Computing Parameter Ranges in Constructive Geometric Constraint Solving: A Correctness Proof

Marta Hidalgo, Robert Joan-Arinyo, Toni Soto
Grup d'Informàtica a l'Enginyeria
Universitat Politècnica de Catalunya

April 11, 2011

Abstract

In parametric design, changing values of parameters to get different solution instances to the problem at hand is a paramount operation. One of the main issues when generating the solution instance for the actual set of parameters is that the user does not know in general which is the set of parameters' values for which the parametric solution is feasible. Similarly, in constraint-based Dynamic Geometry, knowing the set of critical points where construction feasibility changes would allow to avoid unexpected and unwanted behaviors. In this work we report on our experiments implementing the van der Meiden Approach to solve the problem in a 2D space and prove that it is correct.

1 Introduction

Many applications in computer-aided design, computer-aided manufacturing, kinematics, robotics or dynamic geometry are conveniently modeled by geometric problems defined by geometric constraints with parameters, some of them representing dimensions. These generic models allow the user to easily generate specific instances for various parameter and constraint values.

When parametric models are used in real applications, it is often found that instantiation may fail for some parameter values. Assuming that failures are not due to bugs in the system, they should be attributed to a more basic problem, that is, a certain combination of constraints in the model and values of parameters do not define a valid shape.

The failure to instanciate the model poses naturally the question of how to compute ranges for parameters such that model instantiation is feasible. This problem or restricted versions of it have been addressed in the literature. Shapiro and Vossler, [21], and Raghothama and Shapiro, [18, 19, 20], developed a theory on validity of parametric family of solids by investigating the relationship between Brep and CSG schemas in systems with dual representations for solid modeling. The formulation is built on formalisms of algebraic topology. Unfortunately, it seems a rather difficult problem transforming these formalisms into effective algorithms.

Joan-Arinyo and Mata [13] reported on a method to compute feasible ranges for parameters in geometric constraint solving under the assumption that values assigned to parameters are non-trivial-width intervals. The method applies to complex systems of geometric constraints in both 2D and 3D and has been successfully applied in the dynamic geometry field, [5]. It is a general method, the main drawback, however, is that it is based on numerical sampling.

Hoffmann and Kim [8] developed a constructive approach to calculate parameter ranges for systems of geometric constraints that include sets of isothetic line segments and distance constraints between them. Model instantiation for distance parameters within the ranges output by the method preserve the topology of the set of isothetic lines.

In an illuminating work, van der Meiden and Bronsvoort, [24], reported on a constructive method to calculate parameter ranges for systems of geometric constraints. Constraint systems are restricted to systems of distance and angle constraints on points in 2D or 3D spaces that are wellconstrained and decomposable into triangular and tetrahedral subproblems. The method automatically determines the allowable range for a single parameter of the system, called *variant parameter*, such that an actual solution exists for any value in the range. The method consists of two steps. First a set of values for the variant parameter, called *critical* points, [4], for which some welldefined subproblem feasibility changes is computed. Once sorted, critical points define a sequence of intervals and their feasibility is established by checking feasibility at some point within each interval.

The van der Meiden method is the subject of our study. After some preliminar questions, we fix concepts related to geometric constraint problems with one variant parameter. Then we describe the method in detail and report on our own implementation in a 2D scenario. Finally we formalize the underlying concepts and prove that it is correct.

2 Preliminaries

First we recall some basic concepts related to geometric constraint solving in general. Then we focus on the constructive technique. For an in depth discussion on this topic see, for example, [1, 2, 3, 6, 9, 10, 11, 14, 15, 16, 22].

2.1 Geometric Constraint Problems

In this paper we focus on the basic constraint problem defined as follows. Given a set of geometric elements G and a set of constraints between them C, place each geometric element in such a way that the constraints are fulfilled. We consider 2D geometric constraint problems defined by a set of geometric elements like points, lines, line segments,

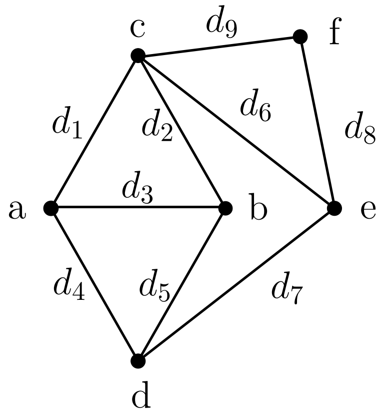


Figure 1: Geometric constraint problem example.

circles and circular arcs with fixed radius, along with a set of constraints like distance, angle, incidence and tangency between any two geometric elements.

Figure 1 shows an example of geometric constraint problem consisting of six points $\{a, b, c, d, e, f\}$, nine point-point distance constraints represented in Figure 1 by the straight segments, and the set of parameters $\{d_i, 1 \leq i \leq 9\}$.

In what follows a geometric constraint problem will be denoted by a tuple $\Pi =< G, C, P >$ where G is the set of geometric objects, C the set of constraints defined on G and P is the set of parameters of constraints in C.

Constraint solving community is mainly interested in objects which are invariant under rigid transformations of translation and rotation. This property is known as *rigidity*. The intuitive concept of rigidity, the one that will be used here, is defined from the number of solutions of the considered problem. In this context, geometric constraint problems are categorized in three different families:

1. Well constrained problems are geometric constraint problems with a non-empty anf finite set of solutions.

2. Over-constrained problems are those problems with no actual solution. Generally, the elimination of one or more constraints results in a well constraint problem.

3. Under-constrained problems are geometric constraint problems for which an infinite set of solutions exists. In these cases, not enough constraints are given.

In this work we only consider well constrained problems.

2.2 Constructive Geometric Constraint Problems Solving

Geometric constraint solving is arguable a core technology of computer aided design and, by extension, geometric constraint solving is also applicable in virtual reality and is closely related in a technical sense to geometric theorem proving. For solution techniques, geometric constraint solving also borrows heavily from symbolic algebraic computation and matroid theory.

Many techniques have been reported in the literature that provide powerful and efficient methods for solving systems of geometric constraints. For a review, see Hoffmann *et al.*, [7]. Among all the geometric constraint solving techniques, our interest here focuses on the one known as *constructive*.

Constructive geometric constraint solving is a technique widely used in the geometric constraint solving field. We briefly recall here the main features underlying this technique. An architecture for constructive solvers is illustrated in Figure 2 where square boxes are *functional units* and rounded boxes are *data entities*. The functional units are the analyzer, the index selector and the constructor. The data entities are the geometric constraint problem, the construction plan, the parameters assignment and the index assignment.

In this technology, the user defines a geometric constraint problem by sketching some geometric elements taken from a given repertoire (points, lines, circles, etc) and annotates the sketch with a set of geometric relationships, called *constraints*, (point-point distance, point-line distance, angle between two lines and so on), that must be fulfilled.

Given the geometric constraint problem, $\Pi =< G, C, P >$, the analyzer is responsible for figuring out whether the solver is able to solve the problem up to degenerated configurations, that is, whether it can find a placement for the geometric objects such that the constraints hold. If the answer is positive, the analyzer outputs the solution as a sequence of construction steps, known as *construction plan*, that will place the geometric elements in the right position. Figure 3 shows a construction plan for the constraint problem given in Figure 1. The meaning of each construction step is the usual. For example, $origin()$ stands for the origin of an arbitrary framework, $b = distD(a, d_3)$ places point b at distance d_3 from point a, $c_2 = circleCR(a, d_1)$ defines the circle c_2 with center a and radius d_1 and $intCC(c_1, c_2)$ defines a point as the intersection of circles c_1 and c_2. Notice that symbols c_i do not represent entities in the problem. They are intermediate results introduced to increase readability.

