Interval Methods for Constructive Geometric Constraint Solving Problems

R. Joan-Arinyo, N. Mata and A. Soto
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Av. Diagonal 647, 8^a, 08028 Barcelona
e-mail: [robert, nmata, tonis]@lsi.upc.es

Abstract

In advanced computer-aided design systems, an object is defined by a collection of geometric elements along with a set of geometric constraints between them. Given the values assigned to the constraint parameters, solving the geometric problem consists of computing the coordinates of the points of the geometric elements defining the object.

Traditionally, constraint parameters have been considered to be exact values. In many applications, however, it is more realistic to consider that some dimensions of the object may take values in given intervals. As a result, the coordinates of the points are also intervals. In this situation, known geometric constraint solving techniques cannot solve directly the geometric problem.

In this paper, we report on an interval-based method to solve geometric problems with interval parameters. We assume that we are given a construction plan for a geometric problem, and an assignment of interval values to the constraint parameters. We present an algorithm that computes precise and rigorous intervals for the coordinates of the points. We illustrate the performance of the algorithm with a case study.

Keywords: CAD, CAM, geometric constraint solving, interval computations.

1 Introduction

Geometric constraint-based design is one of the main features of advanced computer-aided design (CAD) systems. Here, the user inputs the sketch of an object and, at some point, annotates the sketch with a set of constraints. It is left to the CAD system to determine whether the set of constraints precisely defines the object under design and, if so, to place the geometric elements defining the object with respect to each other. This problem is known as the geometric constraint solving problem, and the subsystem in charge of solving it is usually called *the solver*.

Traditionally, geometric constraint solving systems have considered constraint parameters to be exact values. In this situation, the geometric elements of the object are given by points with exact coordinates. In a number of applications, however, the assumption of exact values either does not hold, or it is unacceptable. For instance, in objects whose dimensions have tolerances or, at early stages of the design process, in objects with uncertain parameters. In this situation, it is more realistic to consider constraint parameters to be intervals. Now, the coordinates of the points of the object are also intervals and, in this context, known geometric constraint solving techniques are unable to solve the problem.

Several techniques have been reported in the literature to solve the geometric constraint solving problem. See, for example, references [4, 21] and references therein. Depending on the way geometric constraints are represented, constraint solvers are based either on equational or on constructive techniques. In equational solvers, each geometric constraint is translated into an equation. Solving the geometric problem consists then in solving the corresponding system of equations. The main problem with this approach is that we get very large systems of non-linear equations, with multiple solutions, which are, in general, difficult to handle. In constructive solvers [7, 9, 10, 30, 35, 37], however, geometric constraints are expressed by predicates, and they are interpreted using the axioms of Euclidean geometry. For a reasonably large class of geometric problems, these solvers generate a construction plan that, properly executed, places one-by-one all the geometric elements of the object. We call the realization of the construction plan the *geometric construction problem*.

We assume that we are given a geometric problem defined by a set of points, a set of constraint parameters, and a construction plan in the form of a sequence of basic geometric functions. For an assignment of interval values to the constraint parameters, our goal is to compute the range of the coordinates of the points that place the geometric elements of the object. Unfortunately, simply using interval extensions of the sequence of functions frequently leads to too wide intervals which are of little practical use.

In this paper, we present an interval-based method to solve the geometric construction problem with interval parameters. For the given construction plan, we compute tight bounds for the minimum and the maximum values of each function. Using these bounds, we are able to provide a tight enclosure for the exact range of the locations of points in a way that the geometric constraints defined on them hold.

We have implemented the method using the PASCAL-XSC programming language for verified computing, [24]. We have run it for a number of geometric constraint problems and for different assignments of values to constraint parameters. In all cases, the results were, at least, as good as expected. In Section 8, we illustrate the performance of the algorithm with a case study and discuss the results obtained.

2 Computing with Intervals

The basic mathematical tool used to evaluate functions with interval parameters is interval arithmetic. An interval is a closed and bounded subset of the real numbers

$$\boldsymbol{x} := [\underline{x}, \overline{x}] := \{x \in \mathbb{R} \mid \underline{x} \leq x \leq \overline{x}\}$$

where $\underline{x}$ and $\overline{x}$ denote the lower and the upper bounds of the interval $\boldsymbol{x}$, respectively. When $\underline{x} = \overline{x} = x$, the interval is called *degenerated*, or *thin*.

Using the most widespread system of notation in the interval community, boldface letters denote intervals, lower case letters denote scalar quantities, and upper case letters denote vectors and matrices. Brackets "[ ]" delimit intervals while parentheses "( )" delimit vectors and matrices (vectors are defined to be row vectors).

The set of intervals $\boldsymbol{x} = [\underline{x}, \overline{x}]$ with $\underline{x}, \overline{x} \in \mathbb{R}$, $\underline{x} \leq \overline{x}$, is denoted by $\mathbb{IR}$. The set of interval vectors $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, with $\boldsymbol{x}_i \in \mathbb{IR}$ for $1 \leq i \leq n$, is denoted by $\mathbb{IR}^n$. The interval vector $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ is identified with the box $\{(x_1, \ldots, x_n) \in \mathbb{R}^n \mid x_i \in \boldsymbol{x}_i\ ,\ 1 \leq i \leq n\} \subseteq \mathbb{R}^n$.

Since intervals are sets, it is possible to define set relations such as equality ($=$), membership ($\in$), subset ($\subseteq$), intersection ($\cap$), etc., in the usual way it is done in set theory.

The elementary real operations $\circ \in \{+, -, \cdot, /\}$ are extended to interval arguments $\boldsymbol{x}$ and $\boldsymbol{y}$ by defining the result of an elementary interval operation to be the set of real numbers

$$\boldsymbol{x} \circ \boldsymbol{y} := \{x \circ y \mid x \in \boldsymbol{x}, y \in \boldsymbol{y}\}$$

For each elementary operation, the endpoints of the resulting interval can be expressed in terms of the endpoints of $\boldsymbol{x}$ and $\boldsymbol{y}$. For the explicit formulae, see any standard textbook on interval analysis, for instance, [1, 28].

Let $f : \mathcal{D} \subseteq \mathbb{R} \to \mathbb{R}$ be a real-valued function, continuous on every closed interval in its domain $\mathcal{D}$. When the argument of the function f is an interval $\boldsymbol{x} \subseteq \mathcal{D}$, the range of f over $\boldsymbol{x}$ is defined by

$$f(\boldsymbol{x}) := \{f(x) \mid x \in \boldsymbol{x}\}$$

Note that since f is assumed to be continuous on $\boldsymbol{x}$, $f(\boldsymbol{x})$ is an interval.

In general, it is not possible to express the range of a function in terms of the endpoints of the parameter $\boldsymbol{x}$. This is only possible for some elementary functions such as $\sin x$, $\cos x$, x^2, or $\sqrt{x}$. If the function $f(x)$ is given as the composition of several arithmetic operations and elementary functions, in general, interval arithmetic gives only an enclosure of the range $f(\boldsymbol{x})$, [28].

Let $\mathbb{S}$ be the set of machine floating point numbers available in a computer. Let $\mathbb{IS}$ be the set of machine intervals $\boldsymbol{x} = [\underline{x}, \overline{x}] \in \mathbb{IR}$ such that $\underline{x}, \overline{x} \in \mathbb{S}$. The set of machine interval vectors $\mathbb{IS}^n$ is the set of all interval vectors $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, with each component $\boldsymbol{x}_i \in \mathbb{IS}, 1 \leq i \leq n$.

