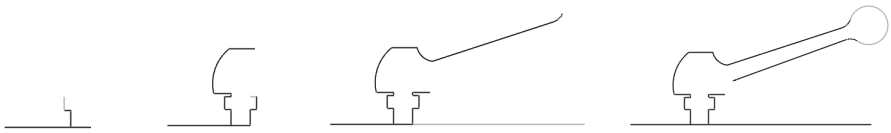

Fig. 4. The lever: 4 of the 29 steps

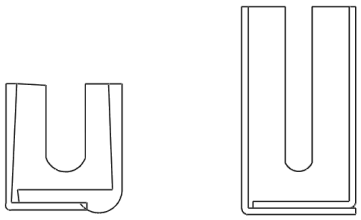

Fig. 5. The support: sketch and expected solution

Applications In the case of the lever, no decomposition was necessary to compute the construction plan, so there isn't any discontinuity during the step by step construction, making it quite easy. On Fig.4, we can see some of the 29 steps leading to the final lever.

On the opposite, a decomposition in two sub-figures was necessary to compute the construction plan of the example of Fig.5, taken from a blueprint from building industry. As this rail support is the result of the assembling of two subfigures built in two different coordinate systems, there are jumps of some parts of the figure during the step by step process. As we can see on Fig.6, the first part of the figure is easily built with this method. But when we start to build the second part, some elements of the first part automatically move to fit the second part. This is due to the displacements applied to assemble the subfigures in the same coordinate system. This implies a temporary incoherence in the figure.

This phenomenon is increased with the number of subfigures. In the triangulation example, the construction plan is made of 4 subfigures, so there are 4 different temporary coordinate systems. Then, the step by step construction is quite difficult to control, as we can see on steps n.20 and n.21 shown on right hand side of Fig.7. Moreover, the coordinate systems may be more or less far from the coordinate system used in the sketch, so the user may be mislead when appreciating the directions, see on left hand side of Fig.7 where the beginning of subfigure 2 is in grey.

Therefore, we tried to find another way to explore the solution space, still involving the user's interaction, but more intuitive, that would free the user from the subfigures problems.

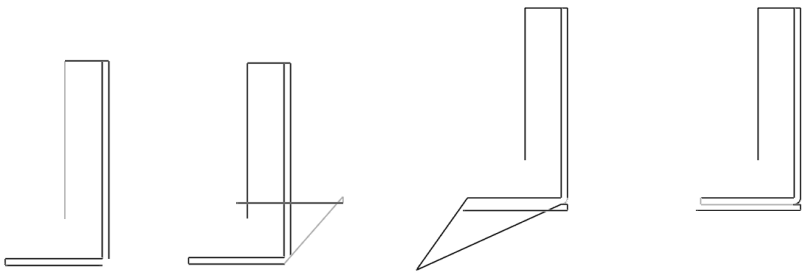

Fig. 6. The support: 4 of the 22 steps

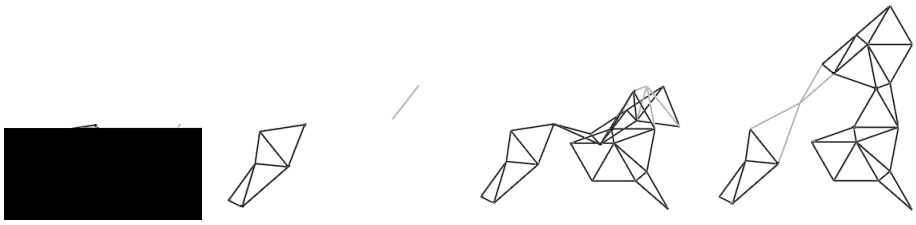

Fig. 7. Triangulation. Left: starting the second subfigure. Right: jump in step by step process (steps 20 and 21)

3.3 Interactive Graphical Manipulation

This approach is based on a completely computed solution. When this is done, we propose the user to review it by moving the misplaced elements with his mouse towards other available places.

Preconditions This method needs the same topological sort as the step by step interpretation, for identical reasons. Auxiliary definitions are placed in the construction program only when they are needed by a sketch definition, and not before. That way, if we change of branching in a layer, we can go on following the same numbering in the layers below if they are not dependent on this layer.

Principle The process starts with a computed solution figure. If the user thinks that an element of the given figure, or a part of it, is not at the right place, then he can point his mouse on the misplaced element. Then, other available values for this element are shown on the figure, according to the initial constraints. The user can drag-and-drop the element towards the new place, and the rest of the figure is updated, by following the same scheme in the tree. It may happen that the element the user moved is linked with some others because of a dependence between them. If so, the whole set of linked elements will be moved altogether.

To propose other solutions for one element, we simply take advantage of the structure of the solution space, by looking in the solutions tree which result can be found by doing a backtracking in the smallest sub-tree containing the element and other values. Doing this ensures that the other proposed values comply with the constraints. The solutions tree allows us to easily find the other possible solutions for one element by jumping from one branch to another.

Limitations However, in some cases, manipulating a figure may not be as intuitive as we wish. For instance, we may want to move an element on which a large part of the figure depends. If so, it is very difficult to predict what will be the behaviour of the dependent elements.

