Dynamic Geometry Based on Geometric Constraints

M. Freixas, R. Joan-Arinyo, A. Soto, S. Vila
Grup d'Informàtica a l'Enginyeria
Universitat Politècnica de Catalunya
Diagonal 647, 08028 Barcelona, Catalunya

Abstract

Dynamic geometry systems are tools for geometric visualization. They allow the user to define geometric elements, establish relationships between them and explore the dynamic behavior of the remaining geometric elements when one of them is moved. The main problem in dynamic geometry systems is the ambiguity that arises from operations which lead to more than one possible solution. While the user is defining the geometric construction, he is responsible to resolve these ambiguities. However, when the user is dragging a geometric element, the system is responsible to choose the intended solution, that is, the same solution that the user would select if we could ask him again.

Most dynamic geometry systems deal with this problem in such a way that the solution selection method leads to a fixed dynamic behavior of the system. This is specially annoying when this behavior is not the one the user intended.

In this work we propose an architecture for dynamic geometry systems built upon a set of functional units which will allow to apply some well known results from the Geometric Constraint Solving field. A functional unit called *filter* will provide the user with tools to unambiguously capture the expected dynamic behavior of a given geometric problem.

Keywords Dynamic geometry, Geometric constraint solving.

1 Introduction

Dynamic geometry systems are tools for geometric visualization. They allow the user to define geometric elements, establish relationships between them and explore the dynamic behavior of the remaining geometric elements when one of them is moved. According to Winroth [23], dynamic geometry systems are classified into two categories: constructive and constraint-based.

Constructive dynamic geometry systems are primarily intended to replace paper, pencil, ruler and compass with equivalent computer tools. They work by recording the way the user constructs new geometric elements, for example points, lines or circles, applying geometric operations, like intersection or perpendicular, to existing geometry. Then, the user can pick a geometric element, move it and see how the construction changes. Sketchpad [9], Cabri

Géomètre [2, 14], Cinderella [13] and pdb [23] are examples of constructive dynamic geometry systems.

Kortenkamp [13] and Winroth [23] identified the main problem in constructive dynamic geometry systems: the ambiguity. Ambiguity arises from operations like the intersection of a line and a circle or two circles which lead to more than one possible solution. While the user is defining the construction, he is responsible to resolve these ambiguities. Usually, the user selects interactively the intended solution among all the possible solutions. However, when the user is dragging a geometric element, the system is responsible to choose the intended solution, the same solution that the user would select if we could ask him again. This is the core problem of dynamic geometry: to find a well-defined method for handling ambiguities while some parameters of a construction are changed continuously. Most of dynamic geometry systems deal with this problem in such a way that the solution selection method leads to a fixed dynamic behavior of the system. This is specially annoying when this behavior is not the behavior intended by the user.

Although constructive dynamic geometry systems are mainly used in teaching geometry, they have also been proved useful in simulating physical motion, in particular the simulation of the motion of mechanical linkages. According to González-López [6], in order to simulate the behavior of a mechanism in a dynamic geometry environment there are three different phases:

Geometric: A geometric model for the mechanism is discovered, so that the geometric elements and the relationships between them that define the geometric properties of the mechanism are enumerated.

Algorithmic: A geometric construction that fulfil the geometric properties described in the geometric phase is selected.

Testing: The behavior of the mechanism is tested using simulation or dragging modes of the constructive dynamic geometry system.

The algorithmic phase can lead to different geometric constructions that fulfil the same geometric properties. Note that if a dynamic geometry system has a fixed dynamic behavior, the only way to get a different dynamic behavior is to select another geometric construction. Since the behavior of the mechanism in the testing phase is not easily deduced from the geometric construction, it is difficult to choose the geometric construction that captures the intended behavior among all the possible geometric constructions that describe the geometry of the mechanism. This problem is extensively analyzed in [6].

In constraint-based dynamic geometry systems the user first draws an approximate sketch and then specifies how geometric elements in the sketch are related by geometric constraints, like distance between points, angle between lines, tangency or incidence. Then, the system computes a new sketch that satisfies all the constraints. The user can change the value of a constraint and see how the geometric elements move to satisfy the new value of the constraint. According to Winroth, GéoSpécif [1], UniGéom [3] and Juno-2 [7] are examples of constraint-based systems written specifically for dynamic geometry. Among the constraint-based systems not specifically written for dynamic geometry, see [8], we are interested in constructive geometric constraints solvers. This systems work by analyzing a geometric constraints problem that describes the geometric elements involved in the problem and the constraints that they

must fulfil, and producing a construction plan, a construction analogous to the user-defined construction in the constructive dynamic geometry systems. Then, the intended solution is selected among all the possible solutions and the construction plan is evaluated leading to a realization, an assignment of coordinates to the geometric elements in the problem that fulfil the constraints. Constructive geometric constraints solvers devotes a component or functional unit, the index selector, to deal with the problem of ambiguity.

In this work we discuss a set of new components or functional units such that will allow us to apply some well known results from the Computer-Aided Design field to build efficient and effective dynamic geometry systems. For example, Constructive geometric constraints solvers outperform constructive dynamic geometry systems when modelling mechanical linkages because the properties of a mechanism stated in the geometric phase can be easily mapped to geometric constraints. In addition, since the construction plan is computed automatically by the system, the algorithmic phase is no longer needed. We also focus on techniques to define different possible dynamic behaviors for a given geometric constraints problem, so that the user can, at least, choose among different behaviors.

The paper is structured as follows. Section 2 presents the architecture for constructive geometric constraint solvers on which the new developments are based. Section 3 discusses semantic issues concerning the dynamic behavior in constructive geometric constraint solving. In Section 4 we extend the architecture of a constructive geometric constraint solver in order to simulate motion of mechanical linkages. Section 5 deals with implementation issues. Section 6 reports the first results from the prototype implementation. Finally, Section 7 draws some conclusions and, Section 8 point out future work.

2 Architecture for a Constructive Solver

In this section we present a general architecture for constructive geometric constraint solvers. See [11, 22] for an in depth presentation. Figure 1 shows a data-flow diagram of the architecture where the rounded boxes represent *data entities* and square boxes represent *functional units*. The data entities are the geometric constraints problem, the construction plan, the parameters assignment and the index assignment. The functional units are the analyzer , the index selector and the constructor .

First, the user defines a *geometric constraints problem* that is a tuple $A = \langle G, C, P \rangle$ where G is a set of geometric elements, C is a set of constraints defined between elements of G, and P is the set of parameters in C. G is a set of symbols denoting geometric elements, for example points and lines in two dimensions. C is a set of first order logic predicates denoting constraints which can be dimensional or topological. Dimensional constraints are drawn from distance between two points, the perpendicular distance between a point and a line and angle between two lines. Coincidences are considered topological constraints. P is a set of symbols denoting the value of a dimensional constraint in C.

Example 1 Figure 2 on the left shows an idealized piston, connecting rod and crankshaft and on the right a geometric abstraction defined with constraints. It can be seen as an abstract problem $A = \langle G, C, P \rangle$ where the set of geometric elements is $G = \{p_0, p_1, p_2, p_3, l\}$

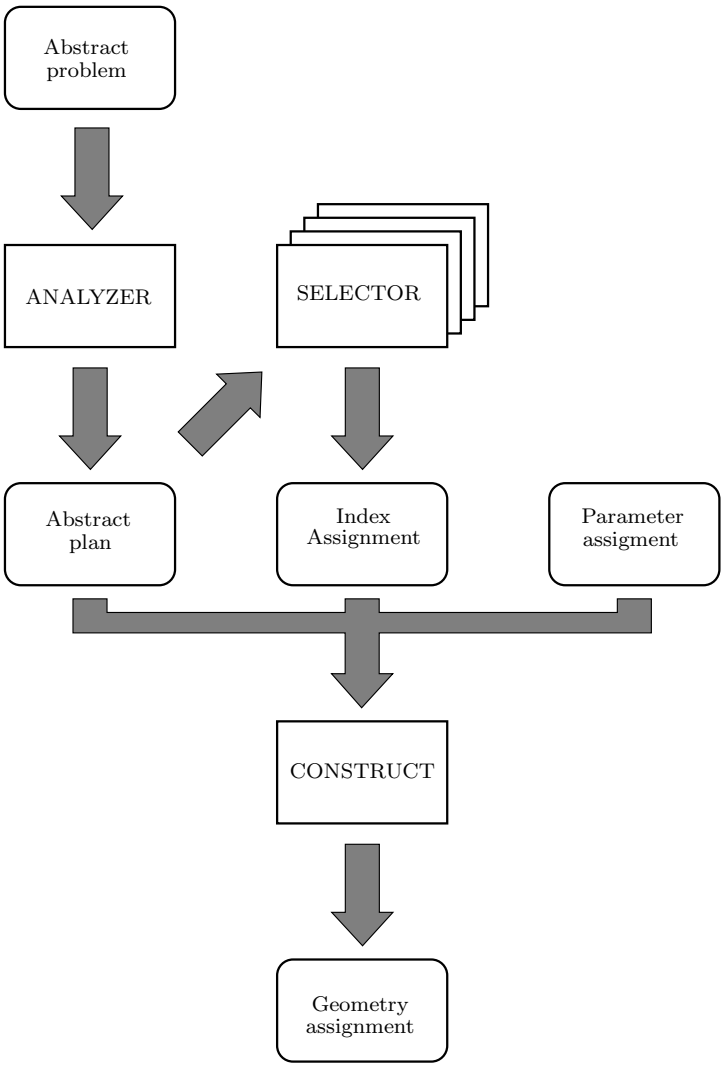


Figure 1: Architecture data-flow diagram.

the set of constraints is

$$\begin{aligned} C \;=\; \{ & \\ & d_0 = distPP(p_0, p_1), \\ & d_1 = dispPP(p_1, p_2), \\ & d_2 = distPP(p_2, p_3), \\ & d_3 = distPP(p_0, p_3), \\ & onPL(p_0, l), \\ & onPL(p_2, l), \\ & onPL(p_3, l) \\ \} & \end{aligned}$$

where $onPL(p, l)$ stands for point p is on the line l and $d = distPP(p_i, p_j)$ is the distance between two points. The set of parameters is $P = \{d_0, d_1, d_2, d_3\}$ $\diamond$

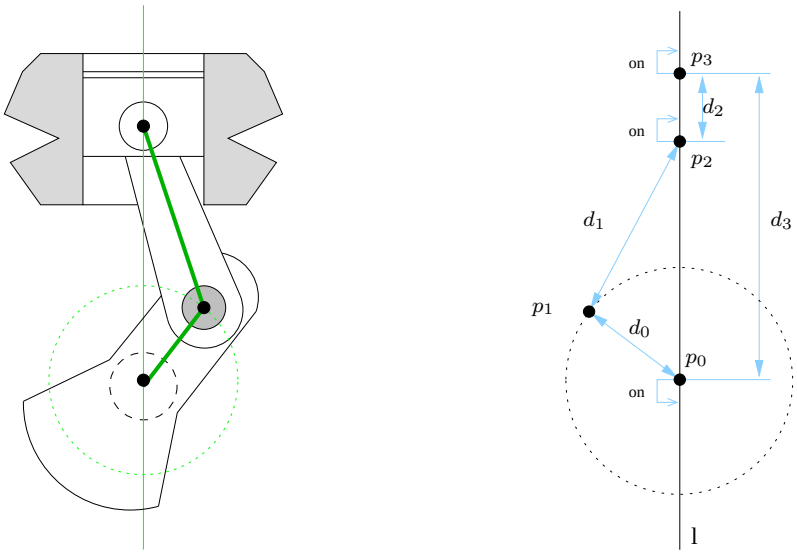


Figure 2: Left) Piston example. Right) Piston abstraction.