Let $F : \mathcal{D} \subseteq \mathbb{IS} \to \mathbb{IS}$ be an interval function. $F(\boldsymbol{x})$ is an interval extension of $f(x)$ if

$$f(\boldsymbol{x}) \subseteq F(\boldsymbol{x}) \quad \forall x \subseteq \mathcal{D}$$

For instance, if $f(x)$ is given as the composition of several arithmetic operations, then the most natural interval extension of f is obtained by replacing each elementary arithmetic operation by the corresponding operation over intervals. Unfortunately, the equality $f(\boldsymbol{x}) = F(\boldsymbol{x})$ rarely holds, and $F(\boldsymbol{x})$ can only compute bounds on the range of f. In this situation, we say that $F(\boldsymbol{x})$ overestimates $f(\boldsymbol{x})$.

3 Related Work

Interval techniques have been used extensively for devising solutions to many problems involving numerical computations. Our aim here is just to illustrate the diversity of the areas of literature discussing aspects of interval techniques.

A significant amount of work has been reported on verified solution of nonlinear systems and global optimization problems. Kearfott, [23], Hansen, [13], and Ratschek and Rokne [31] offer comprehensive works in the field.

The study of interval techniques in relation with interval constraints was first introduced by Cleary in [8]. The initial goal was to address the incorrectness of floating point numerical computations in Prolog language. Later, interval constraints have been introduced systematically in many constraint programming systems. See Benhamou and van Hentenryck, [3], and the references therein.

Geometric computation is a fertile area where interval arithmetic finds natural applications. Mudur and Koparkar, [29], report on a method to deal with recursive subdivision techniques for performing tasks as curve drawing, silhoutte drawing, surface shading, and surface-surface intersection.

Several efforts to develop robust techniques for computational geometry make use of interval concepts. Ratschek and Rokne, [32] developed two box-sphere intersection tests where the exact range of some special functions are computed using interval arithmetic. Kao and Knott, [22], have implemented a numerically correct algorithm to solve the convex hull problem in two dimensions. Guibas *et al.*, [11] proposed the *epsilon geometry* for imprecise computations. Segal, [34], uses intervals to describe uncertainty and to take account for innacuracy and quantify its effects during geometric computations.

Bowyer, [5], implemented a CSG-based solid modeller which includes an interval arithmetic library to compute the range of a function over an interval. Interval arithmetic is used to categorize any box against any primitive. In [6], Bowyer *et al.* reported on more general applications of interval methods in geometric modelling.

Joan-Arinyo and Mata, [14, 15, 16, 19], have applied interval arithmetic in constructive geometric constraint solving systems where parameters have uncertain values.

4 The Geometric Construction Problem

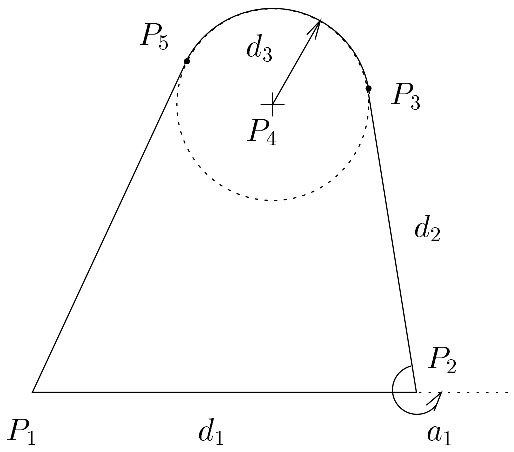
We consider objects built from a collection of points, linear segments, and circular segments with constant radii, and a set of geometric relations between these points and segments, which are the constraints of the problem. The types of geometric constraints we consider are, among others, distance between two points, distance from a point to a line, angle between two rays (infinite semi-lines), point on a segment, and tangencies between segments and circles.

We represent linear segments and circular segments by sets of points. A linear segment is represented by its endpoints, and a circular segment is represented by its endpoints and the centerpoint of the supporting circle. We assume that all the geometric constraints have been reduced to either distance between two points, distance from a point to a line, and angle between two rays, [26]. Figure 1 shows a geometric problem defined by geometric constraints.

4.1 The Problem with Exact Parameters

We assume a constructive solver has generated a construction plan $\mathcal{F}(\mathcal{X},\mathcal{C})$ for a geometric problem with points $\mathcal{X} = \{P_1, \ldots, P_n\}$ and constraint parameters $\mathcal{C} = \{c_1, \ldots, c_m\}$, with each parameter c_i either a distance or an angle.

Furthermore, this construction plan describes only one solution of the geometric problem, preferably the solution intended by the designer, and has the form given in [17]. Namely, $\mathcal{F}(\mathcal{X},\mathcal{C})$ is a structured sequence of basic geometric functions $f_1, \ldots, f_\xi$, where each function f_k computes either a distance, or an angle, or the coordinate of a point. The input data of f_k are constraint parameters in $\mathcal{C}$, and computed values of $f_1, \ldots, f_{k-1}$. The



$$
\begin{array}{l}
distance(P_1, P_2) = d_1 \\
distance(P_2, P_3) = d_2 \\
angle(ray(P_2, P_3), ray(P_2, P_1)) = a_1 \\
tangent(line(P_2, P_3), circle(P_4, d_3)) \\
on(P_3, circle(P_4, d_3)) \\
tangent(line(P_1, P_5), circle(P_4, d_3)) \\
on(P_5, circle(P_4, d_3))
\end{array}
$$

Figure 1: Geometric problem defined by geometric constraints.

structure of $\mathcal{F}(\mathcal{X}, \mathcal{C}) = \{f_1, \ldots, f_\xi\}$ is the following:

$$
\begin{array}{rcl}
e_1 & = & f_1(c_1, \ldots, c_m) \\
e_2 & = & f_2(e_1, c_1, \ldots, c_m) \\
 & \vdots & \\
e_k & = & f_k(e_1, \ldots, e_{k-1}, c_1, \ldots, c_m) \\
 & \vdots & \\
e_\xi & = & f_\xi(e_1, \ldots, e_{\xi-1}, c_1, \ldots, c_m)
\end{array}
$$

When the constraint parameters take exact values, we define the geometric construction problem as follows:

> Given a construction plan $\mathcal{F}(\mathcal{X}, \mathcal{C})$ for an object $\mathcal{X}$, and given an assignment of exact values to the constraint parameters in $\mathcal{C}$, the geometric construction problem consists of computing the coordinates of the points in $\mathcal{X}$ such that all the geometric constraints hold.

Each coordinate of a point in $\mathcal{X}$ is the value of some function f_k in $f_1, \ldots, f_\xi$.

Figure 2 shows the construction plan given by the constructive solver described in [35] for the object of Figure 1. Points Q_i are auxiliary points that do not belong to the object; *asum* and *adif* stand for the addition and subtraction of two angles; and *rc*, *lc*, and *cc*, denote the intersection of a ray and a circle, a line and a circle, and two circles, respectively. For the complete set of the geometric constructions, refer to [18].

4.2 The Problem with Interval Parameters

In a number of applications, however, the assumption that the constraint parameters take exact values either does not hold true, or it is unacceptable. For example, in computer-aided manufacturing, it is more realistic to consider some dimensions of the object to have tolerances; or, at early stages of the design process, it is enough to take ranges of acceptable values for those dimensions which are not essential. In these situations, one can still be able to generate a construction plan for the corresponding geometric problem but, instead of exact values, one has to assign interval values to the constraint parameters.

$$
\begin{aligned}
P_1 &:= point(0,0)\\
P_2 &:= point(d_1,0)\\
\alpha_1 &:= direction(P_1,P_2)\\
\alpha_2 &:= adif(\alpha_1,a_1)\\
P_3 &:= rc(ray(P_2,\alpha_2),\ circle(P_2,d_2),s_{p_3})\\
\alpha_3 &:= direction(P_2,P_3)\\
\alpha_4 &:= asum(\alpha_3,\frac{\pi}{2})\\
Q_1 &:= lc(line(P_2,\alpha_4),\ circle(P_2,d_3),s_{q_1})\\
P_4 &:= lc(line(Q_1,\alpha_3),\ circle(P_3,d_3),s_{p_4})\\
Q_2 &:= midpoint(P_1,P_4)\\
r_1 &:= distance(P_1,Q_2)\\
P_5 &:= cc(circle(P_4,d_3),\ circle(Q_2,r_1),s_{p_5})
\end{aligned}
$$

Figure 2: Construction plan generated by a constructive solver.