Solving a geometric constraint problem can be seen as solving a set of, in general, non linear equations. Therefore, each equation can have as many roots as the equation degree. Obviously, each specific root will result in a different placement for the geometric elements in the problem. Selecting the desired root, known as the *Root identification problem*, [2], is the goal of the index selector that associates with each equation with several roots an index that unambiguously identifies the desired root. The index in the construction plan in Figure 3 is $I = \{s_1, s_2, s_3, s_4\}$. For an in depth study of the

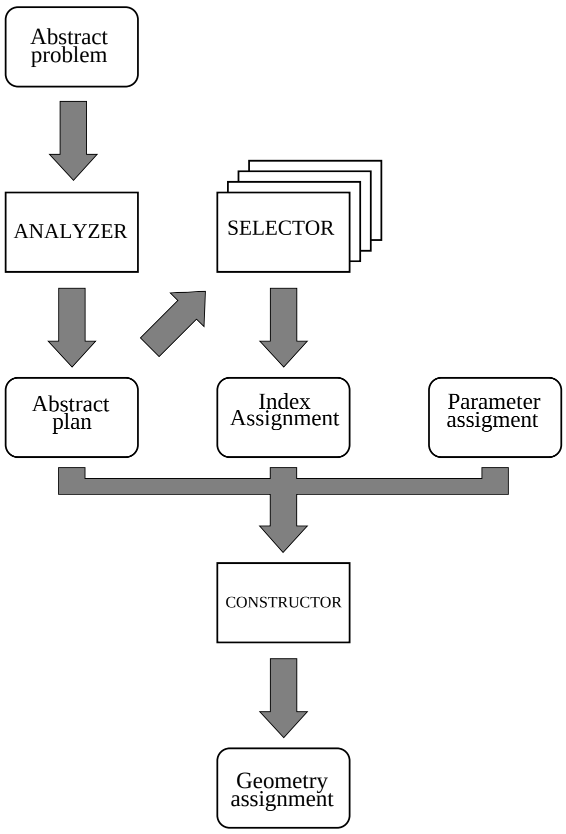


Figure 2: An architecture for the constructive solving technique.

1.	a	$= origin()$	8.	d	$= intCC(c_4, c_5, s_2)$
2.	b	$= distD(a, d_3)$	9.	c_6	$= circleCR(c, d_6)$
3.	c_2	$= circleCR(a, d_1)$	10.	c_7	$= circleCR(d, d_7)$
4.	c_3	$= circleCR(b, d_2)$	11.	e	$= intCC(c_6, c_7, s_3)$
5.	c	$= intCC(c_2, c_3, s_1)$	12.	c_8	$= circleCR(c, d_9)$
6.	c_4	$= circleCR(a, d_4)$	13.	c_9	$= circleCR(e, d_8)$
7.	c_5	$= circleCR(d, d_5)$	14.	f	$= intCC(c_8, c_9, s_4)$

Figure 3: Construction plan for the example problem in Figure 1.

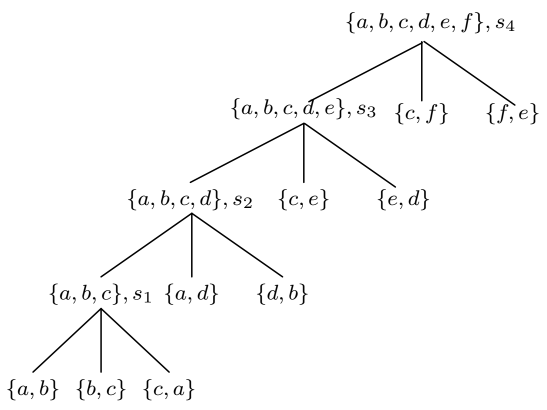


Figure 4: Construction plan as a tree decomposition.

index and the role it plays in a geometric constraint solving see [5]. A number of techniques have been developed to deal with the Root identification problem. See, for example, [2, 12, 17, 23].

The specific solution to the constraint problem Π identified by an assignment of values to the index I is called the *intended solution.* In what follows we consider that the intended solution has been fixed and that equations degree is at most 2, that is, signs s_i in the index take values in, say $\{-1, 1\}$.

Once values have been assigned to the indices, a more convenient way to represent the construction plan is the decomposition tree [15], a way to representent decomposition-recombination algorithms (DR-algorithms) that solve geometric constraint problems, [10]. Figure 4 shows a decomposition tree for the construction plan in Figure 3. Each node in the tree stands for a rigid object, called *cluster*, built on the geometric objects included in the curly brackets list and whose position relative to each other has already been determined. Leaf nodes represent elemental placement problems corresponding to two geometric elements and the constraint defined on them. For example: two points at a given distance, a point and a straight segment at a given distance, two straight segments at a given angle and so on. Edges in the decomposition tree represent the merging of three solved clusters into a larger rigid cluster by application of a specific solving rule. The root node includes all the geometric elements in the problem and represents a solution instance.

Notice that sibling clusters pairwise share one geometric element, for example clusters $\{a, b, c, d\}$, $\{d, e\}$ and $\{c, e, f\}$ pairwise share d, e and c respectively. Figure 5 illustrates the situation. Shared geometric elements d, e and c are called *hinges.* For a more formal rational on this topic see [6] and [14].

Finally, once a set of actual values have been assigned to the constraint parameters and the intended solution has been selected by assigning values to the index signs, the constructor builds an instance of a placement for the geometric objects, a solution

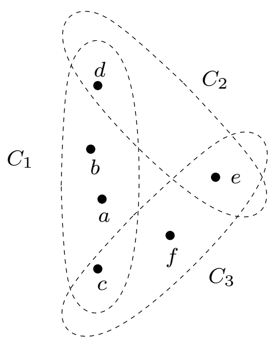


Figure 5: Sibling clusters pairwise share one geometric element.

instance, provided that no numerical incompatibility arises due to geometric degeneracy.

This architecture, known generically as DR-planner, [10], shows some nice properties. First, the nature of the computations in each step is quite different. The analyzer requires symbolic computation while the constructor only performs numerical computations. Second, determining whether the problem is solvable by the solver at hand or not is performed in the analysis step and it does not depend neither on the actual parameter values nor on the geometric computations. Next, with the proposed decoupling, when computing instances for different parameter values, only the construction step needs to be carried out. This allows to skip the analysis step, which is computationally the most expensive, as well as the index selection. Finally, given a symbolically solvable geometric constraint problem and a parameters assignment, the object can be instantiated if there are not numerical impossibilities. These impossibilities are detected while carrying out the geometric computations and we say that the construction plan is unfeasible.

Construction plans expressed as decomposition trees will be denoted by T.

3 Problems with One Variant Parameter

In this section we present basic concepts concerning geometric constraint problems for which the value of a given constraint parameter is not fixed.

3.1 The Construction Plan as a Function

Let $\Pi =< G, C, P >$ be a well constrained geometric constraint problem such that all parameters in P have been assigned a given value except for one, say λ, which can take arbitrary values in $\mathbb{R}$. We will say that the resulting problem has one variant parameter. Figure 6 shows a problem with one variant parameter.