The user also defines an assignment of real values to parameters. Given a set of symbols S and a set of values V, a *textual substitution* α is a total mapping from S to V. A *parameters assignment* is a textual substitution α from $\mathbb{R}$ to a set of parameters P.

Example 2 An example of a parameters assignment α for the geometric constraints problem in Example 1 is

$$\begin{aligned} \alpha(d_0) &= 4 \\ \alpha(d_1) &= 6 \\ \alpha(d_2) &= 2 \\ \alpha(d_3) &= 10 \end{aligned}$$

◇

A *geometry assignment* or *anchor* κ is a textual substitution such that assigns an actual geometry to each geometric element in a set of geometry symbols G.

Example 3 If we represent a point by the pair $(x, y) \in \mathbb{R}^2$ and a straight line by (a, b, c), the coefficients of the normal form $ax + by + c = 0$ with $a^2 + b^2 = 1$, then an example of anchor κ is

$$\begin{aligned} \kappa(p_0) &= (0, 0) \\ \kappa(p_1) &= (2.9, -2.75) \\ \kappa(p_2) &= (0, -8) \\ \kappa(p_3) &= (0, -10) \\ \kappa(l) &= (0, 1, 0) \end{aligned}$$

◇

The goal of a geometric constraints solver is to compute a realization. A realization is an anchor such that satisfies the geometric constraints C in a geometric constraints problem A. Before we give a formal definition of realization we must give a definition for geometric constraints problem. Let $A = \langle G, C, P \rangle$ be an abstract geometric constraint problem with $C = \{c_1, c_2, \ldots, c_m\}$. Then the *characteristic formula* of A is the first order logic formula,

$$\Psi(A) \equiv \bigwedge_{i=1}^{m} c_i$$

where the geometric elements of G and the parameters of P occurring in Ψ are interpreted as free variables.

Example 4 The characteristic formula of the abstract problem A given in the Example 1 is

$$\begin{array}{rcl} \Psi(A) & \equiv & d_0 = distPP(p_0, p_1) \\ & \wedge & d_1 = dispPP(p_1, p_2) \\ & \wedge & d_2 = distPP(p_2, p_3) \\ & \wedge & d_3 = distPP(p_0, p_3) \\ & \wedge & onPL(p_0, l) \\ & \wedge & onPL(p_2, l) \\ & \wedge & onPL(p_3, l) \end{array}$$

$\diamond$

Let W be a set of predicates and α a textual substitution from S to V, we note by $\alpha.W$ the set of predicates obtained by replacing every occurrence in W of any symbol $s \in S$ by $\alpha(s) \in V$.

Example 5 Let C be the set of constraints in the geometric constraints problem of the Example 1 and let α be the parameters assignment in Example 2. Then, $\alpha.C$ is

$$\begin{array}{rcl} C & = & \{ \\ & & \quad 4 = distPP(p_0, p_1), \\ & & \quad 6 = dispPP(p_1, p_2), \\ & & \quad 2 = distPP(p_2, p_3), \\ & & \quad 10 = distPP(p_0, p_3), \\ & & \quad onPL(p_0, l), \\ & & \quad onPL(p_2, l), \\ & & \quad onPL(p_3, l) \\ & & \} \end{array}$$

$\diamond$

We also apply the dot operator to tuples such that a component is a set of predicates. For example, let $A = \langle G, C, P \rangle$ be a geometric constraints problem and α be a parameters assignment for P, we write $\alpha.A = \langle G, \alpha.C, P \rangle$.

Let κ be an anchor for G. The set of anchors for which the formula $\Psi(\kappa.\alpha.A)$ holds

$$V(\alpha.A) = \{\kappa \quad | \quad \Psi(\kappa.\alpha.A)\}$$

define the set of anchors which are solution to the instance geometric constraints problem $\alpha.A$. We refer to the anchors in $V(\alpha.A)$ as *realizations* of the instance problem $\alpha.A$.

A geometric constraints problem precisely describe a set of geometric elements and the constraints that they must fulfill, but geometric constraints problems do not define how to place the geometric elements to satisfy the constraints. The construction plan is the data entity that describes how to compute the position of each geometric element in the problem. Now we present more formally the analyzer , the functional unit that computes a construction plan from a geometric constraints problem, and the construction plan.

The *analyzer* is the functional unit that computes a construction plan $S = \langle G, P, L, I\rangle$ from an abstract problem $A = \langle G, C, P\rangle$. The relationship between the abstract problem A and the abstract plan S established by the definition of the analyzer is that the sets G and P are the same in A and S.

A construction plan describes how to place the geometric elements with respect to each other. More precisely, a *construction plan* is a tuple $S = \langle G, P, L, I\rangle$ where G is a set of geometric elements, P is a set of parameters, the *index* I is a set of sign parameters, and L is a set of basic construction operations parameterized by P and I. L defines how to place with respect to each other the elements in G.

Example 6 Assume that $S = \langle G, P, L, I\rangle$ is the construction plan computed by the analyzer from the geometric constraints problem in Example 1. Thus, G and P are the same as those in Example 1. The set L,

$$\begin{aligned} L \;=\; \{ & \\ & p_0 = origin(), \\ & p_3 = distD(p_0, d_3), \\ & l = line2P(p_0, p_3, s_0), \\ & c_0 = circleCR(p_3, d_2), \\ & p_2 = ilc(l, c_0, s_1), \\ & c_1 = circleCR(p_0, d_0), \\ & c_2 = circleCR(p_2, d_1), \\ & p_1 = icc(c_1, c_2, s_2) \\ \} & \end{aligned}$$

specifies how to build the geometric object illustrated in Figure 2. In L, the operation $origin()$ returns the origin of an arbitrary reference system in the plane, the operation $distD(p, d)$ returns an arbitrary point at a distance d from p, $line2P(p_i, p_j)$ returns the line through the points p_i and p_j, the operation $circleCR(p, r)$ returns the circle centered in point p and with radius r, and the operations ilc and icc return the intersections of a line and a circle, and two circles respectively.

Note that L contains auxiliary symbols, $\{c_0, c_1, c_2\}$, which do not belong to G. They are introduced to increase readability. Nonetheless, these symbols can be replaced by their definitions. For instance, symbol c_0 is defined as $c_0 = circleCR(p_3, d_2)$. If we replace c_0

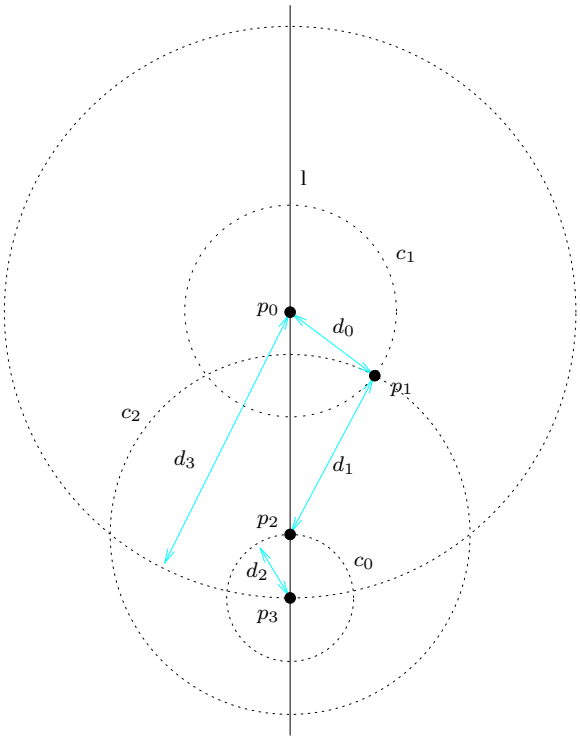


Figure 3: Step by step interpretation of the abstract plan given in Example 6.

in the definition of p_2, we have $p_2 = ilc(l, circleCR(p_3, d_2), s_1)$. This procedure can be repeated for every auxiliary symbol occurring in L.

Figure 3 step by step illustrates how the plan is interpreted. First an arbitrary point p_0 is chosen to start the construction. Then, the point p_3 is created at a distance d_3 from p_0. Next the straight line l, through points p_0 and p_3, and the circle c_0, with center on p_3 and radius d_2, are created. There are two possible orientations for the direction vector of l which are controlled by the sign s_0. The point p_2 is defined as the intersection of l and c_0. Notice that there are two possible locations for point p_2. Every possible location is distinguished with the sign s_1, which takes values in the set $\{+1, -1\}$, following the semantics of signs defined by Mata in [17]. Assuming that point p_2 is located with $s_1 = -1$, the rest of the plan is similarly computed. $\diamond$

Sign parameters in I appear in operations with more than one result. For example, the basic intersection operations involving circles may have more that one intersection point. We characterize each intersection point by using an additional *sign parameter*, s, with value in $\{+1, -1\}$. Therefore, this leads to operations like $ilc(l, c, s)$ and $icc(c_i, c_j, s)$. For a full definition of the semantics of parameter s see the work by Mata, [17].

Note that the index I is not provided by the user and that the meaning of each sign parameter is defined in the set of construction operations L, which is neither provided by the user. Thus, a geometric constraints solver needs a functional unit that computes an index assignment from the construction plan and other user-provided data, such as the initial sketch.

An *index assignment*, denoted ι, is a textual substitution from an index I to the set $\{+1, -1\}$.

Example 7 The index assignment $\iota(s_0) = +1$, $\iota(s_1) = -1$ and $\iota(s_2) = -1$ leads to the construction given in Figure 3 for the construction plan in Example 6. $\diamond$

The *index selector* is a functional unit characterized by its output which is an index assignment ι. An index assignment is always associated to a given construction plan. Therefore the construction plan must also be considered an input to the index selector. Moreover, additional input data must be considered depending on the selection method that the functional unit implements. For an extensive analysis of methods for implementing index selectors, see the work by Luzón [16].

The last step of a geometric constraints solver is the evaluation of the construction plan S given a parameters assignment α and an index assignment ι. The *constructor* is the functional unit that computes an anchor κ from an abstract plan S, a parameters assignment α and an index assignment ι. Before we give the properties of the anchor κ computed by the constructor, we must state the precise meaning of a construction plan.