Given a set of intervals, $\mathcal{V} = \{\boldsymbol{v}_1,\dots,\boldsymbol{v}_m\}$, we define an assignment of interval values to the constraint parameters in $\mathcal{C}$ as the mapping that assigns to each c_i any value in the interval $\boldsymbol{v}_i$. If c_i is a distance then $\boldsymbol{v}_i \subseteq [0,\infty]$, if c_i is an angle then $\boldsymbol{v}_i \subseteq [0,2\pi]$.

Now, we formally define the geometric construction problem with interval parameters as follows:

> Given a construction plan $\mathcal{F}(\mathcal{X},\mathcal{C})$ for an object $\mathcal{X}$, and given an assignment $\mathcal{V}$ to the constraint parameters in $\mathcal{C}$, the geometric construction problem consists of computing the interval values for the coordinates of the points in $\mathcal{X}$ such that all the geometric constraints hold.

Conceptually, the interval values for the coordinates of the points in $\mathcal{X}$ are the union of the coordinates obtained by solving the geometric problem defined by $\mathcal{F}(\mathcal{X},\mathcal{C})$ for all the possible assignments of exact values $v_1 \in \boldsymbol{v}_1,\dots,v_m \in \boldsymbol{v}_m$ to the constraint parameters in $\mathcal{C}$.

Technically, we want to compute the range of the functions $f_1,\dots,f_\xi$ over the interval constraint parameters $\boldsymbol{v}_1,\dots,\boldsymbol{v}_m$. A naive approach would be to carry out the construction plan in interval arithmetic. However, overestimation in interval extensions yield unacceptable results. Moreover, overestimation can lead to an even worse problem. If the enclosures for the ranges of functions $f_1,\dots,f_{k-1}$ are overestimated, then it can happen that the interval input parameters for the function f_k are not a subset of its domain. In this situation, even when the range of f_k exists, it cannot be computed using only interval arithmetic.

5 Computing Tight Enclosures for the Range of a Function

Overestimation in interval extensions appears when the variables in an expression have multiple occurrences, or when they are dependent on each other. We want to compute the range of the functions $f_1, \ldots, f_\xi$ over the interval constraint parameters $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m$. In the sequence of geometric functions $\mathcal{F}(\mathcal{X}, \mathcal{C}) = \{f_1, \ldots, f_\xi\}$, the parameters of a function f_k are constraint parameters in $\mathcal{C} = \{c_1, \ldots, c_m\}$, and computed values of $f_1, \ldots, f_{k-1}$. Usually, m is much smaller than ξ and, therefore, the functions $f_1, \ldots, f_\xi$ have a strong dependency on $c_1, \ldots, c_m$. In this context, the problem of overestimation is difficult to avoid.

In the next sections, we present our method to compute tight enclosures for the endpoints of the range of a multidimensional function f. It is based on the following approach.

Following Asaithambi *et al.*, [2] and Moore, [28], computing the range of a continuous function f restricted to an interval vector $\boldsymbol{V}$ consists of computing the set

$$f(\boldsymbol{V}) = \{f(V) \mid V \in \boldsymbol{V}\} = [y_{min}, y_{max}]$$

with $y_{min} := \min\{f(V) \mid V \in \boldsymbol{V}\}$ and $y_{max} := \max\{f(V) \mid V \in \boldsymbol{V}\}$. That is, computing $f(\boldsymbol{V})$ amount to computing the minimum and the maximum values of f in the interval $\boldsymbol{V}$. First, one computes an enclosure for the minimum value y_{min} of the function in the given interval $\boldsymbol{V}$

$$\boldsymbol{y}_{min} = [\underline{y}_{min}, \overline{y}_{min}] \quad \text{with} \quad y_{min} \in \boldsymbol{y}_{min}$$

Then, one computes an enclosure for the maximum value y_{max} of the function in the interval $\boldsymbol{V}$

$$\boldsymbol{y}_{max} = [\underline{y}_{max}, \overline{y}_{max}] \quad \text{with} \quad y_{max} \in \boldsymbol{y}_{max}$$

Since $y_{min} \in \boldsymbol{y}_{min}$ and $y_{max} \in \boldsymbol{y}_{max}$, the interval $\boldsymbol{y} = [\underline{y}_{min}, \overline{y}_{max}]$ is an enclosure for the range $f(\boldsymbol{V})$.

The enclosures for the minimum and the maximum values are computed by iteratively reducing the interval vector $\boldsymbol{V}$. Since the amount of overestimation decreases with the reduction of the width of the interval data, the enclosures for the endpoints are obtained with arbitrary accuracy.

5.1 A Gradient-based Approach

First we present the gradient-based approach to compute tight enclosures for the endpoints of the range of a multidimensional function f.

Let $f : \mathcal{D} \subseteq \mathbb{R}^m \to \mathbb{R}$ be a real-valued differentiable function. Let $\boldsymbol{V}_0$ be a machine interval vector contained in $\mathcal{D}$. Then $F : \{\boldsymbol{V} \in \mathbb{IS}^m : \boldsymbol{V} \subseteq \boldsymbol{V}_0\} \to \mathbb{IS}$ is an interval extension of f restricted to $\boldsymbol{V}_0$. That is, F is a function mapping subintervals of $\boldsymbol{V} \subseteq \boldsymbol{V}_0$ to intervals $F(\boldsymbol{V})$ with the property $f(\boldsymbol{V}) \subseteq F(\boldsymbol{V})$.

Let $G_f : \mathcal{D} \to \mathbb{R}^m$ be the gradient of f. Then $G : \{\boldsymbol{V} \in \mathbb{IS}^m : \boldsymbol{V} \subseteq \boldsymbol{V}_0\} \to \mathbb{IS}^m$ is an interval extension of G_f restricted to $\boldsymbol{V}_0$. For all subintervals $\boldsymbol{V} \subseteq \boldsymbol{V}_0$ considered from now on, we simply write $G(\boldsymbol{V}) = (\boldsymbol{g}_1, \ldots, \boldsymbol{g}_m)$. Each component $\boldsymbol{g}_i$ is an interval enclosure for the range $\frac{\partial f}{\partial c_i}(\boldsymbol{V})$ with $1 \leq i \leq m$.

Let $\boldsymbol{g}_i$ be an interval enclosure for the range $\frac{\partial f}{\partial c_i}(\boldsymbol{V})$ with $1 \leq i \leq m$. For any $i \in \{1, \ldots, m\}$, if zero is not contained in the interior of $\boldsymbol{g}_i = [\underline{g}_i, \overline{g}_i]$, that is $0 \notin int(\boldsymbol{g}_i) =]\underline{g}_i, \overline{g}_i[$, the function f restricted to the interval vector $\boldsymbol{V}$ is monotonic with respect to the parameter c_i. Moreover, if $0 \leq \underline{g}_i \leq \overline{g}_i$, then f is monotonically increasing with respect to c_i. We know then that f attains its minimum at a $V \in \boldsymbol{V}$ with $v_i = \underline{v}_i$, and its maximum at a $V \in \boldsymbol{V}$ with $v_i = \overline{v}_i$. Similarly, if $\underline{g}_i \leq \overline{g}_i \leq 0$, the function f is monotonically decreasing with respect to c_i. Therefore, f attains its minimum at a $V \in \boldsymbol{V}$ with $v_i = \overline{v}_i$, and its maximum at a $V \in \boldsymbol{V}$ with $v_i = \underline{v}_i$.