Let T be a construction plan for the constraint problem $\Pi =< G, C, P >$. Since the

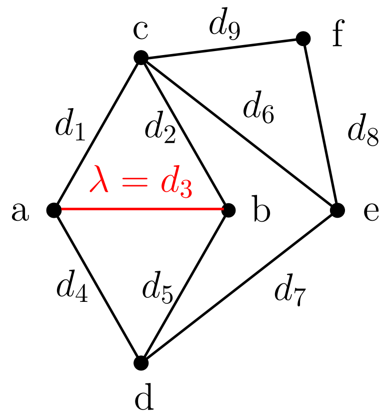


Figure 6: Geometric constraint problem with one variant parameter λ.

construction plan of a constraint problem does not depend on the specific values assigned to parameters in P, T is a construction plan valid for any problem derived from Π by considering one of its parameters as variant. Therefore T(λ) defines a family of objects whose members are built as the value assigned to λ changes. Figure 7 shows from left to right objects in the family defined by the problem in Figure 6 for distance constraint values $d_1 = 3, d_2 = 3, d_4 = 3.5, d_5 = 3.5, d_6 = 4, d_7 = 4.5, d_8 = 4, d_9 = 3.5$ and values of the variant parameter λ in $\{2.5,\ 4.5,\ 5.9\}$.

For some values of the variant parameter λ, however, it may not be possible to satisfy the set of constraints in C, that is the construction plan T is unfeasible for such variant parameter values. To formalize concepts related to construction plan feasibility, we need some definitions.

Definition 3.1 Let $\Pi =< G, C, P >$ be a geometric constraint problem and T a construction plan that solves Π. Let $x = (x_1, \ldots, \lambda, \ldots, x_n)$ be the set of parameters in P with λ the variant parameter. Let $x_c = (x_1, \ldots, \lambda_c, \ldots, x_n)$ be the set of parameters where feasibility of T changes. We say that x_c is a critical point of T and λ_c is a critical variant parameter value.

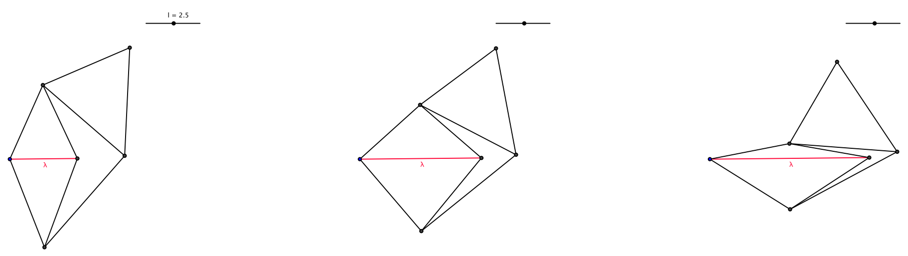


Figure 7: Objects belonging to the family defined by the problem in Figure 6.

1. $a \quad = origin()$
2. $b \quad = distD(a, d_1)$
3. $c_1 \quad = circleCR(b, d_2)$
4. $l \quad = linePA(a, \lambda)$
5. $c \quad = intCL(c, l, s)$

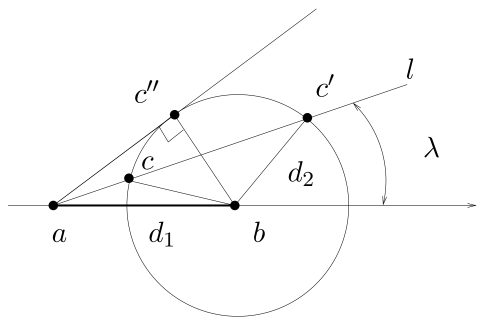


Figure 8: Critical points for a triangle defined by two sides and the angle suported by one of them. Construction plan and actual construction.

To illustrate this concept, consider the construction shown in Figure 8 where a triangle is defined by giving the constraints $b = distD(a, d_1)$, $c = distD(b, d_2)$, and $\lambda = angle(ab, ac)$. If we assume that $d_1 \geq d_2$ and consider λ as the variant parameter, the construction plan shown on the left of Figure 8 is feasible for values of λ in the range $[0, \sin^{-1}(d_2/d_1)]$. The bounds of this range are the critical values of λ for this construction.

The situation described can be found for each basic construction in a constructive solver and the corresponding feasibility ranges can be collected in a dictionary. Table 1 shows examples for some basic constructions.

Definition 3.2 Let $\Pi =< G, C, P >$ be a geometric constraint problem with one variant parameter λ in P and let T be a construction plan that solves Π. The domain of λ is the set of values for which T is feasible.

In general the domain of a variant parameter is a set of disjoint intervals bounded by critical variant parameter values.

In this context, a construction plan can be considered a function of the variant parameter, $\mathrm{T}(\lambda)$. Since construction plan feasibility changes only at critical points, as the value of λ changes continuously in its domain, the solution instance generated by $\mathrm{T}(\lambda)$ traces a continuous path in the space of solutions to problem Π.

3.2 Direct and Indirect Dependency

Dependency of a constraint problem on the variant parameter is a central concept in this work. Figure 9 illustrates the following definitions.

Definition 3.3 Let $\Pi =< G, C, P >$ be a constraint problem with clusters C_1, C_2 and C_3. Let u, v and w be the hinges with $u, v \in C_1$, $v, w \in C_2$ and $w, u \in C_3$. Let the variant parameter λ be such that the corresponding constraint is defined upon hinges u, v in C_1. We say that problem Π depends directly on λ.

Note that computing critical values of the variant parameter in directly dependent prob-

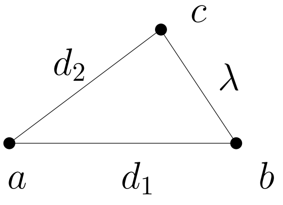
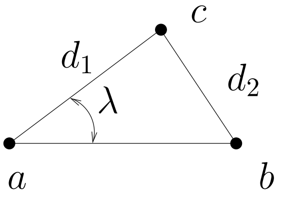
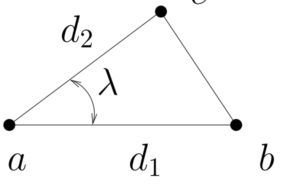
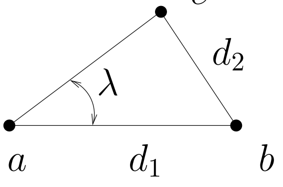
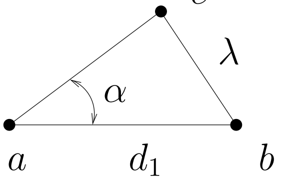
Basic Construction	Feasibility	
	$\mathrm{abs}(\lvert d_1\rvert - \lvert d_2\rvert) \leq \lvert\lambda\rvert \leq \lvert d_1\rvert + \lvert d_2\rvert$	
	$-d_2/d_1 \leq \tan(\lambda) \leq d_2/d_1$	
	$0 \leq \lambda \leq 2\pi$	
	$-\sin(\lambda) \leq d_2/d_1 \leq \sin(\lambda)$	$d_1 \geq d_2$
	$0 \leq \lambda \leq 2\pi$	$d_1 < d_2$
	$d_1 \sin(\alpha) \leq \lambda \leq \infty$	

Table 1: Feasibility conditions for some basic construction steps. λ is the variant parameter.