A construction plan $S = \langle G, P, L, I \rangle$ with $L = \{o_1, o_2, \ldots, o_n\}$ can be interpreted as the following fist order logic formula,

$$\Phi(S) \equiv \bigwedge_{i=1}^{n} o_i$$

where the geometric elements of G, the parameters of P and signs of I occurring in Φ are considered free variables.

Example 8 The characteristic formula of the construction plan S given in Example 6 is

$$\begin{aligned}
\Phi(S) \quad \equiv \quad & p_0 = origin() \\
\wedge \quad & p_3 = distD(p_0, d_3) \\
\wedge \quad & l = line2P(p_0, p_3, s_0) \\
\wedge \quad & c_0 = circleCR(p_3, d_2) \\
\wedge \quad & p_2 = ilc(l, c_0, s_1) \\
\wedge \quad & c_1 = circleCR(p_0, d_0) \\
\wedge \quad & c_2 = circleCR(p_2, d_1) \\
\wedge \quad & p_1 = icc(c_1, c_2, s_2)
\end{aligned}$$

$\diamond$

Let α be a parameters assignment for P, then $\alpha.S = \langle G, P, \alpha.L, I \rangle$ is the instance plan. Thus the first order formula $\Phi(\alpha.S)$ expresses the instance plan.

Example 9 The characteristic formula of the instance plan in Example 8 is

$$\begin{aligned}\Phi(\alpha.S) \equiv\ & p_0 = origin() \\ \wedge\ & p_3 = distD(p_0, 10) \\ \wedge\ & l = line2P(p_0, p_3, s_0) \\ \wedge\ & c_0 = circleCR(p_3, 2) \\ \wedge\ & p_2 = ilc(l, c_0, s_1) \\ \wedge\ & c_1 = circleCR(p_0, 4) \\ \wedge\ & c_2 = circleCR(p_2, 6) \\ \wedge\ & p_1 = icc(c_1, c_2, s_2)\end{aligned}$$

$\diamond$

Let $S = \langle G, P, L, I\rangle$ be an abstract plan and κ an anchor for G. We define $\kappa.S$ as $\langle G, P, \kappa.L, I\rangle$.

Example 10 Let κ be the anchor in Example 3 and $\alpha.S$ the instance plan in Example 9. The characteristic formula Φ after applying the anchor κ to the instance problem $\alpha.S$ is

$$\begin{aligned}\Phi(\kappa.\alpha.S) \equiv\ & (0,0) = origin() \\ \wedge\ & (0,-10) = distD((0,0), 10) \\ \wedge\ & (0,1,0) = line2P((0,0), (0,-10), s_0) \\ \wedge\ & c_0 = circleCR((0,-10), 2) \\ \wedge\ & (0,-8) = ilc((0,0,0), c_0, s_1) \\ \wedge\ & c_1 = circleCR((0,0), 4) \\ \wedge\ & c_2 = circleCR((0,-8), 6) \\ \wedge\ & (2.9,-2.75) = icc(c_1, c_2, s_2)\end{aligned}$$

$\diamond$

Let κ be an anchor for G and α a parameters assignment for P. The set of anchors for which there is an index assignment ι such that the formula $\Phi(\iota.\kappa.\alpha.S)$ holds

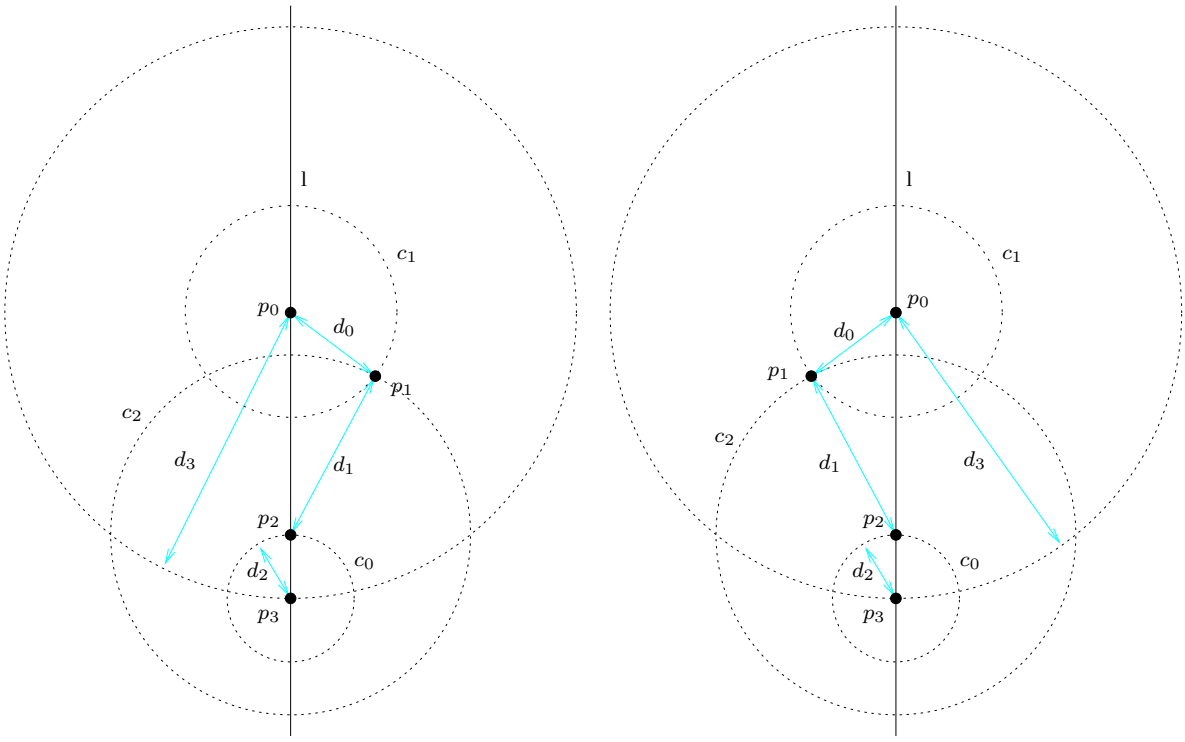
$$V(\alpha.S) = \{\kappa \quad | \quad \exists\, \iota\ \Phi(\iota.\kappa.\alpha.S)\}$$

define the set of anchors which are computed by the instance plan $\alpha.S$. We refer to the anchors in $V(\alpha.S)$ as *indexed anchors* of the instance plan $\alpha.S$.

Figure 4 shows a graphical representation of the set of indexed anchors $V(\alpha.S)$ for the instance plan $\alpha.S$ in Example 9, In Figure 4 left we have $\iota(s_2) = +1$. In Figure 4 right we have $\iota(s_2) = -1$. There are two more indexed anchors that can not be graphically distinguished, for $\iota(s_0) = +1$ and $\iota(s_0) = -1$, because these two indices only flips the orientation of the direction vector of line l. Note that for index assignments such that $\iota(s_1) = +1$, c_1 and c_2 do not intersect therefore p_1 can not be computed.

Given an index assignment ι and a parameters assignment α, there is at most one anchor κ for which $\Phi(\kappa.\iota.\alpha.S)$ holds. Thus, an index assignment uniquely identify an anchor.

Vila in [22] classifies the analyzer according to the relationship between $V(\alpha.A)$ and $V(\alpha.S)$. For example, an *ideal* analyzer computes a construction plan S from a geometric constraints



(a) $\iota(s_2) = +1$ (b) $\iota(s_2) = -1$

Figure 4: The set of indexed anchors $V(\alpha.S)$ for the instance plan in Example 9.

problem A such that the set of indexed anchors of the construction plan S and the set of realizations of the geometric constraints problem A are coincident, that is $V(\alpha.S) = V(\alpha.A)$. In what follows, we assume that we have an ideal analyzer available.

3 Dynamic Behavior

So far, we have seen solvers that work with static geometry models, that is, we just considered problems where the location of each geometric element does not change over time. In this work we want to explore the possibility of extending the scope of constructive solvers, so that they can work with dynamic geometry problems, that is, problems where location of geometric elements are allowed to change over time.

3.1 General Properties for Dynamic Geometry Systems

When using dynamic geometry sistems, the user can find that strange things happen from time to time. An object might suddenly jump into a different position or disappear completely. A group of objects may converge to the same position, [23]. It turns out that these effects are caused by a number of non-trivial mathematical issues that must be addressed. In order to

work out a satisfactory solution for these problems, an ideal dynamic geometry system must exhibit three properties: *determinism, conservatism*, and *continuity*, [13].

Determinism is the property by which, for a given problem and parameters assignment, there is at most one instance of the construction. Clearly, systems considering point and lines with construction steps where operations whith circles are not involved have determinism. If circles and, in general, conics are involved the system is, in principle, no longer determined because, for example, a stright line and a circle intersect in up to two different points.

A dynamic geometry system is conservative if when you move geometric elements and then undo all your moves by reversing them, then you get the same geometric construction. Notice that if a dynamic system is determinated, you always have the same solution instance for the same problem and parameters assignment, therefore it is also conservative. If the system is indetermined we might not expect to be conservative.

In our context, defining continuity is rather elusive and gives rise to the most challenging issues. We say that a system is continuous if small changes in parameters assignment result in small changes in the placements of the geometric elements in the problem. That is, we do not want to have large jumps of geometric elements for small changes in the parameters assignment values.

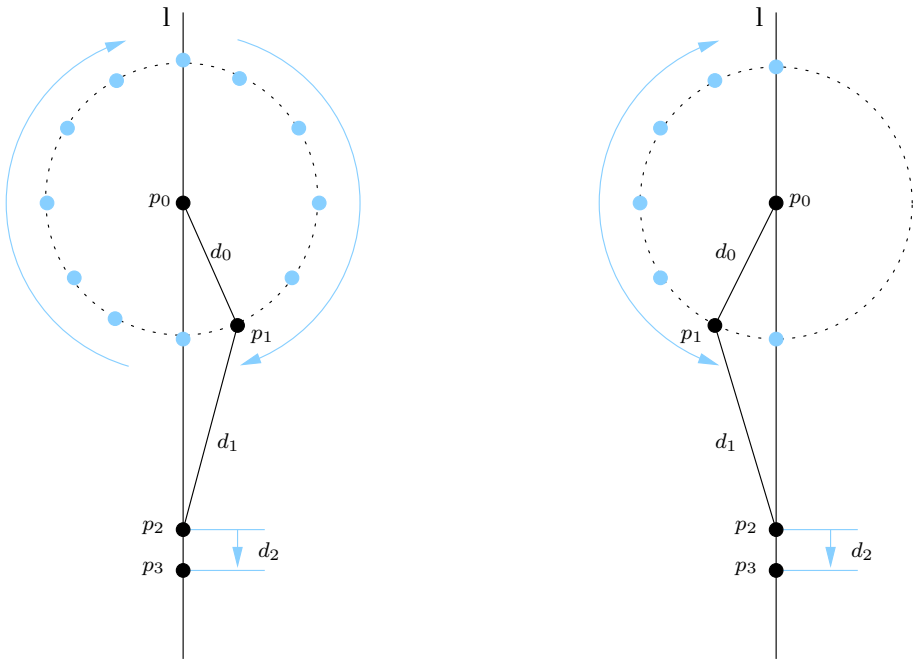
We shall come back to these issues in Section 4.