We define two interval vectors $\boldsymbol{U} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m) \subseteq \boldsymbol{V}$, $\boldsymbol{W} = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m) \subseteq \boldsymbol{V}$ whose components are computed in the following way:

$$\begin{cases} \boldsymbol{u}_i := \underline{v}_i \quad \textit{and} \quad \boldsymbol{w}_i := \overline{v}_i & \text{if } 0 \leq \underline{g}_i \leq \overline{g}_i \\ \boldsymbol{u}_i := \boldsymbol{v}_i \quad \textit{and} \quad \boldsymbol{w}_i := \boldsymbol{v}_i & \text{if } \underline{g}_i < 0 < \overline{g}_i \\ \boldsymbol{u}_i := \overline{v}_i \quad \textit{and} \quad \boldsymbol{w}_i := \underline{v}_i & \text{if } \underline{g}_i \leq \overline{g}_i \leq 0 \end{cases}$$

Then, $F(\boldsymbol{U})$ is an enclosure for the minimum value y_{min} of $f(\boldsymbol{V})$, and $F(\boldsymbol{W})$ is an enclosure for the maximum value y_{max} of $f(\boldsymbol{V})$. We take $\boldsymbol{y}_{min} := F(\boldsymbol{U})$ and $\boldsymbol{y}_{max} := F(\boldsymbol{W})$. Using these two enclosures, we get the enclosure

$$\boldsymbol{y} = [\underline{y}_{min}, \overline{y}_{max}]$$

for the range of f restricted to the interval vector $\boldsymbol{V}$. If no component of the gradient contains zero in its interior, $\boldsymbol{y}$ is the most accurate enclosure for $f(\boldsymbol{V})$ we can get from F. In fact, in this case, F can be usually implemented in such a way that we can get an enclosure for $f(\boldsymbol{V})$ in which only machine precision and rounding errors have caused overestimation.

If some component of $G(\boldsymbol{V})$ contains zero in its interior, we use a global optimization approach to get better enclosures for y_{min} and y_{max}.

5.2 A Global Optimization Approach

When some component of the enclosure for the gradient contains zero in its interior, say $0 \in int(\boldsymbol{g}_i)$, the function f may have the minimum or the maximum values of the range at some interior point $V \in \boldsymbol{V}$. In this situation, if the widths of $\boldsymbol{y}_{min}$ and $\boldsymbol{y}_{max}$ exceed a certain tolerance, we apply an algorithm for verified global optimization in order to get more accurate enclosures for the endpoints of the range.

The algorithm we use is a simplified version of the method reported by Ratz in [33]. Since we are only interested in the extreme values y_{min} and y_{max} of the function, we do not need to compute all the minimizers and maximizers of the function f in $\boldsymbol{V}$.

Given the two initial subboxes $\boldsymbol{U}, \boldsymbol{W} \subseteq \boldsymbol{V}$, the algorithm computes the enclosures $\boldsymbol{y}_{min}$ and $\boldsymbol{y}_{max}$ for the minimum and the maximum values of the function f restricted to the interval vector $\boldsymbol{V}$. Note that the minimum value of the function f restricted to $\boldsymbol{V}$ is equal to the minimum value of the function f restricted to the interval vector $\boldsymbol{U}$. That is,

$$y_{min} = \min_{X \in \boldsymbol{V}} f(X) = \min_{X \in \boldsymbol{U}} f(X) \ \subseteq \ \boldsymbol{y}_{min}$$

Similarly, for the maximum value we have

$$y_{max} = \max_{X \in \boldsymbol{V}} f(X) = \max_{X \in \boldsymbol{W}} f(X) \ \subseteq \ \boldsymbol{y}_{max}$$

We briefly describe next the algorithm we use to compute $\boldsymbol{y}_{min}$. $\boldsymbol{y}_{max}$ is computed in a similar way.

Starting with the interval vector $\boldsymbol{U}$, we subdivide $\boldsymbol{U}$ in two subintervals $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$. Subintervals which are guaranteed not to contain y_{min} are discarded, while the remaining subintervals are stored in a list $\mathcal{L}$. The list $\mathcal{L}$ is sorted increasingly with respect to the lower bound of $F(\boldsymbol{X}_i)$, with $i \in \{1, \ldots, \nu\}$ one element in $\mathcal{L}$. The first element is then taken to be the most likely candidate to contain y_{min}, and gets again subdivided. The algorithm stops when the enclosure for y_{min} achieves a desired accuracy $\varepsilon_{\mathcal{X}}$, or when the first box in the list cannot be further subdivided. The speed of the method depends on the ability to discard boxes in which it is not possible to get the minimum value of the function. The test we use to discard pending subboxes combines three different approaches: the midpoint and the monotonicity tests, [12, 33], and a constant-value test we have developed, [27].

If the selected box cannot be discarded after the monotonicity test and the constant-value test, we usually get a reduced subbox to be properly inserted in the sorted list $\mathcal{L}$. The new first box $\boldsymbol{X}_1$ in $\mathcal{L}$ is then selected to be the new candidate to contain y_{min}.

6 Computing Enclosures for the Gradient

A technical issue, when applying the interval algorithm we have just described, is how to compute the enclosure $G(\boldsymbol{V})$ for the gradient of the functions f_k in the sequence of geometric functions $\mathcal{F}(\mathcal{X}, \mathcal{C})$. We describe here the most salient aspects in our implementation.

6.1 Mathematical Concepts

A function f_k, with $k \in \{1, \ldots, \xi\}$, depends on the constraint parameters $c_1, \ldots, c_m$. However, f_k is often defined in terms of intermediate variables computed by the functions $f_1, \ldots, f_{k-1}$, [17]. Let

$$\begin{array}{rcl} e_1 & = & f_1(c_1, \ldots, c_m) \\ & & \vdots \\ e_{k-1} & = & f_{k-1}(e_1, \ldots, e_{k-2}, c_1, \ldots, c_m) \end{array}$$

If $e_k = f_k(e_1, \ldots, e_{k-1}, c_1, \ldots, c_m)$, then the gradient

$$\nabla f_k(c_1, \ldots, c_m) = (g_1, \ldots, g_m)$$

is computed by applying the chain rule. That is, for each $i \in \{1, \ldots, m\}$, the component g_i is obtained as follows

$$g_i = \frac{\partial f_k}{\partial c_i} + \frac{\partial f_k}{\partial e_1} \cdot \frac{\partial e_1}{\partial c_i} + \ldots + \frac{\partial f_k}{\partial e_{k-1}} \cdot \frac{\partial e_{k-1}}{\partial c_i}$$

Note that, when computing the gradient of the function f_k, the gradients $\nabla f_1, \ldots, \nabla f_{k-1}$ have to be known. Besides, we need a suitable representation for the functions $f_1, \ldots, f_{k-1}$, and for the gradients $\nabla f_1, \ldots, \nabla f_{k-1}$, so that we are able to apply, easily, the chain rule.

6.2 Code List of Functions

We decompose $f_1, \ldots, f_\xi$ using a code list of functions, similar to the code lists used by Kearfott in [23] and Krämer *et al.* in [25], where the non-constant functions assigned to the variables $z_1, \ldots, z_N$ are exactly the basic geometric functions $f_1, \ldots, f_\xi$.

The input parameters of a function f_k in a geometric construction problem can be of three types: constraint parameters in $\mathcal{C}$, constant values, and previously computed values of $f_1, \dots, f_{k-1}$.

In the code list of $f_1, \dots, f_\xi$, first we identify the m constraint parameters $c_1, \dots, c_m$ in the variables $z_1, \dots, z_m$; that is, $z_1 = c_1, \dots, z_m = c_m$.