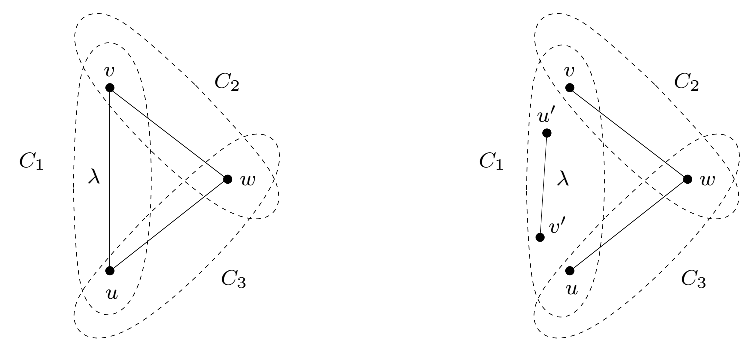


Figure 9: Left) Directly dependent problem. Right) Indirectly dependent problem.

lems from these feasibility rules is straightforward.

Definition 3.4 Let $\Pi =< G, C, P >$ be a constraint problem with clusters C_1, C_2 and C_3. Let u, v and w be the hinges with $u, v \in C_1$, $v, w \in C_2$ and $w, u \in C_3$. Let the variant parameter λ be such that the corresponding constraint is defined upon two diferent geometric elements in C_1 such that at least one of them is not a hinge, u or v. We say that problem Π depends indirectly on λ.

3.3 The Variant Parameter and Decomposition Subtrees

Here we define the concepts of decomposition subtree that fixes the value of the variant parameter λ and the set of signs significative to λ.

Definition 3.5 Let $\Pi_\lambda =< G, C, P >$ be a geometric constraint problem vith variant parameter $\lambda \in P$ whose constraint is defined over geoms $u, v \in G$. Let T_λ be a decomposition tree that solves Π_λ. Let C_λ be the smallest cluster in T_λ such that places geoms u, v with respect to a given framework. We define the decompostion tree that fixes the value of λ as the subtree of T_λ whose root is C_λ.

The set of signs occurring in the decomposition tree that fixes the value of λ, will play a central role in solving problems that depend indirectly on the variant parameter. We define this set as follows.

Definition 3.6 Let $\Pi_\lambda =< G, C, P >$ be a geometric constraint problem vith variant parameter λ and let T_λ be a decomposition tree that solves Π_λ. Let T'_λ be the subtree that fixes the value of λ in T_λ. The significative index associated with λ is the set of signs occuring in the decomposition subtree T'_λ. We denote this index as I_λ.

Since each sign in the index entails a possible different placement for the geometric elements, the significative index associated with the variant parameter, I_λ, fixes the maximum number of different actual constructions for cluster C_λ, the root of the subtree that fixes λ.

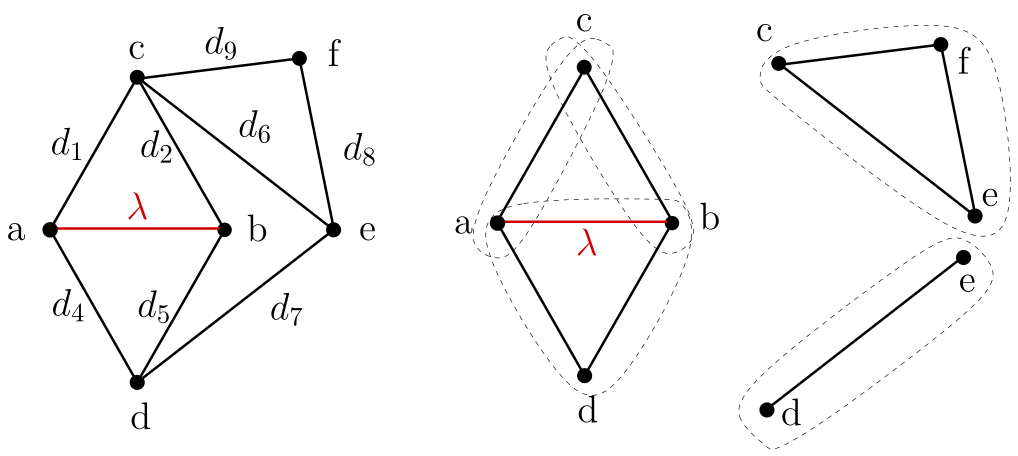


Figure 10: Computing critical points. Subproblems $\{a, b, d\}$ and $\{a, b, c, d\}$ directly depend on the variant parameter λ. Subproblem $\{a, b, c, d, e, f\}$ indirectly depends on λ.

4 The van der Meiden Method

The van der Meiden method to compute the domain of the variant parameter has two steps, [24]. In the first step the critical values for the variant parameter, that is, variant parameter values for which the construction plan feasibility might change, are computed. Then, in each interval defined by two subsequent critical values, pick a value for the variant parameter and determine whether the construction plan is feasible. The variant parameter's domain is the union of the intervals where the construction plan is feasible.

The approach relies on the fact that, under the assumption that the variant parameter continuously changes within the interval bounded by two subsequent critical points, construction plan feasibility does not change. Therefore, checking for feasibility in one point within the interval is sufficient to establish feasibility over the whole interval.

Because only one variant parameter is considered, degenerate situations appears only on those subproblems that depend on the variant parameter. For a subproblem that depends directly on the variant parameter, critical values are computed applying a dictionary that collects for each degenerate subproblem case an specific solution method. For example, for a triangular problem consisting on three points pairwise constraint with a point-point distance, the triangular inequality is applied. Figure 10 illustrates the case for the subproblems $\{a, b, c, d\}$ (clusters in the middle) and $\{a, b, c, d, e, f\}$ (clusters in the middle plus clusters on the right) with variant parameter λ.

If distance constraints take values $d_1 = 3, d_2 = 3, d_4 = 3.5, d_5 = 3.5, d_6 = 4, d_7 = 4.5, d_8 = 4, d_9 = 3.5$, the critical values for the variant parameter $\lambda = d_3$ in direct dependent problems $\{a, b, d\}$ and $\{a, b, c, d\}$ are computed applying the rules in the dictionary. Resulting critical values are respectively $\{0, 6\}$ and $\{0, 7\}$.

When a subproblem depends indirectly on the variant parameter, computing the relation

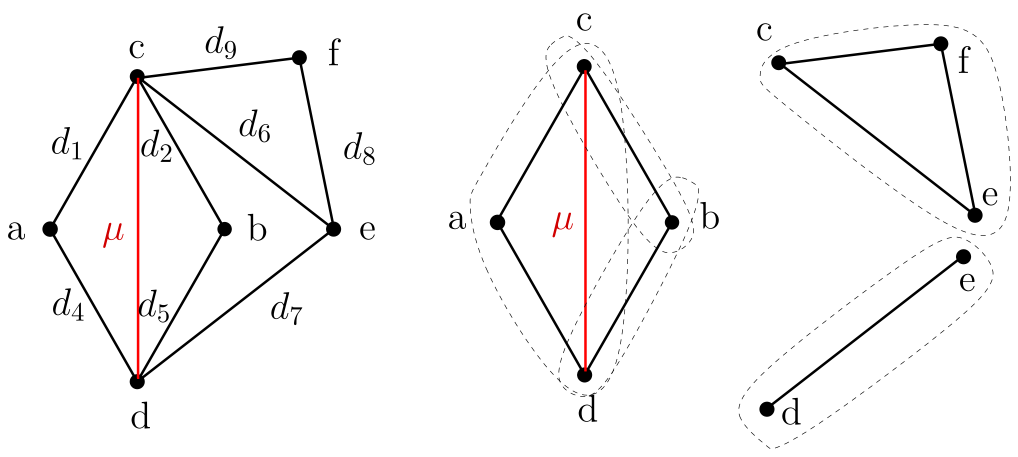


Figure 11: Modified problem. Subproblem $\{a, b, c, d, e, f\}$ now directly depends on the variant parameter μ.

between the critical variant parameter values and the solutions of the degenerate problem is more complex and they are computed indirectly. First the indirect dependency is transformed into a direct one by removing the constraint corresponding to the variant parameter and adding a new constraint that captures the degeneracy in the current subproblem. Then the modified problem is solved and critical values for the original variant parameter are measured in the solution built for the critical values of the modified problem.