3.2 Constructive-Based Dynamic Behavior

We will build a dynamic geometry system by extending the general architecture of constructive solvers like that reported by Vila [22].

We consider a dynamic geometry environment where geometric problems have one degree of freedom. Here all the parameters but one have a given fixed value. The parameter whose value changes over time is called the *driving parameter*. Figure 2 illustrates an example of a geometric problem with one degree of freedom where the driving parameter is d_2. A simple analysis of the problem shows that the mechanism is feasible if the driving parameter takes values in the interval $[d_3 - (d_1 + d_0), d_3 - (d_1 - d_0)]$. Within this interval there is at least one solution that fulfills the constraints. In this example the interval bounds of feasible values for the driving parameter can be computed exactly, but in general this is not possible. Mata [17] gives a detailed explanation for this issue. Since a construction plan can be seen as a continuous function, whenever the geometric problem with one degree of freedom fulfills some property in the bounds of the interval of the driving parameter, this property is also fulfilled for values within the interval.

The main problem which we try to solve with the new architecture is the definition of the dynamic behavior. A dynamic behavior is a type of movement that the geometry of the geometric problem describes along the time. Most of the existing systems of dynamic geometry only allow one type of dynamic behavior. With our architecture we want to provide the user with tools to define different possible behaviors. In general, a geometric problem with one degree of freedom has multiple dynamic behaviors. For example, in Figure 5 we can see two types of dynamic behavior for the piston geometric problem. In order to be able to better understand these two dynamic behaviors we look at an specific geometric point, called *point of interest*, p_1, in Figures 2 and 5. When the driving parameter reaches one bound of the feasible



(a) Piston behavior

(b) Rocker behavior

Figure 5: Dynamic behaviors of the piston-connecting rod-crankshatf.

interval values there are two possible motions for the point of interest. One corresponds to a piston behavior where the point of interest should describe a full circular path. The other corresponds to a rocker behavior and the point of interest should flip its motion and describe an arc of circle. Notice that the dynamic behavior is defined by the path described by the point of interest. But in general, the dynamic behavior or motion may be described by a set of geometric elements. As a consequence, in general, it is not easy to choose the intended dynamic behavior.

In the piston example the position of the point of interest depends on the value of one sign in the construction plan. In the Example 6 we can see the construction plan for the geometric problem of the piston, and how the position of the point of interest depends on the sign s_1. As a consequence, the problem of selecting a dynamic behavior is closely related to the problem of selecting an index assignment. In general, there are signs whose value never change and therefore they should be kept out of the computations. Clearly, discarding as much signs as possible in the process of generating dynamic paths would be paramount. We will devote Section 4.1 to develop the functional unit called *trimmer* whose aim is to solve this issue.

Finally, the solver in a dynamic geometry system should generate a data entity that defines a dynamic behavior. This data entity, called *behavior graph*, captures the index assignment that must be applied when a transition occurs, that is, when the driving parameter arrives at one bound of the interval where it takes values. Figure 6 depicts the behavior graph for the piston behavior. The graph nodes represent index assignments. For each index assignment

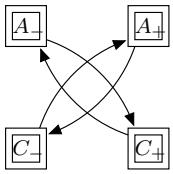


Figure 6: Example of behavior graph for the piston problem.

and each interval of feasible values, there are two nodes in the graph, one corresponds to increasing driving parameter values and the other one for decreasing values. The edges in the graph represent transitions between two index assignments. One of our goals is to pack the system that defines the dynamic behavior in a flexible data entity which could be applied in several contexts.

4 The Architecture of a Dynamic Behavior Selector

In this work we define a new solver architecture to be applied to built dynamic geometry systems for problems with one degree of freedom.

We define an architecture with four functional units or steps: the trimmer, the sampler, the router and the filter. In Figure 7 we show the architecture pipeline and how the different components are connected. The input to the pipeline is a geometric problem with one degree of freedom, defined as a construction plan $S = \langle G, P, L, I \rangle$, and a driving parameter $p \in P$. The output is a graph that captures the intended dynamic behavior. In what follows, $L_p \subseteq L$ will denote the subset of construction steps in the plan L where the parameter p is involved. Clearly, L_p must be evaluated every time the value of p changes. Let us describe in detail each functional unit.

4.1 Trimmer

The trimmer is the functional unit that computes the dependences between a parameter and the construction steps in a construction plan. Based on these dependences we will define the induced index which is the minimum set of signs involved in the dynamic behavior associated to a driving parameter. We will denote the induced index associated to the driving parameter p by the set of signs $I_p \subseteq I$ involved in L_p.

The process to compute the induced index, I_p, consists of three steps. First the directed graph which represents de dependences is computed by analyzing the geometric operations in the construction plan. In the second step the subgraph of dependences where the driving parameter is involved is determined. Finally the induced index is extracted from the subgraph of dependences.

The graph nodes represent geometric elements (points and lines), parameters and signs in the construction plan. Nodes are identified by the name of the element they represent. Graph arcs represent the dependences between two elements. The parameters and signs do not depend on any other element. Geometric elements can depend on other geometric elements,

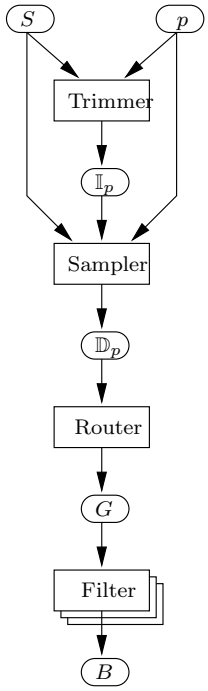


Figure 7: Architecture Pipeline.

parameters or signs. Figure 8 shows a directed graph for the dependences in the construction plan of the Example 6.

The subgraph that captures the dependences with respect to the driving parameter is the strongly connected component, [15], of the dependences graph defined by the driving parameter. Columns in Figure 9 show the strongly connected component for a different parameter in the construction plan shown in Example 6.

Finally, the induced index is computed from the strongly connected component graph of the driving parameter. The induced index consists on those sign nodes in the dependences graph such that there is an arc between them and some node in the strongly connected subgraph. For example, if d_2 is the driving parameter in the third column in Figure 9, the corresponding

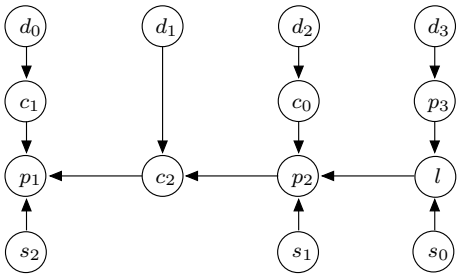


Figure 8: Graph of dependences between geometric elements, parameters and signs in a construction plan.

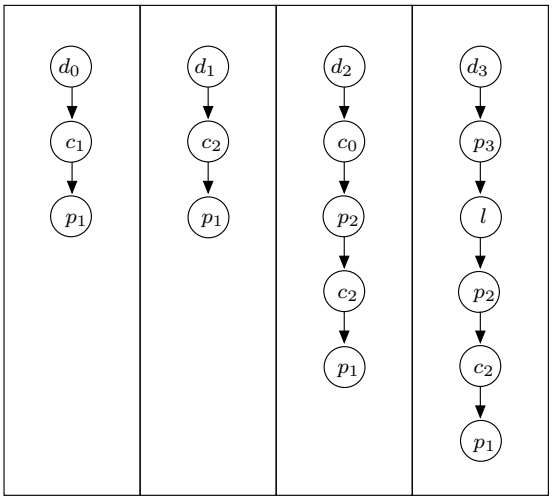


Figure 9: Strongly connected components for each parameter of the construction plan in Example 6.

induced index is $\{s_1, s_2\}$. Notice that index s_0 in the graph in Figure 8 does not belong to the strongly connected component defined by d_2.

4.2 Sampler

The evaluation of the construction plan depends on a parameters assignment and on an index assignment. The parameters assignment fix the relative position of the geometric elements. The index assignment identifies the intended solution instance among those possible solutions. In a mechanism with one degree of freedom, all the parameters are fixed but one. Clearly, the construction does not need to be feasible for all the values in the driving parameter rang. This issue must be taken into account in the evaluation process.

The aim of the sampler is to figure out the set of values of the driving parameter for which the construction plan is feasible. This set is called *domain*. With each index assignment we associate a unique domain called *associated domain*. In general, an associated domain is composed by disjoint *intervals*. Figure 10 illustrates the domain for the mechanism in Figure 2. Driving parameter values are represented on the X axis. Each row in the Y axis corresponds to an index assignment. White cells denote interval of feasible values.

Next we give some definitions that we will use in the sequel. A detailed study of the concepts presented can be found in [17]. We start with the concept of *partial assignment* for the parameters.

Definition 1 *Let $\alpha : P \to \mathbb{R}$ be a parameters assignment. Let $p \in P$ be the driving parameter and let $x \in \mathbb{R}$ be the varying value assigned to p. The partial parameters assignment $\alpha' : P \to \mathbb{R}$ for q in P is*

$$\alpha'(q) = \begin{cases} \alpha(q) & ; \forall q \in P \quad with \quad q \neq p \\ x & ; if \quad q = p \end{cases}$$

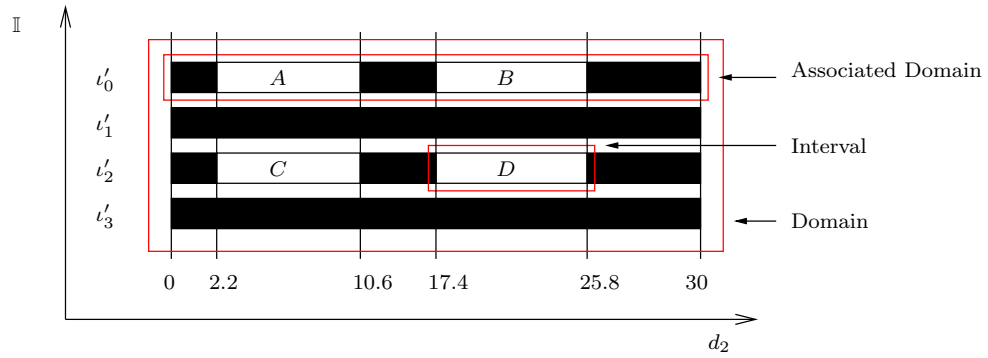


Figure 10: Domain for the piston problem using the parameter d_2 as driving parameter.

Next we define the set of index assignments for which we will explore the feasibility of the construction plan $S = \langle G, P, L, I\rangle$. Let p denote the driving parameter.