Next, we take the l constant values in the functions $f_1, \dots, f_\xi$ and we assign them to the next variables $z_{m+1}, \dots, z_{m+l}$. The only constant values occuring in the geometric functions we consider, see [20], are the following: zero distance, right angle, and the integer parameters $\{1, 0, -1\}$. Therefore,

$$\begin{aligned} z_{m+1} &= 0 \\ z_{m+2} &= \tfrac{\pi}{2} \\ z_{m+3} &= 1 \\ z_{m+4} &= -1 \end{aligned}$$

Hence, in our application l is equal to 4. Finally, the remaining z_i's are the sequence of geometric functions $f_1, \dots, f_\xi$

$$\begin{aligned} z_{m+4+1} &= f_1(z_1, \dots, z_{m+4}) \\ &\vdots \\ z_{m+4+k} &= f_k(z_1, \dots, z_{m+4+k-1}) \\ &\vdots \\ z_N &= f_\xi(z_1, \dots, z_{N-1}) \end{aligned}$$

with $N = m + 4 + \xi$.

6.3 Computing the Gradient

In order to apply the chain rule efficiently, we generate the Jacobian of $z_1, \dots, z_N$ with respect to $c_1, \dots, c_m$. Then, we get an $m \times N$ matrix of the form

- the submatrix $m \times m$ corresponding to the first m columns is the identity;
- the columns $m+1, \dots, m+4$ are zero; and
- the column $(m+4+k)$ is the gradient $\nabla f_k(c_1, \dots, c_m)$ of the function f_k.

We compute $\nabla f_k(c_1, \dots, c_m) = (g_1^k, \dots, g_m^k)$ using the following approach.

Since $z_{m+4+k}, \dots, z_N$ do not occur in f_k, applying the chain rule we obtain the formula:

$$g_i^k = \frac{\partial f_k}{\partial c_i} = \sum_{j=1}^{m+4+k-1} \frac{\partial f_k}{\partial z_j} \cdot \frac{\partial z_j}{\partial c_i}$$

In this sum, the terms $\frac{\partial f_k}{\partial z_j}$ are obtained using automatic differentiation, while the terms $\frac{\partial z_j}{\partial c_i}$ are previously computed components of the Jacobian.

Using the code list $z_1, \dots, z_N$ and the Jacobian computed in such a way, we obtain enclosures for the gradient $G(\boldsymbol{V})$ in a very simple and natural way. In the next section, we apply the algorithm described in Section 5 to the code list of functions $z_1, \dots, z_N$.

7 The Interval Propagation Algorithm

In the sequence of geometric functions $\mathcal{F}(\mathcal{X}, \mathcal{C}) = \{f_1, \ldots, f_\xi\}$, each function f_k is defined in terms of $c_1, \ldots, c_m$ and $f_1, \ldots, f_{k-1}$. The code list described in Section 6 contains the complete information about the relationship among the function f_k and the input parameters $c_1, \ldots, c_m$.

Now, for an assignment of interval values $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m$ to the constraint parameters $c_1, \ldots, c_m$, we want to compute an enclosure for the range of the geometric function f_k. To do so, for each endpoint of the range of f_k, we iteratively compute smaller subboxes $\boldsymbol{V}' \subseteq \boldsymbol{V} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m)$ until the enclosure for the range of f_k over $\boldsymbol{V}'$ encloses the minimum (maximum) value of f_k at a desired accuracy $\varepsilon_{\mathcal{X}}$. Note that when computing a new enclosure $\boldsymbol{z}_{m+4+k}$ for one of the endpoints of the range over an interval vector $\boldsymbol{V}'$, we also need to evaluate $z_1, \ldots, z_{m+4+k-1}$ over $\boldsymbol{V}'$. Because we need to evaluate these intermediate functions every time we compute an enclosure for f_k over an interval vector $\boldsymbol{V}'$, the algorithm we describe in this section is clearly an interval propagation algorithm.

7.1 Algorithm ConstructPlan

The algorithm ConstructPlan, shown in Figure 3, computes tight enclosures for the range of the coordinates of the points of an object $\mathcal{X} = \{P_1, \ldots, P_n\}$ over the interval vector $\boldsymbol{V} = \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m\} \in \mathbb{IS}^m$, in principle, at a desired accuracy $\varepsilon_{\mathcal{X}}$. This accuracy may not be achieved because the algorithm stops subdividing the interval vector that minimizes, or maximizes, the function f when the gradient has reached an accuracy ε_g. In this situation, it is still possible to compute tight enclosures for the coordinates of the points in $\mathcal{X}$, and the quality of these enclosures can be checked through the inner approximation of the ranges.

The algorithm starts initializing the first $m+4$ columns in $\boldsymbol{Z}$ and in $\boldsymbol{J}$. Here, by an assignment $\boldsymbol{z}_i := r$ of a real number r to an interval $\boldsymbol{z}_i$, we mean that $\boldsymbol{z}_i = [\underline{z}_i, \overline{z}_i]$ is the best possible machine number enclosure for r, that is, $\underline{z}_i$ is the largest machine number less than or equal to r, and $\overline{z}_i$ is the smallest machine number greater than or equal to r. Then, the interval $\boldsymbol{z}_i$ represents r in the computer. Analogously, by an assignment $\boldsymbol{z}_i := \boldsymbol{r}$ of a real interval $\boldsymbol{r} = [\underline{r}, \overline{r}]$, we mean that $\boldsymbol{z}_i = [\underline{z}_i, \overline{z}_i]$ is the best possible machine number enclosure for $\boldsymbol{r}$.

After that, we compute the enclosures for the range of the geometric functions $f_1, \ldots, f_\xi$. While the parameters of the function f_k are in its domain $\mathcal{D}_k$, we compute an enclosure for the range of the function f_k using the procedure RangeGOp. In the algorithm, $\mathcal{F}_k$ denotes the subsequence of geometric functions $f_1, \ldots, f_k$.

7.2 Procedure RangeGOp

The procedure RangeGOp in Figure 4 computes a tight enclosure for the range of the function f_ξ restricted to the interval vector $\boldsymbol{Z} = (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_{n-1})$ with $n = m + 4 + \xi$.

First, the subboxes $\boldsymbol{U}$ and $\boldsymbol{W}$ are initialized to the values $\boldsymbol{U} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{m+4}, 0, \ldots, 0) \subseteq \boldsymbol{Z}$, and $\boldsymbol{W} = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{m+4}, 0, \ldots, 0) \subseteq \boldsymbol{Z}$, according to the information contained in the components of the gradient $\boldsymbol{G} = (\boldsymbol{g}_1, \ldots, \boldsymbol{g}_m)$. The initialization process also creates a list $\mathcal{I}$ with the indices of the components of $\boldsymbol{G}$ that contain zero in their interior. If

algorithm ConstructPlan

INPUT

$\mathcal{F}(\mathcal{X}, \mathcal{C})$: Sequence of geometric function $f_1, \ldots, f_\xi$.

$\boldsymbol{V}$: Vector of interval constraint parameters.

$\varepsilon_{\mathcal{X}}$: Required accuracy for the coordinates.

ε_g : Stopping criterion for the gradient in the constant-value test.

OUTPUT

$\mathcal{X}$: Enclosures for the range of the coordinates.

ok : Termination flag.

for $k := 1$ **to** m **do** $\boldsymbol{z}_k := \boldsymbol{v}_k$ **endfor** { input parameters }
$\boldsymbol{z}_{m+1} := 0$; $\boldsymbol{z}_{m+2} := \pi/2$; { constants }
$\boldsymbol{z}_{m+3} := 1$; $\boldsymbol{z}_{m+4} := -1$;

InitializeJacobian($\boldsymbol{J}$);

$k := 1$;
$ok := \boldsymbol{Z} \subseteq \mathcal{D}_k$;
while $k \leq \xi$ **and** ok **do**
  RangeGOp($\mathcal{F}_k$, $\boldsymbol{Z}$, $m + 4 + k$, $\varepsilon_{\mathcal{X}}$, ε_g, $\boldsymbol{Z}_{inn}$, $\boldsymbol{J}$);
  $k := k + 1$;
  if $k \leq \xi$ **then** $ok := \boldsymbol{Z} \subseteq \mathcal{D}_k$ **endif**
endwhile

if ok **then**
  extract coordinates of $\mathcal{X}$ from $\boldsymbol{Z}$ and $\boldsymbol{Z}_{inn}$;
endif

endalgorithm

Figure 3: Algorithm ConstructPlan.

procedure RangeGOp

INPUT

$\mathcal{F}_\xi(\mathcal{X},\mathcal{C})$: Sequence of geometric function $f_1, \dots, f_\xi$.