If the variant parameter is λ as illustrated in Figure 10 and the subproblem currently under construction is $\{a, b, c, d, e, f\}$, the modified problem is the one shown in Figure 11. Notice that now the subproblem depends directly on the new variant parameter μ and it can be solved by applying the dictionary. Critical values for λ are obtained by measuring the parameter λ in the solution to the modified problem for critical values of μ. Figure 12 Left shows values measured for λ as the variant parameter μ changes in the modified problem for the example at hand. For the actual parameters values and according to the rules in the dictionary, critical values of μ are $\{0.5,\ 6.5,\ 8.5\}$. Corresponding values measured for λ are 0 if μ is either 0.5 or 6.5. For $\mu = 8.5$ the construction plan for the modified problem is not feasible thus λ cannot be measured and does not lead to any critical point.

The resulting set of critical values are then $\{0,\ 6,\ 7\}$. Cheking now construction plan feasibility for values of λ at the critical values plus, for example, at 3, 6.5 and 7.5, results in the feasible domain]0,6] depicted in thick line in Figure 12 Right.

The main drawback of the method is that the constructive solver considered does not need to be able to solve the transformed problem. In this case, either a different solving approach is applied to solve the transformed problem or the method fails.

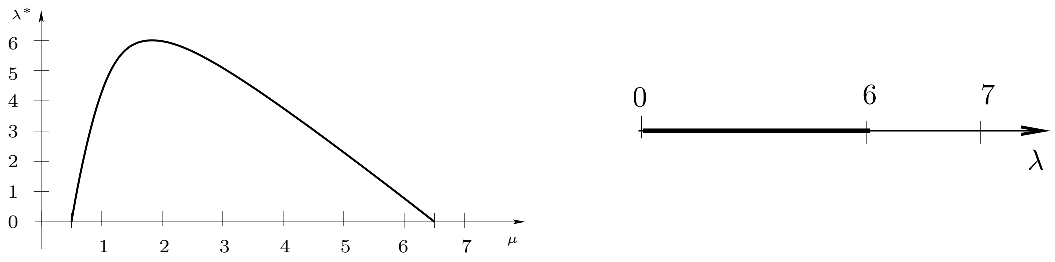


Figure 12: Left) Measured values for λ as μ changes. Right) Critical points. The domain of variant parameter λ for which the construction plan is feasible shown as a thick line.

5 Our Implementation

Let T be the decomposition tree that solves the constraint problem at hand. In our implementation, each node in T stores:

1. T.*built*: A boolean flag that takes value true whenever the coordinates of the geometric object have been actually computed with respect to a local framework,
2. T.*coordinates*: An array of coordinates that places every geometric object with respect to a local framework,
3. T.*rule*: An identifier for the solving rule. If the node is a leaf, the rule is a basic placement. Otherwise the rule identifies the merging of three clusters, and
4. T.*hinges.u*, T.*hinges.v*, T.*hinges.w*: Pointers to the hinges in the set of geometric objects on which the merging was carried out.

Without loss of generality and for the sake of simplicity and convenience, we assume that the variant parameter is always defined upon hinges u and v in the leftmost cluster in each set of sibling nodes of T. Since we consider problems with just one variant parameter and the order in which siblings are depicted in the decomposition tree is meaningless, this assumption just implies that the tree has been conveniently rewritten.

We asssume that the set of geometric elements, G, the set of constraints, C, the vector of parameters in C, P, the index in P of the variant parameter, i, and, the decomposition tree of the problem being solved, T are stored as static variables. Moreover, T.*built* value is true for the leaf nodes and false otherwise. The algorithm we have implemented is given in Algorithm 1 and Algorithm 2.

6 Algorithm Correctness

In this section we show the correctness of our algorithm to compute the domain of the variant parameter in a geometric constraint solving problem with one variant parameter.

Algorithm 1 Computing de Domain

```
K : Empty set of critical values
D : Empty set of intervals of feasible values
function VariantParameterDomain()
ComputeCriticalPoints(T)
for each subsequent interval [c1, c2] in K do
  p_i := (c1 + c2)/2
  if ConstructionPlanFeasible(T) then
    D := D + [c1, c2]
    Check for feasibility at interval bounds
  end if
end for
endfunction
```

6.1 The Transformation

We start by showing that the transformation defined is always possible. See Figure 13.

Theorem 6.1 Let C_1, C_2 and C_3 be the three clusters in a construction step with hinges u, v and w such that $u, v \in C_1$, $v, w \in C_2$ and $w, u \in C_3$. Let C_1 be the cluster dependent on the variant parameter λ. If the construction depends indirectly on λ then no constraint is defined between hinges u and v in C_1.

Proof
For a contradiction assume that there is a constraint in C_1 defined between hinges u and v. Since the problem is wellconstrained, clusters $C_1' = \{u, v\}$, C_2 and C_3 define a rigid object. Therefore merging C_1, C_2 and C_3 does not depend on λ. □

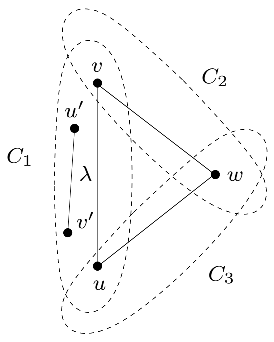


Figure 13: In wellconstrained problems, no constraint is defined on the hinges in cluster C_1 dependent on the variant parameter λ.

Algorithm 2 function ComputeCriticalPoints(T)

```
if not T.left.built then
  ComputeCriticalPoints(T.left)
else
  if DirectDependence(T.hinges.u, T.hinges.v, i) then
    K := K + DictionaryCriticalPoints(T.rule, i)
  else
    d := DummyParameter(T.hinges.u, T.hinges.v)
    Q := DictionaryCriticalPoints(T.rule, d)
    C':= $\{C - c_i\} \cup \{c_d\}$
    P':= $\{P - p_i\} \cup \{d\}$
    T':= SubtreeThatFixesTheVariantParameter(G, C', P')
    I := Significative index associated with the dummy parameter d
    for each q in Q do
      for each possible assignment to signs in I do
        R := EvaluateTree(T')
        K := K + Measure(R, i)
      end for
    end for
  end if
  T.built := true
end if
endfunction

function DummyParameter(u, v)
if IsApoint(u) and isApoint(v) then
  return point-point-distance
else if (IsApoint(u) and isAline(v)) or (IsApoint(v) and isAline(u)) then
  return point-line-distance
else if IsAline(u) and isAline(v) then
  return line-line-angle
end if
endfunction
```

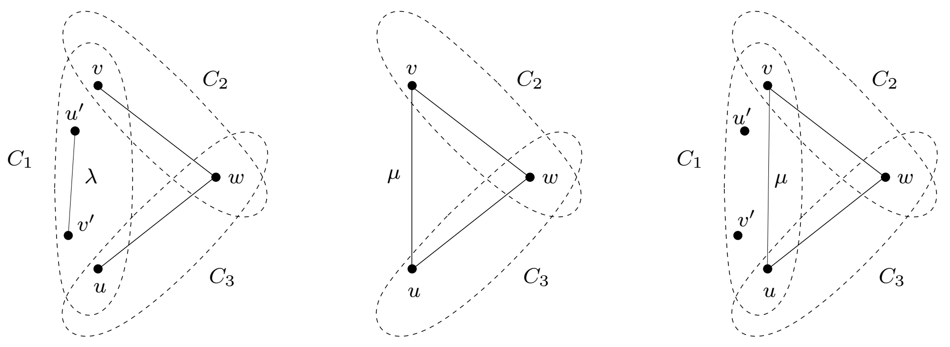


Figure 14: Left) Problem that depends indirectly on λ. Middle) Transformed problem that depends directly on μ. Right) Problem where values for the variant parameter λ are measured.