Definition 2 *Let $\iota : I \to \{+1, -1\}$ be a given signs assignment for the index I in the considered construction plan S. For the sake of readability and without loss of generality, let us write I as $I = \{s_0^p, \ldots s_{n_p-1}^p, s_{n_p}, \ldots s_n\}$ where $I_p = \{s_0^p, \ldots s_{n_p-1}^p\}$ is the index induced by p in I. The set of index assignments to be explored is*

$$\mathbb{I} = \{\iota' | \iota' = \{s_0^p, \ldots, s_{n_p-1}^p, \iota(s_{n_p}), \ldots, \iota(s_n)\} \text{ where } \forall i\ s_i^p \in \{-1, +1\}\}$$

Notice that $\mathbb{I}$ is the set of signs assignments where the signs in I_p take all the possible combinations of values in $\{-1, +1\}$ and the signs in $I - I_p$ are fixed by the initial signs assignment. Example 11 illustrates this concept.

Example 11 Consider the mechanism in Figure 2. The driving parameter is d_2, the set of signs is $I = \{s_0, s_1, s_2\}$ and the induced index is $I_p = \{s_1, s_2\}$. Assume that the initial index assigment $\iota : I \to \{+1, -1\}$ is defined by

$$\iota(s_0) = -1, \quad \iota(s_1) = +1, \quad \iota(s_2) = +1$$

Then the set of index assigments to be explored is:

$$\begin{aligned} \mathbb{I} \ = \{ \ & \{ \ +1, \ +1, \ \iota(s_0) \ \}, \\ & \{ \ +1, \ -1, \ \iota(s_0) \ \}, \\ & \{ \ -1, \ +1, \ \iota(s_0) \ \}, \\ & \{ \ -1, \ -1, \ \iota(s_0) \ \} \ \} \end{aligned}$$

◇

Now we define a feasible interval for a construction plan and a driving parameter.

Definition 3 *Let $S = \langle G, P, L, I\rangle$ be a construction plan, $p \in P$ the driving parameter, $\alpha : P \to \mathbb{R}$ a given parameters assignment and α' the associated partial parameters assignment. We say that an interval $[a, b] \in \mathbb{R}$, with $a < b$, belongs to the domain of S if for all $x \in [a, b]$ there is a $\iota' \in \mathbb{I}$ and an anchor k for G such that $\Phi(k.\alpha'.\iota'.S)$ holds. We will denote such an interval by $R_a(\alpha, \iota', p)$.*

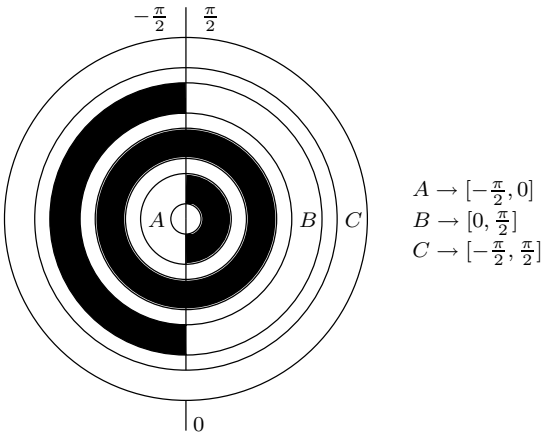


Figure 11: Example of domain for an angle as driving parameter.

Finally, we define the domain for which the problem is feasible given a driving parameter, a parameters assignment and, a set of indices to be explored.

Definition 4 *Let $S = \langle G, P, L, I \rangle$ be a construction plan and $p \in P$ the driving parameter. Let $\alpha : P \to \mathbb{R}$ a parameters assignment and $\mathbb{I}$ a set of explored indices. The domain of feasible values for the driving parameter p is*

$$\mathbb{D}(\alpha, p) = \bigcup_{\forall \iota' \in \mathbb{I}} \bigcup_{\forall a} R_a(\alpha, \iota', p)$$

We consider three types of driving parameters: point-line distance, point-point distance and angle between two segments. In our context, we do not consider negative distances, therefore the set of values that apply for point-line distances and for point-point distance are subsets of $[0, \infty)$. The driving parameter for the domain in Figure 10 is a point-point distance.

When the driving parameter is an angle, the situation is somehow more complicated. If geometric elements are oriented, angles take value within the interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$ however, since angles are circular, they can be given as values in $\mathbb{R}$. Figure 11 shows the domain for an angular parameter. Index assignments are represented along a radial axis. Feasible angle values are represented as circular intervals.

4.3 Router

The router, fed with the domain of feasible values for the construction plan, computes a directed graph, called the *routing graph*, which captures the transitions of driving parameters values between feasible intervals.

In the routing graph, nodes represent feasible intervals of the domain and edges represent valid transitions between intervals. For each feasible interval there are two nodes. One node corresponds to driving parameter values increasing from the lower to the upper bound. The other node corresponds to driving parameter values decreasing from the uper to the lower bounds. Figure 12 illustrates a routing graph with four intervals, A, B, C and D. Nodes

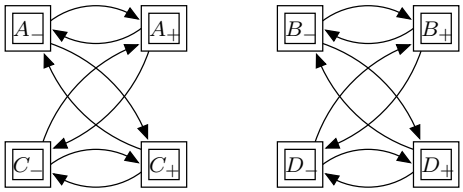


Figure 12: Routing graph for the piston example.

for increasing driving parameter values are denoted X_+ while X_- denotes decreasing values. Notice that in the example there are two disconnected components.

To compute the routing graph the router explores the input domain seeking for specific interval configurations which represent valid transitions for the driving parameter. We define valid transitions on a continuity bases. Concerning the properties of an ideal dynamic geometric system listed in Section 3, we consider two continuity factors: continuity in values assigned to the driving parameter and in the values assigned to the indices. Continuity in the driving parameter means that small changes in the value assigned to the driving parameter will result in small changes in the placements of geometric elements. Continuity in the index assignment means that the driving parameter will take values in a given feasible interval unless an explicit jump into another feasible interval is explicitely indicated. Moreover, jumping from a feasible interval to another will be allowed only at the current interval bounds.

In these conditions, we first define valid transitions for driving parameters which are distances. Let $A = [a_l, a_u]$ and $B = [b_l, b_u]$ be two domain intervals and let p be the driving parameter. Refer to Figure 13. Valid transitions are

1. If p is taking increasing values in A_+, when $p = a_u$, valid transitions are either
   (a) The new interval is A_- and p takes decreasing values.
   (b) Whenever $a_u = b_u$, the new interval is B_- and p takes decreasing values.
   (c) Whenever $a_u = b_l$, the new interval is B_+ and p takes increasing values.
2. If p is taking decreasing values in A_-, when $p = a_l$, valid transitions are either
   (a) The new interval is A_+ and p takes increasing values.
   (b) Whenever $a_l = b_l$, the new interval is B_+ and p takes increasing values.
   (c) Whenever $a_l = b_u$, the new interval is B_- and p takes decreasing values.

Valid transitions defined for distance parameters also apply to angle parameters. However, because of their circular nature, angle parameters have an additional set of valid transitions illustrated in Figure 14. In the conditions given for distance driving parameters, valid specific transitions for angles are

1. If p is taking increasing values in A_+, when $p = a_u$, valid transitions are either
   (a) The new interval is A_- and p takes decreasing values.

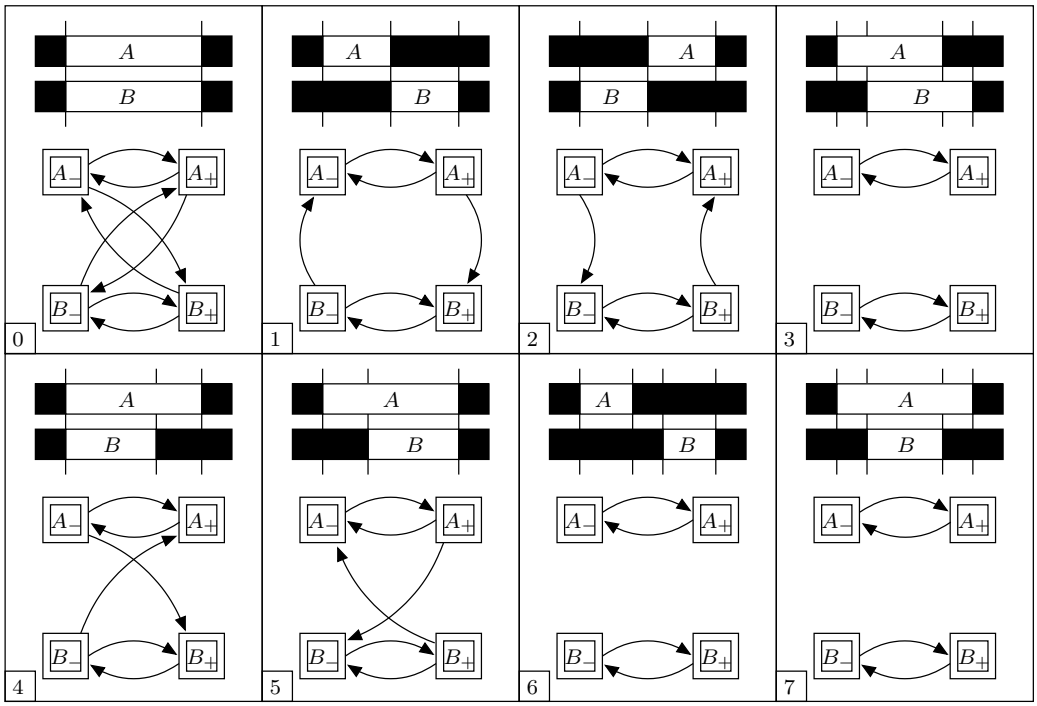


Figure 13: Basic transitions for a distance driving parameter.

(b) Whenever $a_u = \pi/2$, if there is an interval B_+ with $b_l = -\pi/2$, then the new interval is B_+ with p taking increasing values.

2. If p is taking decreasing values in A_-, when $p = a_l$, valid transitions are either

   (a) The new interval is A_+ and p takes increasing values.

   (b) Whenever $a_l = -\pi/2$, if there is an interval B_- with $b_u = \pi/2$, then the new interval is B_- with p taking decreasing values.

When considering an angle driving parameter, specific angle rules will have precedence over transition rules shared with distance driving parameters.

4.4 Filter

As defined, nodes in a routing graph can be source to several arcs which define transitions to another nodes. This is not surprise because, in fact, the routing graph capturers all the possible dynamic behaviors. Since we are only interested in those that, loosely speaking, are correct, we need to provide the user with tools able to define the expected behavior.