$\boldsymbol{Z}$: Code list of $f_1, \dots, f_\xi$.

n : Component in $\boldsymbol{Z}$ corresponding to f_ξ.

$\varepsilon_\mathcal{X}$: Required accuracy for the coordinates.

ε_g : Stopping criterion for the gradient in the constant-value test.

$\boldsymbol{Z}_{inn}$: Inner enclosure for functions in $\mathcal{F}_\xi$.

$\boldsymbol{J}$: The Jacobian matrix.

OUTPUT

$\boldsymbol{Z}$, $\boldsymbol{Z}_{inn}$, $\boldsymbol{J}$

EvalGradient($\mathcal{F}_\xi$, $\boldsymbol{Z}$, n, $\boldsymbol{J}$);
$\boldsymbol{G} := \boldsymbol{J}^n$; {column n}
InitializeBoxes($\boldsymbol{G}$, $\boldsymbol{Z}$, $\boldsymbol{U}$, $\boldsymbol{W}$, $\mathcal{I}$, n);

if $\mathcal{I} = \emptyset$ **then**
EvalLstFunctions($\mathcal{F}_\xi$, $\boldsymbol{U}$, n, $\boldsymbol{y}_{min}$);
EvalLstFunctions($\mathcal{F}_\xi$, $\boldsymbol{W}$, n, $\boldsymbol{y}_{max}$);
else
Minimize($\mathcal{F}_\xi$, $\boldsymbol{U}$, n, $\mathcal{I}$, $\varepsilon_\mathcal{X}$, ε_g, $\boldsymbol{y}_{min}$);
Maximize($\mathcal{F}_\xi$, $\boldsymbol{W}$, n, $\mathcal{I}$, $\varepsilon_\mathcal{X}$, ε_g, $\boldsymbol{y}_{max}$);
endif

$\boldsymbol{z}_n := [\underline{y}_{min}, \overline{y}_{max}]$;

if $\overline{y}_{min} \leq \underline{y}_{max}$ **then** $\boldsymbol{z}_n^{inn} := [\overline{y}_{min}, \underline{y}_{max}]$
else $\boldsymbol{z}_n^{inn} := empty$;
endif

endprocedure

Figure 4: Procedure RangeGOp.

$\mathcal{I}$ is empty, $\boldsymbol{U}$ and $\boldsymbol{W}$ are the minimizer and the maximizer of f in $\boldsymbol{Z}$. This means that the evaluation of the list of geometric functions $f_1, \dots, f_\xi$ for an assignment to the constraint parameters $\boldsymbol{U}$ is an enclosure for the lower endpoint of the range. Analogously, the evaluation of the list of geometric functions over $\boldsymbol{W}$ is an enclosure for the upper endpoint of the range. If $\mathcal{I}$ is not empty, we need to minimize the function f_ξ in $\boldsymbol{U}$, and maximize f_ξ in $\boldsymbol{W}$.

Procedures InitializeBoxes, Minimize, and Maximize are described in detail in reference [18].

8 Results

We have implemented the algorithm ConstructPlan using the PASCAL-XSC programming language for verified computing [24], together with the module for automatic differentiation `grad_ari` from [12]. The algorithm has been run for a number of geometric problems and for different assignments of constraint parameters. To illustrate its performance, we present

$$
\begin{aligned}
z_1 &= d_1 \\
z_2 &= d_2 \\
z_3 &= d_3 \\
z_4 &= a_1 \\
z_5 &= 0 \\
z_6 &= \tfrac{\pi}{2} \\
z_7 &= 1 \\
z_8 &= -1 \\
z_9 &= x\_point(z_5, z_5) \\
z_{10} &= y\_point(z_5, z_5) \\
z_{11} &= x\_point(z_1, z_5) \\
z_{12} &= y\_point(z_1, z_5) \\
z_{13} &= direction(z_9, z_{10}, z_{11}, z_{12}) \\
z_{14} &= adif(z_{13}, z_4)
\end{aligned}
\qquad
\begin{aligned}
z_{15} &= x\_rc(z_{11}, z_{12}, z_{14}, z_{11}, z_{12}, z_2, z_7) \\
z_{16} &= y\_rc(z_{11}, z_{12}, z_{14}, z_{11}, z_{12}, z_2, z_7) \\
z_{17} &= direction(z_{11}, z_{12}, z_{15}, z_{16}) \\
z_{18} &= asum(z_{17}, z_6) \\
z_{19} &= x\_lc(z_{11}, z_{12}, z_{18}, z_{11}, z_{12}, z_3, z_7) \\
z_{20} &= y\_lc(z_{11}, z_{12}, z_{18}, z_{11}, z_{12}, z_3, z_7) \\
z_{21} &= x\_lc(z_{19}, z_{20}, z_{17}, z_{15}, z_{16}, z_3, z_5) \\
z_{22} &= y\_lc(z_{19}, z_{20}, z_{17}, z_{15}, z_{16}, z_3, z_5) \\
z_{23} &= x\_midpoint(z_9, z_{10}, z_{21}, z_{22}) \\
z_{24} &= y\_midpoint(z_9, z_{10}, z_{21}, z_{22}) \\
z_{25} &= distance(z_9, z_{10}, z_{23}, z_{24}) \\
z_{26} &= x\_cc(z_{21}, z_{22}, z_3, z_{23}, z_{24}, z_{25}, z_8) \\
z_{27} &= y\_cc(z_{21}, z_{22}, z_3, z_{23}, z_{24}, z_{25}, z_8)
\end{aligned}
$$

Figure 5: Code list of functions for the geometric problem of Figure 1.

the results obtained for the geometric problem $\mathcal{X} = \{P_1, P_2, P_3, P_4, P_5\}$ with constraint parameters $\mathcal{C} = \{d_1, d_2, d_3, a_1\}$ shown in Figure 1. The construction plan in the form described in [20] is given in Figure 2. It was provided by the constructive geometric constraint solver described in [35].

The construction plan is transformed into a sequence of functions $\mathcal{F}(\mathcal{X}, \mathcal{C})$ using the approach described in [17]. The corresponding code list of functions is given in Figure 5. Note that coordinate functions are usually denoted by the x or y coordinate followed by the corresponding geometric operation in the construction plan.

Now, we compute the enclosures for the coordinates of the points in $\mathcal{X}$ for three different assignments of interval values and constant accuracies $\varepsilon_{\mathcal{X}} = 10^{-2}$ and $\varepsilon_g = 10^{-3}$. The interval values have increasing width.

The results for the first assignment, $d_1 = [17.995, 18.005]$, $d_2 = [19.995, 20.005]$, $d_3 = 5$ and $a_1 = 4.5$ are given in Table 1. The first column shows the name for the coordinates of the points $P_i = (x_i, y_i)$ in $\mathcal{X}$. The second and third columns show the computed enclosures for the range using, respectively, interval arithmetic (denoted by 'ia' in Table 1) and our algorithm ('ip' in the table). The fourth column is the exact range computed via elementary trigonometry in the example.