6.2 The Set of Solution Instances

Now we show that the given problem and the transformed problem have the same set of solution instances.

In Section 4, we have seen that, when a problem depends indirectly on the variant parameter, the van der Meiden method computes ranges for feasible values of variant parameters transforming the problem by removing the variant parameter in the given problem and adding a convenient, new variant parameter. Thus, we need to prove that the sets of solution instances for the given problem and the modified problem are the same set.

Let $\Pi =< G, C, P >$ be a wellconstrained geometric constraint problem and T a decomposition tree that solves Π. We shall denote by T(P) the instance of T which is solution to Π resulting from evaluating T for the specific values in P of the parameters in the constraints C.

We start the proof by stating a trivial lemma.

Lemma 6.2 Let $\Pi =< G, C, P >$ be a wellconstrained geometric constraint problem. Let T be a decomposition tree that solves Π and let $\mathrm{T}(P)$ be a realization of the solution instance. Then for any pair of geometric objects $g_i, g_j \in G$ any relationship between them can be measured in $\mathrm{T}(P)$.

Proof
Since the problem is wellconstrained, any realization $\mathrm{T}(P)$ places all the geometric elements in G with respect to a common reference. $\square$

Next we state and prove a lemma that relates solution instances for different geometric constraint problems defined on the same set of geometric elements for which tree decompositions are known.

Lemma 6.3 Let $\Pi_1 =< G, C_1, P_1 >$ and $\Pi_2 =< G, C_2, P_2 >$ be two wellconstrained geometric constraint problems defined on the same set of geometric elements G. Let T_1 and T_2 be tree decompositions that respectively solve problems Π_1 and Π_2 and let $\mathrm{T}_1(P_1)$ be a solution instance for the problem Π_1. If parameters in P_2 are assigned values measured in the actual solution $\mathrm{T}_1(P_1)$, then $\mathrm{T}_2(P_2)$ is a solution instance for Π_2.

Proof
By Lemma 6.2 we can measure in the actual solution instance $\mathrm{T}_1(P_1)$ a value for any constraint parameter in C_2. If values measured in $\mathrm{T}_1(P_1)$ are assigned to constraint parameters P_2, clearly $\mathrm{T}_2(P_2)$ is feasible and therefore it is a solution instance to problem Π_2. $\Box$

Finally, we have

Theorem 6.4 Let $\Pi_\lambda =< G, C_\lambda, P_\lambda >$ and $\Pi_\mu =< G, C_\mu, P_\mu >$ be two wellconstrained geometric constraint problems, defined on the same set of geometric elements G with variant parameter λ, and μ respectively, and such that $C_\lambda - \{\lambda\} = C_\mu - \{\mu\}$. Moreover, let T_λ and T_μ be decomposition trees that respectively solve Π_λ and Π_μ. Then the sets of solution instances $\mathrm{T}_\lambda(P_\lambda)$ and $\mathrm{T}_\mu(P_\mu)$ generated by arbitrarily varying λ and μ are the same set.

Proof
For each value assigned to the variant parameter of one of the problems, apply Lemma 6.3. $\Box$

6.3 Correctness

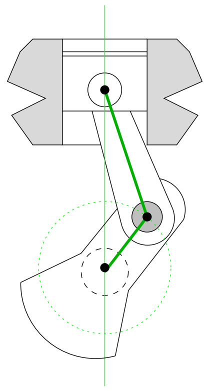
First we see that values of the variant parameter measured at critical points of the transformed problem are critical values of the initial problem.

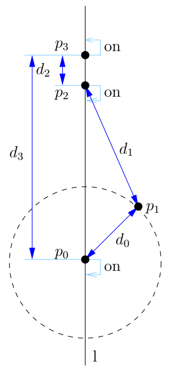
Lemma 6.5 Let $\Pi_\lambda =< G, C_\lambda, P_\lambda >$ be a problem that indirectly depends on λ. Let $\Pi_\mu =< G, C_\mu, P_\mu >$ be the problem that directly depends on parameter μ, resulting from transforming Π_λ. Let T_μ be a solution tree to Π_μ and let μ^* be a critical point of T_μ. If λ^* is the value of parameter λ measured in $T_\mu(\mu^*)$, then constructibility of T_λ changes at λ^*. That is, λ^* is a critical point for the problem Π_λ.

Proof
Let μ^* be a critical point for Π_μ and λ^* be the measure in T_μ for parameter λ. Assume that C_1, C_2 and C_3 are the three clusters involved in the indirectly dependent problem with C_1 being the cluster that undergoes the problem transformation.

Since the problem is wellconstrained, and according to Theorem 6.4, continuously changing λ in cluster C_1 of T_λ, will result in λ reaching the value λ^*. Then, μ will take the value μ^* and constructibility of T_λ will change accordingly to the constructibility of the problem defined over C_1, C_2 and C_3. Therefore λ^* is a critical point for Π_λ. $\Box$

Then we show that the algorithm figures out all the critical points and only critical





$$\begin{array}{lll}
1.\ p_0 & = & origin() \\
2.\ p_3 & = & distD(p_0, d_3) \\
3.\ l & = & line2P(p_0, p_3) \\
4.\ c_0 & = & circleCR(p_0, d_2) \\
5.\ p_2 & = & iLC(l, c_0, s_1) \\
6.\ c_1 & = & circleCR(p_0, d_0) \\
7.\ c_2 & = & circleCR(p_2, d_1) \\
8.\ p_1 & = & iCC(c_1, c_2, s_2)
\end{array}$$

c

Figure 15: a) Piston and connecting rod crankshaft. b) Geometric abstraction. c) Construction plan.

points.

Theorem 6.6 The algorithm given in Section 5 computes exactly the set of critical points of a given problem $\Pi_\lambda =< G, C, P_\lambda >$ if T_λ is a decomposition tree that solves the problem.

Proof
Since the algorithm visits each node in T_λ once and only once, each construction step is considered once and only once. $\square$

7 Case Study

To further illustrate how our algorithm works, we develop a case study, depicted in Figure 15. From left to right, the figure shows a piston and connecting rod crankshaft, an abstraction of the piston represented as a geometric constraint problem, and an actual construction plan. The set of geometric elements includes four points $\{p_0, p_1, p_2, p_3\}$ and a straight line l. The set of constraints includes four point-point distances, $d(p_1, p_0) = d_0$, $d(p_1, p_2) = d_1$, $d(p_2, p_3) = d_2$, and $d(p_0, p_3) = d_3$; and three point-on-line constraints $onPL(p_0, l)$, $onPL(p_2, l)$ and $onPL(p_3, l)$. We consider as variant parameter λ the distance d_2.