Many systems of dynamic geometry focused on a single dynamic behavior ignoring the many different possibilities. It is our aim to build a dynamic geometry system offering the user a more rich variety of behaviors. In this context we define a behavior graph as follows:

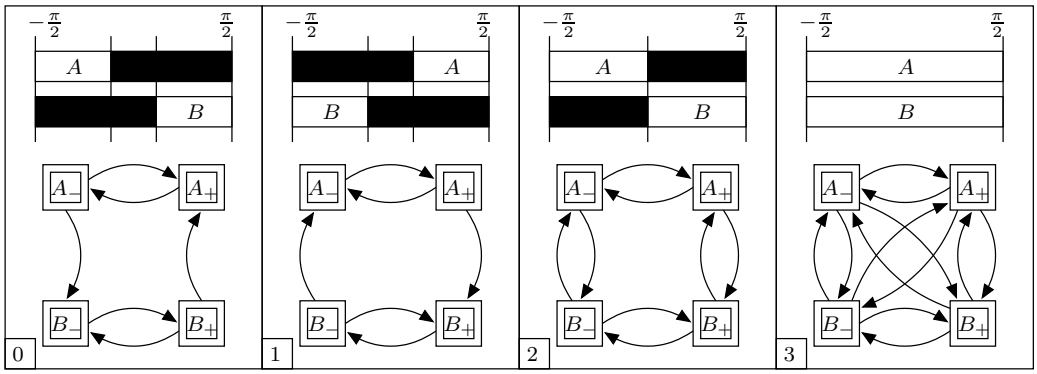


Figure 14: Additional transitions for an angle driving parameter.

Definition 5 *Let $G = (V, E)$ be a routing graph. A* behavior graph *is a subgraph $G'(V, E')$ such that for each node $v \in V$ the number of edges whose source is v is either zero or one.*

In general, several different behavior subgraphs can be derived from a given routing graph. Let G be a routing graph and $\mathbb{B}_G$ be the set of all its behavior subgraphs. Arcs in a behavior graph can be selected following different techniques. Consider the routing graph given in Figure 12.

A very simple technique consists in selecting randomly the arcs. Figure 15 shows a behavior graph whose arcs are generated randomly from the routing graph. Clearly, the resulting behavior is rather useless.

Figure 16 shows a routing graph where the allowed transitions correspond to the classic circular path of a piston. The routing graph in Figure 17 captures the set of transitions corresponding to a path where the piston travels up and down but the crakshaft describes a forward-backward semicircular path. In Section 5 we will show how piston and rocker behaviors have been defined. Devising general automatic or semiautomatic methods to rule out undesired transitions is part of the work in progress.

5 Implementation

solBCN is a free software project whose goal is to implement the architecture for constructive geometric constraints solvers described in Section 2. First, we briefly describe some

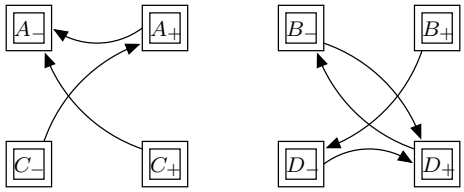


Figure 15: Behavior graph with arcs computed at random.

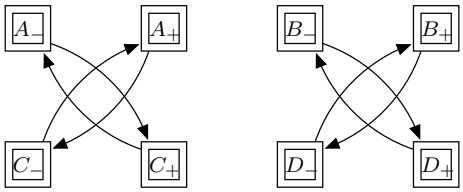


Figure 16: Graph for the piston example with piston behavior.

implementation issues in solBCN and next we devote a section to each component of the new architecture defined in Section 4.

5.1 solBCN

solBCN is a free software project hosted at *La Farga* [20], the free software repository of the *Càtedra de Programari Lliure* of the *Universitat Politècnica de Catalunya*. solBCN is written in Java so that the architecture in Section 2 is mapped to a set of classes. The classes corresponding to the data entities and the functional units in the architecture are described in several documents: the geometric constraints problem and the assignments in [19], the construction plan in [4], the analyzer in [18] and the constructor in [4].

The analyzer implements the algorithm given in [21] for computing a tree-decomposition of a geometric constraints graph. The concept of tree-decomposition has been proved useful to show that several geometric constraints solving techniques have the same domain, i.e. they solve the same set of geometric constraints problems, see [10, 12]. It has also been proved to be of practical interest because a tree-decomposition synthetically describes the sequence of geometric operations L in a construction plan. In addition, it can be evaluated without explicitly computing L as we show below.

Before formally defining the concept of tree-decomposition we must give a definition for geometric constraints graph and for set decomposition. Let $A = \langle G, C, P \rangle$ be a geometric constraints problem, it can be represented by a *geometric constraints graph* $G = (V, E)$ such that the vertices of the graph are the geometric elements in the problem A, $V = G$, and there is an edge $(g_1, g_2) \in E$ if there is a constraint in C involving the geometric elements g_1 and g_2. For example, Figure 18 shows the geometric constraints graph corresponding to the geometric constraints problem in Figure 2.

The concept of *set decomposition* refers to a way of partitioning a given abstract set. We

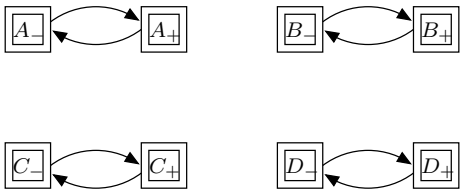


Figure 17: Graph for the piston example with rocker behavior.

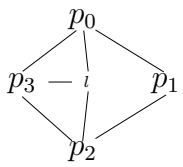


Figure 18: Geometric constraints graph corresponding to the geometric constraints problem in Figure 2.

define the concept of set decomposition of a graph, then we define the recursive application of set decompositions that leads to the concept of *tree-decomposition* of a graph.

Definition 6 *Let V be a set with at least three different members, say $\langle a, b, c\rangle$. Let $V_1, V_2, V_3 \subset V$. We say that $\langle V_1, V_2, V_3\rangle$ is a* set decomposition *of V if*

1. $V_1 \cup V_2 \cup V_3 = V$,
2. $V_1 \cap V_2 = \{a\}$,
3. $V_1 \cap V_3 = \{b\}$ *and*
4. $V_2 \cap V_3 = \{c\}$.

We say that $\langle a, b, c\rangle$ are the *hinges* of the set decomposition.

Definition 7 *Let $G = (V, E)$ be a graph and $V_1, V_2, V_3 \subseteq V$. Then $\langle V_1, V_2, V_3\rangle$ is a* set decomposition *of G if it is a set decomposition of V and for every edge $e \in E$, $V(e) \subseteq V_i$ for some $i, 1 \leq i \leq 3$.*

Roughly speaking, a set decomposition of a graph $G = (V, E)$ is a set decomposition of the set of vertices of V such that does not break any edge in E. Figure 19 shows a set decomposition of the graph in Figure 18.

Definition 8 *Let $G = (V, E)$ be a graph. A 3-ary tree T is a* tree-decomposition *of G if*

1. *V is the root of T,*
2. *Each node $W \subseteq V$ of T is the father of exactly three nodes, say $\langle W_1, W_2, W_3\rangle$, which are a set decomposition of the subgraph of G induced by W, and*
3. *Each leaf node contains exactly two vertices of V.*

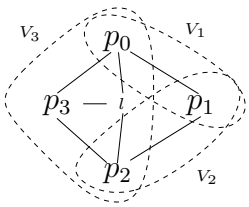


Figure 19: Set decomposition of the geometric constraints graph in Figure 18.

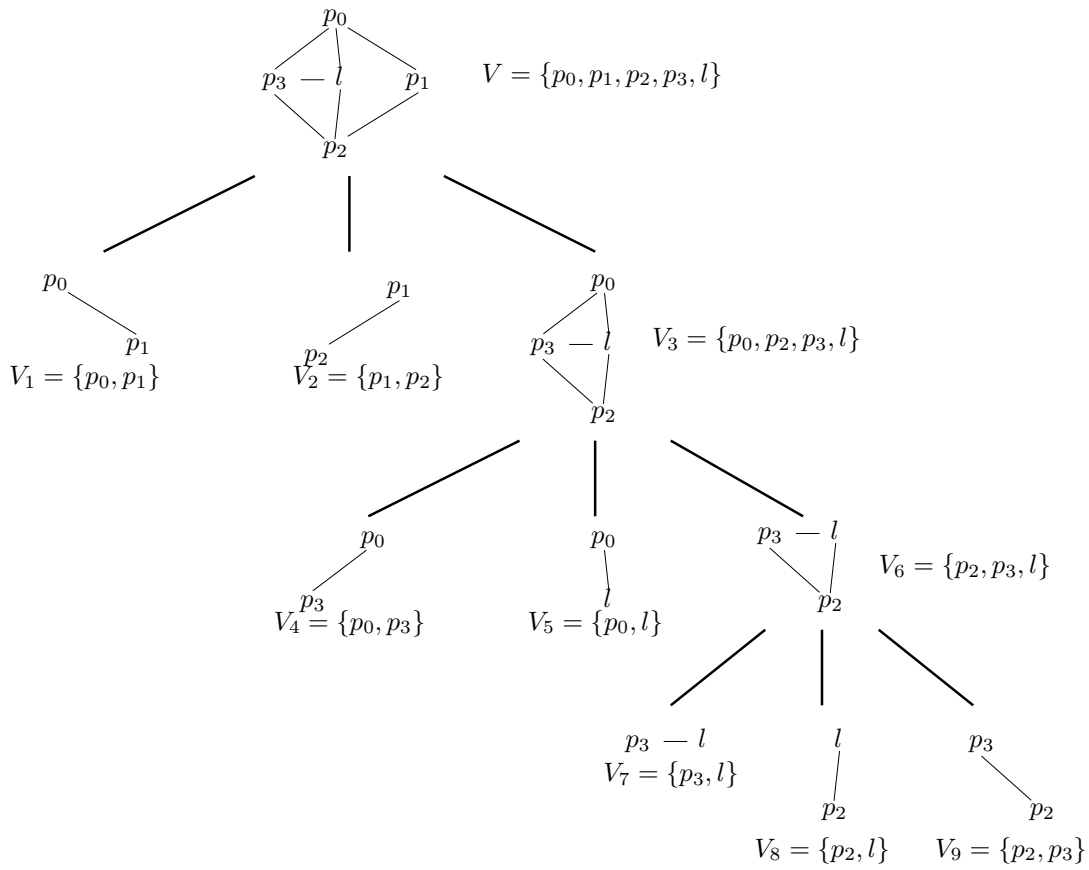


Figure 20: Tree decomposition of the geometric constraint graph in Figure 18.

Figure 20 shows a tree-decomposition of the geometric constraints graph in Figure 18. According to Definition 8, only the vertex set of each subgraph in Figure 20 corresponds to a node in the tree-decomposition. However, Figure 20 also shows the induced subgraph to emphasize that a tree-decomposition induces a decomposition of the graph. A tree-decomposition describes how to decompose a complex problem into successively simpler problems. The leaves correspond to the trivial problem of computing the coordinates of two geometric elements involved in a constraint of the problem. The former is a top down view of a tree-decomposition, but a tree-decomposition can be interpreted bottom up as follows. A tree-decomposition describes how to assemble a geometric constraints graph from its simplest components, the edges. This is just how the evaluation of a tree-decomposition works.

Now, we show how a tree-decomposition, T, can be evaluated. The constructor is the functional unit that computes an anchor κ from a construction plan $S = \langle G, P, L, I \rangle$, a parameters assignment α and an index assignment ι. Since a tree-decomposition describes a sequence of geometric operations, T plays the role of the sequence of geometric operations L in the constructor of solBCN, and thus, the constructor is the functional unit that evaluates the tree-decomposition T.