The results in Table 1 show that our algorithm definitely computes tighter enclosures than interval arithmetic alone. Even more, in most cases, the precision we get is better than the required accuracy $\varepsilon_{\mathcal{X}}$. This is possible because the input parameters have small variations, the variation of the coordinate functions over this parameters is also small, and it is more likely that the monotonicity of the function remains unchanged. Then, it is possible to compute the best possible enclosure for the range of the coordinates using the gradient-based approach of Section 5.1, and global optimization is not required.

Table 2 shows the results obtained for the assignment $d_1 = [17.95, 18.05]$, $d_2 = [19.95, 20.05]$, $d_3 = 5$ and $a_1 = 4.5$. Here, one can see how the overestimation of the enclosures increases when using only interval arithmetic, and how our algorithm provides tighter enclosures for the range. Again, the precision we get is much better than the required accuracy $\varepsilon_{\mathcal{X}}$.

Results in Table 3 correspond to the assignment $d_1 = [17.5, 18.5]$, $d_2 = [19.5, 20.5]$, $d_3 = 5$, and $a_1 = 4.5$. In this case, the interval arithmetic is not able to compute an enclosure

	Computed Enclosure (ia)	Computed Enclosure (ip)	Exact Range
x_1	[0.00000000000, 0.00000000000]	[0.00000000000, 0.00000000000]	[0.00000000000, 0.00000000000]
y_1	[0.00000000000, 0.00000000000]	[0.00000000000, 0.00000000000]	[0.00000000000, 0.00000000000]
x_2	[17.9950000000, 18.0050000000]	[17.9950000000, 18.0050000000]	[17.9950000000, 18.0050000000]
y_2	[0.00000000000, 0.00000000000]	[0.00000000000, 0.00000000000]	[0.00000000000, 0.00000000000]
x_3	[13.7775856836, 13.7905828428]	[13.7780300323, 13.7901379904]	[13.7780300323, 13.7901379904]
y_3	[19.5436517744, 19.5575505964]	[19.5436517744, 19.5575505964]	[19.5457147027, 19.5554900039]
x_4	[8.86345221111, 8.92938217102]	[8.89037944402, 8.90248740211]	[8.89037944404, 8.90248740208]
y_4	[18.4686105381, 18.5245892714]	[18.4917357054, 18.5015110068]	[18.4917357055, 18.5015110068]
x_5	[3.85401583465, 4.14153716259]	[3.99192582856, 4.00473123051]	[3.99192582882, 4.00473123025]
y_5	[19.3375750352, 19.6610873865]	[19.4952993027, 19.5064937979]	[19.4952993031, 19.5064937976]

Table 1: Enclosures for the range of the coordinates of the points in $\mathcal{X}$ for the input parameters: $d_1 = [17.995, 18.005]$, $d_2 = [19.995, 20.005]$, $d_3 = 5$, and $a_1 = 4.5$.

	Computed Enclosure (ia)	Computed Enclosure (ip)	Exact Range
x_1	[0.00000000000, 0.00000000000]	[0.00000000000, 0.00000000000]	[0.00000000000, 0.00000000000]
y_1	[0.00000000000, 0.00000000000]	[0.00000000000, 0.00000000000]	[0.00000000000, 0.00000000000]
x_2	[17.9500000000, 18.0500000000]	[17.9500000000, 18.0500000000]	[17.9500000000, 18.0500000000]
y_2	[0.00000000000, 0.00000000000]	[0.00000000000, 0.00000000000]	[0.00000000000, 0.00000000000]
x_3	[13.7191007344, 13.8491177721]	[13.7235442214, 13.8446238014]	[13.7235442214, 13.8446238014]
y_3	[19.4808858130, 19.6200847835]	[19.5017258474, 19.5994788592]	[19.5017258474, 19.5994788592]
x_4	[8.56492542801, 9.22469215288]	[8.83589363304, 8.95697321308]	[8.83589363307, 8.95697321305]
y_4	[18.2141305612, 18.7744216902]	[18.4477468502, 18.5454998621]	[18.4477468502, 18.5454998621]
x_5	[2.48566163321, 5.40182138168]	[3.93433538267, 4.06238952270]	[3.93433538293, 4.06238952243]
y_5	[17.7073057118, 20.9780581041]	[19.4449019971, 19.5568470724]	[19.4449019974, 19.5568470720]

Table 2: Enclosures for the range of the coordinates of the points in $\mathcal{X}$ for the input parameters: $d_1 = [17.95, 18.05]$, $d_2 = [19.95, 20.05]$, $d_3 = 5$, and $a_1 = 4.5$.

	Computed Enclosure (ia)	Computed Enclosure (ip)	Exact Range
x_1	[0.00000000000, 0.00000000000]	[0.00000000000, 0.00000000000]	[0.00000000000, 0.00000000000]
y_1	[0.00000000000, 0.00000000000]	[0.00000000000, 0.00000000000]	[0.00000000000, 0.00000000000]
x_2	[17.5000000000, 18.5000000000]	[17.5000000000, 18.5000000000]	[17.5000000000, 18.5000000000]
y_2	[0.00000000000, 0.00000000000]	[0.00000000000, 0.00000000000]	[0.00000000000, 0.00000000000]
x_3	[13.1342512426, 14.4390848782]	[13.1786861116, 14.3894819112]	[13.1786861116, 14.3894819112]
y_3	[18.8318118652, 20.2454266548]	[19.0618372944, 20.0393674122]	[19.0618372944, 20.0393674122]
x_4	[8.03496723503, 9.73618604693]	[8.29103552329, 9.50183132283]	[8.29103552333, 9.50183132280]
y_4	[15.4236356975, 21.3006110186]	[18.0078582972, 18.9853884151]	[18.0078582973, 18.9853884150]
x_5	———	[3.35210247128, 4.65195193880]	[3.36141910047, 4.64208611010]
y_5	———	[18.9281154190, 20.0687588103]	[18.9388804473, 20.0584618519]

Table 3: Enclosures for the range of the coordinates of the points in $\mathcal{X}$ for the input parameters: $d_1 = [17.5, 18.5]$, $d_2 = [19.5, 20.5]$, $d_3 = 5$, and $a_1 = 4.5$.

$\boldsymbol{d}_1$	$\boldsymbol{d}_2$	$\boldsymbol{d}_3$	$\boldsymbol{a}_1$	$\varepsilon_{\mathcal{X}}$	ε_r	STU
18	20	5	4.5	-	-	6.4
$[17.995, 18.005]$	$[19.995, 20.005]$	5	4.5	10^{-2} 10^{-3} 10^{-4}	1 10^{-1} 10^{-2}	24.4 37.6 105.8
$[17.95, 18.05]$	$[19.95, 20.05]$	5	4.5	10^{-1} 10^{-2} 10^{-3}	1 10^{-1} 10^{-2}	24.3 37.6 105.8
$[17.5, 18.5]$	$[19.5, 20.5]$	5	4.5	1 10^{-1} 10^{-2}	1 10^{-1} 10^{-2}	117 295 595

Table 4: Execution times in Standard Time Units (STU) for different assignments and required accuracies $\varepsilon_{\mathcal{X}}$.

for the coordinates of point P_5. Our algorithm, however, computes the enclosures for the range of the coordinates at the required accuracy $\varepsilon_{\mathcal{X}}$.

Finally, we fed the algorithm with different accuracies and recorded the corresponding run times. Times are normalized with respect to the width of the interval input data $\boldsymbol{V} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m)$ by defining the relative error

$$\varepsilon_r = \max_{1 \le i \le m} \frac{w(\boldsymbol{v}_i)}{\varepsilon_{\mathcal{X}}}$$

The measured times are made computer independent by using Standard Time Units (STU), [36]. One STU is the time spent for 1000 real evaluations of the Shekel-5 function in $(4, 4, 4, 4)$.