Figure 16 a) shows the graph of the geometric constraint problem $\Pi_\lambda =< G, C, P >$, and Figure 16 b) is the tree decomposition T_λ of a construction plan that solves the problem. Since we consider that an intended solution for the problem has been selected, signs in the index $I = \{s_1, s_2\}$ have been fixed.

Clusters in the decomposition tree T_λ the construction of which depend on λ are

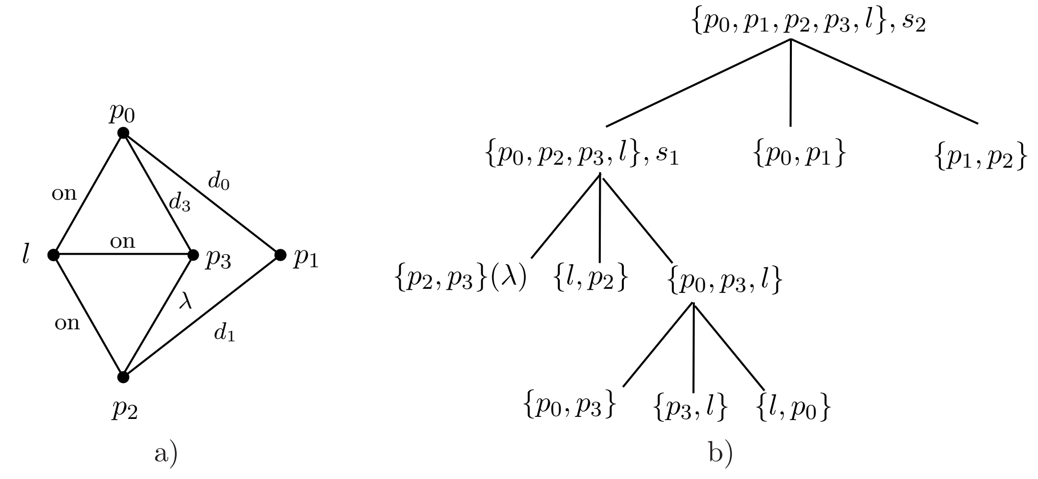


Figure 16: Piston and connecting rod crankshaft. a) Problem graph. b) Tree decomposition that solves the problem.

$\{p_0, p_2, p_3, l\}$, and $\{p_0, p_1, p_2, p_3, l\}$. Cluster $\{p_0, p_2, p_3, l\}$ directly depends on the variant parameter λ, so the dictionary provides the critical values. The construction places point p_2 on the line through points p_0 and p_3 and at distance λ from p_0. Since we do not consider signed distances, the construction is clearly feasible for $0 \leq \lambda < \infty$.

Cluster $\{p_0, p_1, p_2, p_3, l\}$ depends indirectly on the variant parameter λ. Thus the problem is transformed by replacing the constraint $\lambda = d(p_2, p_3)$ with $\mu = d(p_2, p_0)$. The graph of the transformed problem Π_μ and the corresponding decomposition tree that solves it are shown in Figure 17.

When solving the transformed problem, cluster $\{p_0, p_2, p_3, l\}$ depends directly on μ. Note that the problem again is feasible for $0 \leq \lambda < \infty$. Cluster $\{p_0, p_1, p_2, p_3, l\}$ also depends directly on the variant parameter μ. According to the dictionary of cricial points, Π_μ is feasible if $|d_1 - d_0| \leq \mu \leq |d_1| + |d_0|$. Considering, for example, specific parameters values $d_0 = 5$, $d_1 = 8$ and $d_3 = 14$, we have that the transformed problem is feasible if $3 \leq \mu \leq 13$.

Figure 18 shows on the left a construction plan for the transformed problem, Π_μ, and on the right an actual geometric construction. Values for signs have been fixed for the original problem but they are unknown in the transformed problem.

The variant parameter λ in the original problem is defined over points p_2 and p_3 and the smallest cluster in the transformed problem that includes these points is $\{p_0, p_2, p_3, l\}$, which is built in the construction step number 5, $p_2 = iLC(l, c_0, s_1)$. See Figure 18a. Therefore the significative index associated with μ is $I_\mu = \{s_1\}$, and, in general, there will be two different possible placements for point p_2, corresponding to the two possible intersections of circle c_0 with line l. One construction places point p_2 on the same side than p_3 with respect to point p_0. The other places p_2 opposite to p_3 with respect to p_0.

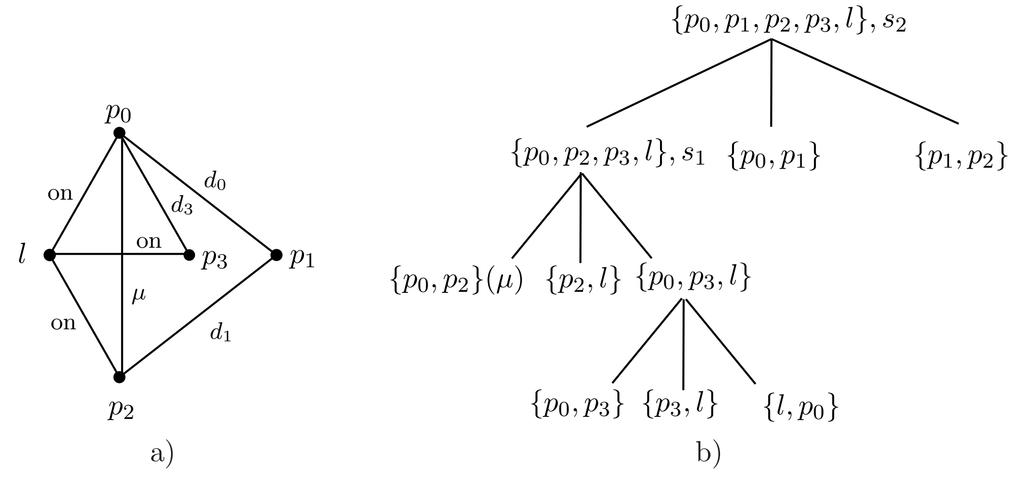


Figure 17: Piston and connecting rod crankshaft. Transformed problem. a) Problem graph. b) Tree decomposition that solves the problem.

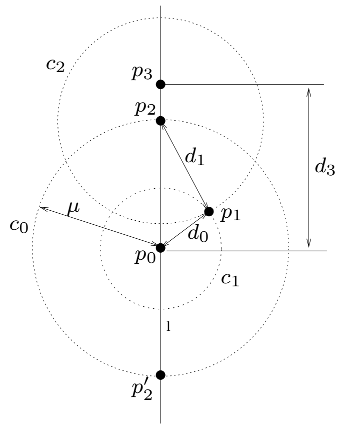
See Figure 18b. Thus, there are two different measures for the variant parameter λ for each critical value of μ.

In the case we are considering, measures for λ taken in $\mathrm{T}_\mu(3)$ are 11 and 17. Values measured for λ from $\mathrm{T}_\mu(13)$ are 1 and 27. Therefore, the set of critical values for the variant parameter λ is $\{0, 1, 11, 17, 27, \infty\}$.