Let $A = \langle G, C, P \rangle$ be a geometric constraints problem, let $G = (V, E)$ be its geometric constraints graph and let T be a tree-decomposition of G. Let W be a node of T. We associate an anchor κ_W with domain W to each node of T. The anchors associated to the

nodes in T fulfill the following invariant property: let a and b be two distinct geometric elements in G, the measurement of the distance or angle between a and b is the same for every anchor κ_W such that $a, b \in W$. For example, the distance between p_0 and p_2 measured in the node V_3 of the tree in Figure 20 is equal to the distance measured in node V. In other words $dist(\kappa_{V_3}(p_0), \kappa_{V_3}(p_2)) = dist(\kappa_V(p_0), \kappa_V(p_2))$.

Let us show how to compute each κ_W. There are two cases.

1. W is a leaf of T. A leaf of T corresponds to an edge in G which corresponds to a geometric constraint in C. Thus, we have to assign coordinates to the two geometric elements involved in the constraint. This can be easily accomplished, see [5]. For example, assume that the constraint is a distance between points, $d_1 = dist(p_1, p_2)$, then the anchor $\kappa_W(p_1) = (0, 0)$ and $\kappa_W(p_2) = (d, 0)$ satisfy the constraint.

2. W is not a leaf of T. Then, W has three sons $\langle W_1, W_2, W_3 \rangle$, which are a set decomposition of the subgraph of G induced by W. Let $\langle a, b, c \rangle$ be the hinges of this set decomposition such that $W_1 \cap W_2 = \{a\}$, $W_1 \cap W_3 = \{b\}$ and $W_2 \cap W_3 = \{c\}$. This implies that $\{a, b\} \subseteq W_1$, $\{a, c\} \subseteq W_2$ and $\{b, c\} \subseteq W_3$. Assume that we already have computed the anchors associated with W_1, W_2 and W_3, say κ_{W_1}, κ_{W_2} and κ_{W_3} respectively. Now, κ_W is computed in two steps: first we compute $\kappa_{\{a,b,c\}}$ for the hinges and second we compute $\kappa_{W_1-\{a,b\}}$, $\kappa_{W_2-\{a,c\}}$ and $\kappa_{W_3-\{b,c\}}$ for the elements in W_1, W_2 and W_3 other than a, b and c. κ_W is the union of $\kappa_{\{a,b,c\}}$, $\kappa_{W_1-\{a,b\}}$, $\kappa_{W_2-\{a,c\}}$ and $\kappa_{W_3-\{b,c\}}$.

   Computing $\kappa_{\{a,b,c\}}$ is equivalent to the problem of computing the coordinates of three geometric elements $\{a, b, c\}$ such that there is a constraint between every pair of elements, i.e. a constraint between $\{a, b\}$, $\{a, c\}$ and $\{b, c\}$. Depending on whether a, b and c are points or lines there are three cases, see [5]. For example, if a, b and c are points, and let d_1, d_2 and d_3 be the distance between $\{a, b\}$, $\{a, c\}$ and $\{b, c\}$ respectively, then $\kappa_{\{a,b,c\}}(a) = (0, 0)$, $\kappa_{\{a,b,c\}}(b) = (d_1, 0)$ and $\kappa_{\{a,b,c\}}(c)$ is the intersection of a circle centered in a with radius d_2 and a circle centered in b with radius d_3. Note that the intersection between two circles may have up to two different solutions and, thus, there is a sign $s \in I$ involved in this calculation. The values of the constraints between $\{a, b\}$, $\{a, c\}$ and $\{b, c\}$ can be computed using the anchors κ_{W_1}, κ_{W_2} and κ_{W_3} respectively. For example, $d_1 = dist(\kappa_{W_1}(a), \kappa_{W_1}(b))$.

   Note that for a set of symbols $\mathcal{S}$, an anchor $\kappa_{\mathcal{S}}$ establishes a coordinate system where the elements of $\mathcal{S}$ are placed. The geometric elements in W_1 have coordinates assigned in the coordinate system of κ_{W_1} but may have different coordinates in the coordinate system of κ_W. Note that the coordinate system of κ_W is the same as the coordinate system of $\kappa_{\{a,b,c\}}$ by definition. Since the elements in $W_1 - \{a, b\}$ are also in W, we can transform their coordinates to the coordinate system of W. This can be accomplished by computing a linear transformation $M_{\{a,b\}}$ that transforms a and b in the coordinate system of κ_{W_1} to the coordinate system of κ_W and then applying $M_{\{a,b\}}$ to $\kappa_{W_1}(x)$ for all x in $W_1 - \{a, b\}$ leading to $\kappa_{W_1-\{a,b\}}(x)$. The anchors $\kappa_{W_2-\{a,c\}}$ and $\kappa_{W_3-\{b,c\}}$ are computed analogously.

The way a tree-decomposition is evaluated is relevant to the trimmer functional unit. The trimmer computes the induced index from the construction plan and the driving parameter. Since in solBCN the construction plan contains a tree-decomposition instead of a sequence of geometric operations, the induced index must be computed from the tree-decomposition.

5.2 Trimmer

The trimmer functional unit computes the induced index from a construction plan and a driving parameter, see Section 4. We have shown in Section 4.1 that the induced index is computed from a sequence of geometric operations through a dependency graph. In this section we show how to compute the induced index from a tree-decomposition.

Let T be a tree-decomposition of the geometric constraints problem $A = \langle G, C, P\rangle$ that leads to the construction plan $S = \langle G, P, L, I\rangle$, and let $d \in P$ be the driving parameter. First, we identify the *driving leaf* of T, the leaf that corresponds to the constraint involving the driving parameter d. For example, assume that d_2 is the driving parameter in the geometric constraints problem in Example 1. Then the leaf V_9 in Figure 19 is the driving leaf. The relevant nodes in T for computing the induced index are those in the *driving path* h, the path from the root to the driving leaf. For each node in h, we should figure out whether or not the sign associated with the node is involved in a computation that depends on d.

Accurately computing the dependencies of the driving parameter is challenging because not all of the sings associated with the nodes in the driving path h depend on d. Let W be a non-terminal node in T with hinges $\langle a, b, c\rangle$. To compute $\kappa_{\langle a,b,c\rangle}$ we must measure the distance or angle between $\{a, b\}$, $\{a, c\}$ and $\{b, c\}$. By the invariant property, this measure can be taken in any node of T that contains the two elements to measure. Thus, we can avoid dependencies on d taking measurements in nodes of T that do not depend on d when possible. Computing $\kappa_{\langle a,b,c\rangle}$ depends on d in the following cases:

1. When the driving leaf is either $\{a, b\}$, $\{a, c\}$ or $\{b, c\}$, or
2. When there are no three nodes in T, say W_1, W_2 and W_3, such that $\{a, b\} \in W_1$, $\{a, c\} \in W_2$ and $\{b, c\} \in W_3$ and none of these pairs depends on the driving parameter d in their respective nodes.

Now we give conditions to know whether the coordinates of an element e , which is not a hinge in W, depends on d or not. Let W' be the son of node W in the tree T which contains the element e. Then, the coordinates of the element e in W depend on d in the following cases:

1. When e already depends on d in W', or
2. When the coordinates of e are transformed from the coordinate system of $\kappa_{W'}$ to the coordinate system of κ_W by a transformation matrix that depends on d.

Finally, the signs in the induced index are those involved in the calculations of the hinges that depend on the driving parameter.

Example 12 Assume that d_2 is the driving parameter in the geometric constraints problem in Example 1. Then the leaf V_9 in Figure 19 is the driving leaf. The driving path includes the nodes V_6, V_3 and V in Figure 19. The hinges $\langle l, p_3, p_2\rangle$ in V_6 depend on d_2 because $\{a, b\}$ is the driving leaf. The hinges $\langle p_0, p_3, l\rangle$ in V_3 does not depend on d_2 because neither $\{p_0, p_3\}$, $\{p_0, l\}$ nor $\{p_3, l\}$ are the driving leaf, and there are same nodes that do not depend on d_2 which contain some of the pairs above; for example V_4,

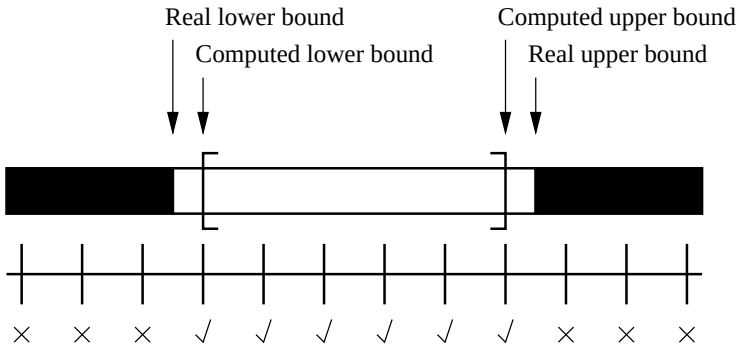


Figure 21: Computed interval and the real interval bounds.

V_5 and V_7. The hinges $\langle p_1, p_0, p_2 \rangle$ in V depend on d_2 because the distance between the pair $\{p_0, p_2\}$ can only be measured in node V_3 and the coordinates of p_2 are converted from those in V_6 by a transformation matrix that depends on d_2. The transformation matrix is computed from p_3 and l in V_6 which depend on d_2.

The induced index is thus $\{s_1, s_2\}$ where s_1 is the sign involved in the calculations of the hinges $\langle l, p_3, p_2 \rangle$ in V_6 and s_2 is the sign involved in the calculations of the hinges $\langle p_1, p_0, p_2 \rangle$ in V. $\diamond$

5.3 Sampler

The main idea used to implement the sampler is to sample the parameter space. The sampler makes samples for the driving parameter values and checks whether the geometric problem is feasible. If the answr is yes, the sample value is stored. Finally, sets of consecutive feasible sampled values are merged into domain intervals. The upper bound for the interval is the first sampled value and the lower bound of the interval is the largest sampled value.

The final result is an interval with approximate bounds that represents the true interval. As shown by Mata [17], computing the true bounds of a feasible interval for a geometric construction is an NP problem. However, Mata reports on a technique that allows to compute the bounds of the parameter domain with a specific precision.

Figure 21 gives examples of the intervals computed by the sampler functional unit. We know that the real bound is between two values. One is a computable sample and a non computable sample. To obtain more accurate interval bounds we would perform an oversampling on the driving parameter values.

5.4 Filter

We have implemented two different *adhoc* filters for the piston problem which capture the behavior of a piston and rocker respectively. Both filters use the point of interest in order to measure certain property just in the interval bounds. The filtering process consists of two steps: assigning a weight to each edge and pruning edges.