Moreover, we remain dependent on the construction program. For example, if the user wants to move the element used as the origin of the system, he will not

be able to do it unless he makes an equivalent inverse displacement on all other elements of the figure, or unless the system is solved again starting from another origin. We plan to search a way to rearrange dynamically the construction plan during the process, according to the needs.

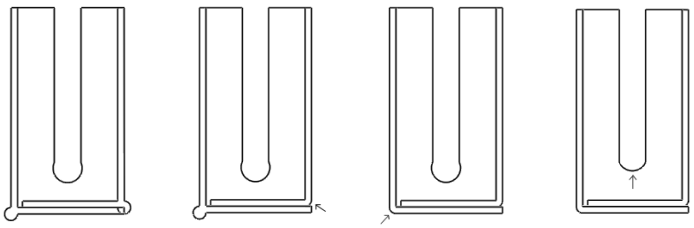

Fig. 8. The support: 4 steps of the interactive manipulation

Applications Let us take again the significative, and quite simple to understand, example of the rail support. The arcs that make up the left and right bends and the central arc are built using a multifunction that can provide up to 2 solutions, according to the parameters. Thus, suppose that we did not use the branch freezing method, and that the computed solution is the one presented on top left part of Fig.8. In that case, the user can successively move the 3 misplaced arcs to their respective alternative solution (see steps on Fig.8), in order to obtain the final result shown on right of Fig.8. In the same way, points can also be moved with this method. Most of the time, these manipulations of elements are intuitive enough to allow a quick convergence to the expected solution.

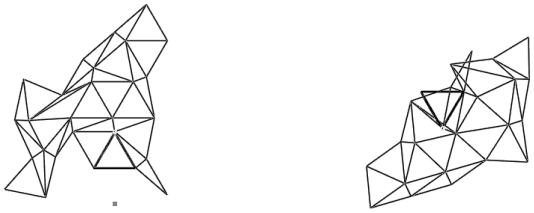

Fig. 9. Triangulation: jump in interactive manipulation of one point

Unfortunately, sometimes interactive manipulation may not be intuitive enough. As we said in Section 3.2, the triangulation of the space is computed using 4 subfigures. This is also a drawback for this manipulation method, because the displacement of one point may have consequences over several subfigures and may imply an update of a large part of the final figure. On Fig.9, we show a jump caused by the displacement of one single point (part of the bold triangle) that made a large number of other points move with the update: a part of the figure is like "bended" after the change. However, this example was only reused

here as a case study, since it was perfectly handled by branch freezing (recall Section 3.1).

4 Conclusion

In this paper, we tried to bring a part of an answer to the problem of the multiplicity of solutions provided for geometric constraint systems. We presented 3 methods to help the designer to choose between the multiple solutions yielded by the symbolic geometric solver *YAMS*. These methods were accompanied by a sample of application examples covering several domains.

All the methods we exposed here have both advantages and drawbacks. Most of the time, if one has a limitation, one of the others can either reduce its effects or even take its place, and help the user in his investigations. For instance, the interactive methods handle the cases of Boolean constraints and help in browsing a remaingin sub-tree, result of a branch freezing. These multiple combinations of tools are currently being studied, as well as other new heuristics.

This quite applicative presentation points out at once that symbolic solvers, and most particularly *YAMS*, allow to handle a large range of problems in many fields, and that they lead in most cases quickly to the expected solution without redrawing the sketch.

References

1. Anderl, R. and Mendgen, R.: Parametric design and its impact on solid modeling applications. In proceedings of 3rd ACM/IEEE Symposium on Solid Modeling and Applications, ACM Press (1995) 1–12
2. Dufourd, J.-F. and Mathis, P. and Schreck, P.: Formal resolution of geometric constraint systems by assembling. In proceedings of the ACM-Siggraph Solid Modelling Conference, ACM Press (1997) 271–284
3. Sohrt, W. and Brüderlin, B.: Interaction with Constraints in 3D Modeling. In proceedings of Symposium on Solid Modeling Foundations and CAD/CAM Applications, ACM Press (1991) 387–396
4. Owen, J.: Algebraic solution for geometry from dimensional constraints. In proceedings of the 1st ACM Symposium of Solid Modelling and CAD/CAM Applications, ACM Press (1991) 397–407
5. Aldefeld, B.: Variations of geometries based on a geometric-reasoning method. Computer-Aided Design, Elsevier Science (1988) 20(3):117–126
6. Essert-Villard, C. and Schreck, P. and Dufourd, J.-F.: Sketch-based pruning of a solution space within a formal geometric constraint solver. Artificial Intelligence, Elsevier Science (2000) 124:139–159
7. Bouma, W. and Fudos, I. and Hoffmann, C. and Cai, J. and Paige, R.: Geometric constraint solver. Computer-Aided Design, Elsevier Science (1995) 27(6):487–501
8. Lamure, H. and Michelucci, D.: Solving Constraints Systems by Homotopy. In proceedings of 3rd ACM/IEEE Symposium on Solid Modeling and Applications, ACM Press (1995) 263–269
9. Jermann, C. and Trombettoni, G. and Neveu, B. and Rueher, M.: A Constraint Programming Approach for Solving Rigid Geometric Systems. In proceedings of CP2000, Springer LNCS 1894 (2000) 233–248