Checking feasibility of T_λ at the critical values and at, say, $\lambda \in \{0.5, 5, 14, 22, 28\}$ we

1. $p_0 \quad = \quad origin()$
2. $p_3 \quad = \quad distD(p_0, d_3)$
3. $l \quad = \quad line2P(p_0, p_3)$
4. $c_0 \quad = \quad circleCR(p_0, \mu)$
5. $p_2 \quad = \quad iLC(l, c_0, s_1)$
6. $c_1 \quad = \quad circleCR(p_0, d_0)$
7. $c_2 \quad = \quad circleCR(p_2, d_1)$
8. $p_1 \quad = \quad iCC(c_1, c_2, s_2)$

a



b

Figure 18: Piston and connecting rod crankshaft. Transformed problem. a) Construction plan. b) Geometric realization.

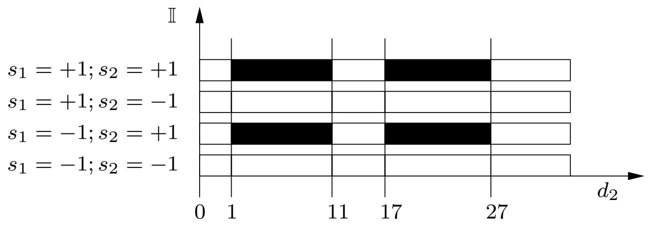


Figure 19: Piston and connecting rod crankshaft. Feasible domain for variant parameter $\lambda = d_2$.

find that the construction plan is feasible at the critical values and at $T_\lambda(5)$ and $T_\lambda(22)$. Therefore the domain for the problem Π_λ is $[1, 11] \cup [17, 27]$.

The index of the construction plan shown in Figure 15 that solves the piston and connecting rod crankshaft problem includes two signs $I = \{s_1, s_2\}$, hence up to four different intended solucion instances can be selected. Applying our algorithm to each of them yields the feasibility domain depicted in Figure 19 where feasible intervals are filled in black. Notice that for signs assigment $s_1 = +1, s_2 = -1$ and $s_1 = -1, s_2 = -1$ there are no feasible values.

8 Summary

Methods to figure out ranges of parameters can be of much help in, for example, parametric solid modeling systems and in applications of dynamic geometry. Changing parameter values in parametric solid modeling is at the heart of the technology. Therefore having a correct way to figure out ranges is paramount. In current dynamic geometry systems, behavior discontinuities due to the presence of critical points result in unpleasant or undesired behavior. In this context, computing before hand the ranges of parameters would allow preventing such behavior and providing the user with useful information that would help in understanding the insights of geometry.

In this work we have considered the technique reported in [24] to compute ranges of variant parameters. We offer a formalization of the concepts involved, our own implementation as well as a proof of correctness for it.

Acknowledgements

This research has been partially funded by the Spanish Ministerio de Educación y Ciencia and by FEDER under grant TIN2007-67474-C03-01.

References

[1] B. Aldefeld. Variation of geometric based on a geometric-reasoning method. *Computer-Aided Design*, 20(3):117–126, April 1988.

[2] W. Bouma, I. Fudos, C. Hoffmann, J. Cai, and R. Paige. Geometric constraint solver. *Computer Aided Design*, 27(6):487–501, June 1995.

[3] B.D. Brüderlin. Symbolic computer geometry for computer aided geometric design. In *Advances in Design and Manufacturing Systems*, Tempe, AZ, Jan. 8-12 1990. Proceedings NSF Conference.

[4] B. Denner-Broser. *Tracing-Problems in Dynamic Geometry*. PhD thesis, Fachbereich Mathematik und Informatik der Freien Universität Berlin, 2008.

[5] M. Freixas, R. Joan-Arinyo, and A. Soto-Riera. A constraint-based dynamic geometry system. *Computer-Aided Design*, 2010. DOI:10.1016/j.cad.2009.02.016.

[6] I. Fudos and C.M. Hoffmann. A graph-constructive approach to solving systems of geometric constraints. *ACM Transactions on Graphics*, 16(2):179–216, April 1997.

[7] C.M. Hoffmann and R. Joan-Arinyo. A brief on constraint solving. *Computer-Aided Design and Applications*, 2(5):655–663, 2005.

[8] C.M. Hoffmann and K.J. Kim. Towards valid parametric CAD models. *Computer-Aided Design*, 33(1):376–408, 2001.

[9] C.M. Hoffmann, A. Lomonosov, and M. Sitharam. Decompostion Plans for Geometric Constraint Problems, Part II: New Algorithms. *Journal of Symbolic Computation*, 31:409–427, 2001.

[10] C.M. Hoffmann, A. Lomonosov, and M. Sitharam. Decompostion Plans for Geometric Constraint Systems, Part I: Performance Measurements for CAD. *Journal of Symbolic Computation*, 31:367–408, 2001.

[11] C. Jerman, G. Trombettoni, B. Neveu, and P. Mathis. Decomposition of geometric constraint systems: A survey. *International Journal of Computational Geometry and Applications*, 16:379–414, 2006.

[12] R. Joan-Arinyo, M.V. Luzón, and A. Soto. Genetic algorithms for root multiselection in constructive geometric constraint solving. *Computer & Graphics*, 27(1):51–60, 2003.

[13] R. Joan-Arinyo and N. Mata. Applying constructive geometric constraint solvers to geometric problems with interval parameters. *Nonlinear Analysis*, 47(1):213–224, 2001.

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

[15] R. Joan-Arinyo, A. Soto-Riera, S. Vila-Marta, and J. Vilaplana. On the domain of constructive geometric constraint solving techniques. In R. Duricovic and S. Czanner, editors, *Spring Conference on Computer Graphics*, pages 49–54, Budmerice, Slovakia, April 25-28 2001. IEEE Computer Society, Los Alamitos, CA.

[16] J.C. Owen. Algebraic solution for geometry from dimensional constraints. In R. Rossignac and J. Turner, editors, *Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 397–407, Austin, TX, June 5-7 1991. ACM Press.

[17] E. Yeguas R. Joan-Arinyo, M.V. Luzón. Search space pruning to solve the root identification problem in geometric constraint solving,. *Computer-Aided Design and Applications*, 6, 2009.

[18] S. Raghothama and V. Shapiro. Necessary conditions for boundary representations variance. In *Proceedings of the 13th International Annual Symposium on Computational Geometry*, pages 77–86, New York, 4-6 June 1997. ACM Press.

[19] S. Raghothama and V. Shapiro. Boundary representation deformation in parametric solid modeling. *ACM TRansactions on Graphics*, 17(4):259–286, 1998.

[20] S. Raghothama and V. Shapiro. Consistent updates in dual representation systems. In *Proceedings of the Fith Symposium on Solid Modeling and Applications*, pages 65–75, New York, 9-11 June 1999. ACM Press.

[21] V. Shapiro and D.L. Vossler. What is a parametric family of solids? In *Proceedings of the Third Symposium on Solid Modeling and Applications*, pages 65–75, Salt Lake City, 17-19 May 1995. ACM Press.

[22] S.E.B. Thierry. *Décomposition et paramétrisation de systèmes de contraites géométriques sous-constraints*. PhD thesis, Université de Strasbourg, 2010.

[23] H.A. van der Meiden and W.F. Bronsvoort. An efficient method to determine the intended solution for a system of geometric constraints. *International Journal of Computational Geometry*, 15(3):79–98, 2005.

[24] H.A. van der Meiden and W.F. Bronsvoort. A constructive approach to calculate parameter ranges for systems of geometric constraints. *Computer-Aided Design*, 38:275–283, 2006.