We first see the pruning edge step because it is common to both filters. Let us suppose that,

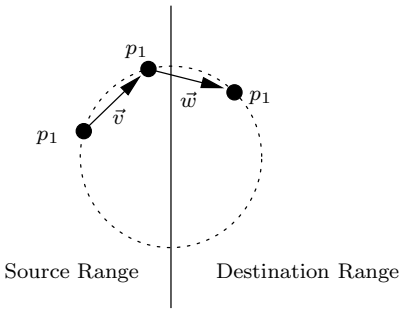


Figure 22: Example of the velocity vectors $\vec{v}$ and $\vec{w}$ in the piston problem.

using some criterion, the edge weight step has assigned a weight for each edge of the routing graph. Then, for each node, the edge pruning step deletes all the edge incident on it except that with minimum weight. Clearly, the resulting graph fulfills the definition of behavior graph.

Now let us see the step of edge weighting. The idea that has inspired our implementation is the following. We assume that the point of interest is the foot of the connecting rod, p_1 in Figure 2. The point of interest traces a circular path with no return points. In this behavior, the kinematics corresponds to continuous functions with continuous derivatives.

With the same point of interest, if we want a rocker dynamic behavior we expect that the point of interest will trace a path along a circular arc and, when reaching the endpoint of the path will return back to the initial location. Clearly, while the position is a continuous function, the velocity of the point of interest is discontinuous at the path bounds.

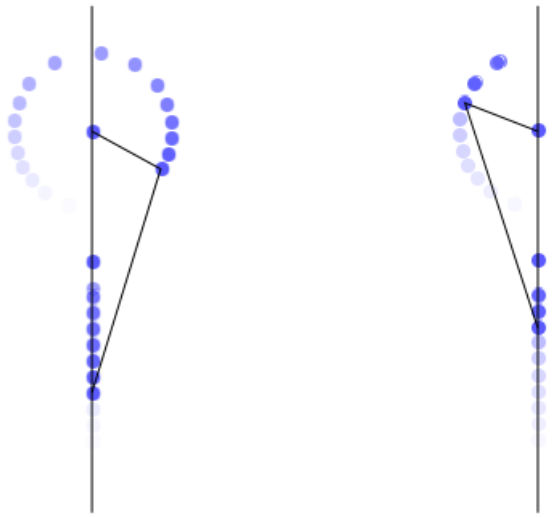
As defined, when an edge of the routing graph represents a transition between different intervals implies a change in the index assignment. This situation is figured out by computing velocities, one when the point of interest would leave the current interval and another when it would enter into the destination interval. Once these velocities are known the rule for weighting edges are defined as follows.

Definition 9 *Let $\vec{v}$ be the velocity vector of the point of interest when leaving an interval. Let $\vec{w}$ be the velocity vector of the point of interest when entering the destination interval. Let α be the angle between $\vec{v}$ and $\vec{w}$. Then the edge weight is given by*

- *Piston behavior: weight* $= |\vec{w}| * \alpha$
- *Rocker behavior: weight* $= |\vec{w}| * (\pi - \alpha)$

The velocity vectors are computed using keyframes sampled for parameter values in the interval of the domain under study. Each keyframe provides the information needed to compute the position of the point of interest while two keyframe allow to compute a velocity vector. Figure 22 gives an example of velocity vector for the piston problem where point p_1 is the point of interest.

The implementation so far reported can inspire other type of filters where edge weighting functions can be defined according to the specific goals to be reached.



(a) Piston keyframes (b) Rocker keyframes

Figure 23: Sequence of keyframes of the piston and rocker filters.

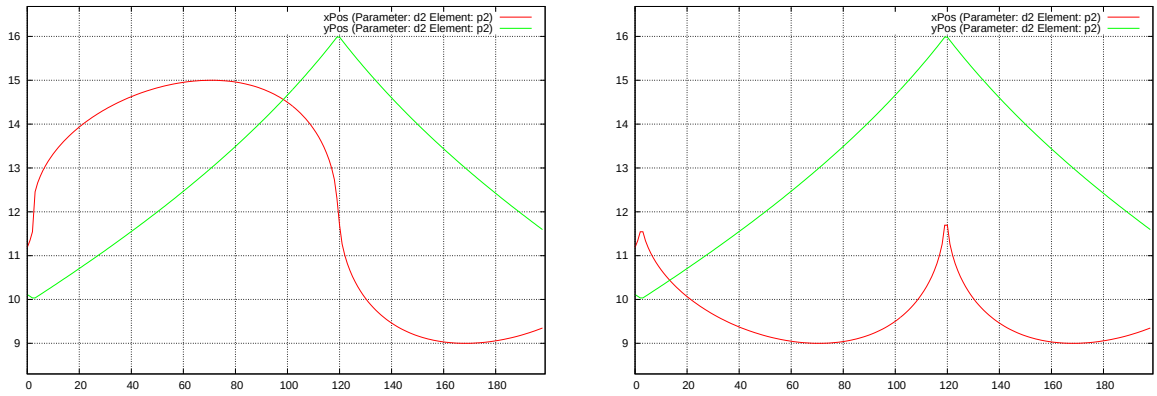
6 Results

We have implemented the architecture defined in this work. The trimmer, sampler and the router have been successfully developed. At the time of writing, the filter only includes the piston and rocker behaviors.

The prototype implemented has proven to be both effective and efficient. The results for the piston simulation are shown as a sequence of keyframes of the set geometric elements in the piston and in the rocker, See Figure 23, and as a point of interest location versus time, see Figure 24 In both cases the filters implemented guarantee that the geometric elements follow the expected paths.

These results have been generated by running the prototype over a cycle in the behavior graph. The cycle of starts with an initial configuration and evolves until reaching a configuration coincident with the starting one. Although in general the existence of a cycle of simulation is not guaranteed, no problem exists for the piston and for the rocker.

The location versus time graph allows to check whether the path followed by the point of interest is the expected. If the interest is to monitor a set of geometric elements, a graph for each geometric element must be plotted. This graph also provides a tool to analyze other interesting issues. For example, the speed of a geometric element or possible discontinuities in the movement.



(a) Piston filter

(b) Rocker filter

Figure 24: Location versus time plots.

7 Conclusions

This work deals with two problems in dynamic geometry: selecting the dynamic behavior and extending the architecture of constructive solvers so that they can be used as an efficient and effective functional unit in dynamic geometry.

We have defined an abstract architecture that extends current architectures of constructive solvers.

The hardest problem is to devise a general solution to the problem of defining general dynamic behaviors. In our work, we have encapsulated it in the filter. At the time of writing this manuscript, we only have developed a partial solution, but we have been able to implement a preliminary prototype with promising results.

8 Future Work

The problems we plan to tackle in the near future are:

- To explore methods to define and capture general behaviors according to specific dynamic requirements.
- Developing a fully functional dynamic geometry environment.
- Developing applications for a number of areas like, form example, teaching geometry, Computer-Aided Design industry, and molecular design.

9 Acknowledgments

This research has been partially funded by Ministerio de Educación y Ciencia and by FEDER under grant TIN2004-06326-C03-01.

References

[1] R. Allen and L. Trilling. Dynamic geometry and declarative geometric programming. In *Geometry Turned On*. Mathematical Association of America, 1997.

[2] Y. Baulac, F. Bellemain, and J.-M. Laborde. *Cabri - the Interactive Geometry Notebook*. Brooks/Cole Publishing Company, 1992.

[3] S. Channac. Techniques d'intelligence artificielle pour l'exécution de programmes logiques géométriques. In *Journées Infografie Intéractive et Intelligence Artificielle*, Limoges, 1996.

[4] M. Freixas. solBCN Constructor. Written in Catalan. `https://lafarga.cpl.upc.edu/docman/view.php/19/82/constructor.pdf`, 2004.

[5] I. Fudos and C.M. Hoffmann. Correctness proof of a geometric constraint solver. *International Journal of Computational Geometry and Applications*, 6(4):405–420, 1996.

[6] M.J. González-López. Using dynamic geometry software to simulate physical motion. *International Journal of Computers for Mathematical Learning*, 6(2):127–142, May 2001.

[7] A. Heydon and G. Nelson. The juno-2 constraint-based drawing editor. Src research report 131a, Digital Equipment, 1994.

[8] C.M. Hoffmann and R. Joan-Arinyo. A Brief on Constraint Solving. *Computer-Aided Design and Applications*, 2(5):655–663, 2005.

[9] N. Jackiw. *The Geometer's Sketchpad*. Key Curriculum Press, Berkeley, 1991–1995.

[10] R. Joan-Arinyo, A. Soto-Riera, S. Vila-Marta, and J. Vilaplana. On the domain of constructive geometric constraint solving techniques. In R. Ďurikovič and S. Czanner, editors, *Spring Conference on Computer Graphics*, pages 49–54. IEEE Computer Society, 2001.

[11] R. Joan-Arinyo, A. Soto-Riera, S. Vila-Marta, and J. Vilaplana. Declarative characterization of a general architecture for constructive geometric constraint solvers. In D. Plemenos, editor, *The Fifth International Conference on Computer Graphics and Artificial Intelligence*, pages 63–76, Limoges, France, 14-15 May 2002. Université de Limoges.

[12] R. Joan-Arinyo, A. Soto-Riera, S. Vila-Marta, and J. Vilaplana-Pastó. Revisiting decomposition analysis of geometric constraint graphs. In *SMA'02: Proceedings of the seventh ACM symposium on Solid modeling and applications*, pages 105–115, New York, NY, USA, 2002. ACM Press.

[13] U. Kortenkamp. *Foundations of Dynamic Geometry.* PhD thesis, Swiss Federal Institute of Technology Zurich, 1999.

[14] J.-M. Laborde and F. Bellemain. *Cabri-Geometry II.* Texas Instruments and Université Joseph Fourier, 1993–1998.

[15] L. Lovász and M.D. Plummer. *Matching Theory.* Number 29 in Annals of Discrete Mathematics. North-Holland, 1986.

[16] M.V. Luzón. *The Root Identification Problem in Constructive Geometric Constraint Solving.* PhD thesis, Universidade da Vigo, 2001. Writen in spanish.

[17] N. Mata. *Constructible Geometric Problems with Interval Parameters.* PhD thesis, Universitat Politècnica de Catalunya, LSI Department, Barcelona, Spain, 2000.

[18] D. Silva. solBCN Analyzer. Written in Catalan. `https://lafarga.cpl.upc.edu/docman/view.php/19/81/analyzer.pdf`, 2004.

[19] D. Silva. Problemes basats en restriccions geomètriques. Written in Catalan. `https://lafarga.cpl.upc.edu/docman/view.php/19/85/problem.pdf`, 2004.

[20] SolBCN. La Farga. `https://lafarga.cpl.upc.edu/projects/solbcn`.

[21] M. Tarrés-Puertas. Direct tree decomposition of geometric constraint graphs. Personal communication, 2007.

[22] S. Vila. *Contribution to Geometric Constraint Solving in Cooperative Engineering.* PhD thesis, Departament de Llenguatges i Sistemes Informàtics. Universitat Politècnica de Catalunya, 2003.

[23] H. Winroth. *Dynamic Projective Geometry.* PhD thesis, Stockholms Universitet, 1999.