In Table 4 we present the results obtained for the assignment of thin intervals $d_1 = 18, d_2 = 20, d_3 = 5$ and $a_1 = 4.5$, and for the assignments of Tables 1 to 3 at different accuracies. As expected, the execution time increases with the width of the intervals, and with smaller accuracies. Note, however, that the times remain stable for small variations of the interval input data where few steps of the global optimization approach are required.

9 Conclusions and Future Work

In this work, we have developed an interval propagation algorithm to compute tight enclosures for the range of a sequence of geometric functions. The algorithm ConstructPlan combines two different approaches: a gradient-based method, and an optimization method. First, we compute the gradient and, according to the information contained in the components of the gradient, we compute an enclosure for the lower and the upper bounds of the function. If the resulting enclosures are not tight, we apply an optimization method in order to get the desired accuracy. We have implemented the algorithm and run it for several cases. Its performance is correct and the accuracy of the results is often much better than the demanded precision.

The main feature of the algorithm ConstructPlan is that, instead of solving the geometric problem in the solution space (like most global optimization approaches), it is solved in the parameters space. This means that the space where the parameters take values is iteratively reduced until the range of the geometric functions reaches the desired accuracy. Using this methodology, ConstructPlan has the following properties: 1) it does not require an initial guess box for the coordinates of the points; 2) it computes the enclosures for the endpoints of the range with arbitrarily small overestimation and computes, therefore, a tight enclosure for the range; and 3) it guarantees that the enclosures for the range contain all possible realizations of the object for the interval input data.

The experiments conducted show that our approach yields ranges very close to the exact solution to the geometric construction problem, whereas interval arithmetic deviates by much after even a small number of construction steps. Therefore, computing tight bounds is an absolute necessity for practical problems. But the question whether it is generally sufficient remains open. We plan to investigate where the limits of the method are with respect to the number of construction steps.

Acknowledgements

The authors wish to thank Prof. G. Heindl for many fruitful discussion and helpful remarks during the development of this work. This research has been partially supported by the Spanish Research Agency CICYT, under the project TIC98-0586-C03-03. N. Mata is also partially supported by a PhD fellowship of the Associació d'Enginyers Industrials de Catalunya.

References

[1] G. Alefeld and J. Herzberger. *Introduction to Interval Computations.* Computer Science and Applied Mathematics. Academin Press, 1983.

[2] N.S. Asaithambi, Shen Zuhe, and R.E. Moore. On computing the range of values. *Computing*, 28:225–237, 1982.

[3] F. Benhamou and P. van Hentenryck. Introduction to the special issue on interval constraints. *Constraints*, 1997.

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

[5] A. Bowyer. *SvLIS. Introduction and user manual.* Information Geometers, 1994.

[6] A. Bowyer, J. Berchtold, D. Eisenthal, I. Voiculescu, and K. Wise. Interval methods in geometric modelling. In *Geometric Modelling and Processing*, Hong Kong, China, April 2000. IEEE Computer Society.

[7] B.D. Brüderlin. Using geometric rewrite rules for solving geometric problems symbolically. In *Theoretical Computer Science 116*, pages 291–303. Elsevier Science Publishers B.V., 1993.

[8] J.G. Cleary. Logical arithmetic. *Future Computing Systems*, 2(2):125–149, 1987.

[9] J.F. Dufourd, P. Mathis, and P. Schreck. Geometric construction by assembling solved subfigures. *Artificial Intelligence*, 99:73–119, 1998.

[10] 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.

[11] L. Guibas, D. Salesin, and J. Stolfi. Epsilon geometry: Building robust algorithms from imprecise computations. In *5th ACM Symposium on Computational Geometry*, pages 208–217, 1989.

[12] R. Hammer, M. Hocks, U. Kulisch, and D. Ratz. *Numerical Toolbox for Verified Computing I.* Springer-Verlag, 1993.

[13] E. Hansen. *Global Optimization using Interval Analysis.* Marcel Dekker, Inc., 1992.

[14] R. Joan-Arinyo and N. Mata. Basic geometric operations in constructive constraint solvers using interval arithmetic. In *International Conference on Interval Methods and their Applications on Global Optimization (INTERVAL'98)*, pages 6–8, April 1998.

[15] R. Joan-Arinyo and N. Mata. How to improve estimations of geometric construction problems with one interval parameter. In *Applications of Interval Analysis to Systems and Control with special emphasis on recent advances in Modal Interval Analysis (MISC'99)*, pages 69–83, Girona, Catalonia, Spain, 24-26 February 1999.

[16] R. Joan-Arinyo and N. Mata. Applications of the geometric construction problem with interval parameters. In *International Conference on Interval Methods in Science and Engineering (SCAN'2000 / INTERVAL'2000)*, Karlsruhe, Germany, September 2000.

[17] R. Joan-Arinyo and N. Mata. A data structure for solving geometric construction problems with interval parameters. Technical Report LSI-00-24-R, Department LSI, Universitat Politècnica de Catalunya, 2000.

[18] R. Joan-Arinyo and N. Mata. An interval propagation algorithm for geometric construction problems. Technical Report LSI-00-25-R, Department LSI, Universitat Politècnica de Catalunya, 2000.

[19] R. Joan-Arinyo and N. Mata. Solving geometric problems with interval parameters by means of symbolic construction plans. In 6^{th} *IMACS International Conference on Applications of Computer Algebra (IMACS ACA 2000)*, pages 86–87, St. Petersburg, Russia, 25-28 June 2000.

[20] R. Joan-Arinyo, N. Mata, and T. Soto-Riera. Basic operations and functions in constructible geometric problems. Technical Report LSI-00-23-R, Department LSI, Universitat Politècnica de Catalunya, 2000.

[21] R. Joan-Arinyo and A. Soto-Riera. Combining constructive and equational geometric constraint solving techniques. *ACM Transactions on Graphics*, 18(1):35–55, January 1999.

[22] T. Kao and G. Knott. An efficient and numerically correct algorithm for the 2D convex hull problem. *BYTE*, 30:311–331, 1990.

[23] R.B. Kearfott. *Rigorous Global Search: Continuous Problems.* Kluwer Academic Publications, 1996.

[24] R. Klatte, U. Kulisch, M. Neaga, D. Ratz, and Ch. Ullrich. *PASCAL-XSC*. Springer-Verlag, 1991.

[25] W. Krämer, U. Kulisch, and R. Lohner. *Numerical Toolbox for Verified Computing II*. Springer-Verlag, 2000. To appear.

[26] N. Mata. Solving incidence and tangency constraints in 2D. Technical Report LSI-97-3R, Department LSI, Universitat Politècnica de Catalunya, 1997.

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

[28] R.E. Moore. *Methods and Applications of Interval Analysis*. SIAM Studies in Applied Mathematics. SIAM, Philadelphia, PA, 1979.

[29] S.P. Mudur and P.A. Koparkar. Interval methods for processing geometric objects. *IEEE Computer Graphics & Applications*, 4(2):7–17, February 1984.

[30] J.C. Owen. Algebraic solution for geometry from dimensional constraints. In *ACM Symp Foundations of Solid Modeling*, pages 397–407, Austin, TX, 1991.

[31] H. Ratschek and J. Rokne. *New Computer Methods for Global Optimization*. Ellis Horwood Series in Mathematics and its Applications. Ellis Horwood Limited, Chichester, UK, 1988.

[32] H. Ratschek and J. Rokne. Box-sphere intersection tests. *Computer-Aided Design*, 26(7):579–584, July 1994.

[33] D. Ratz. An inclusion algorithm for global optimization in a portable PASCAL-XSC implementation. In L. Atanassova and J. Herzberger, editors, *Computer Arithmetic and Enclosure Methods*, pages 329–338. IMACS, Elsevier Science Publishers B.V. (North-Holland), 1992.

[34] M. Segal. Using tolerances to guarantee valid polyhedral modeling results. *Computer Graphics*, 24(4):105–114, 1990.

[35] A. Soto-Riera. *Geometric Constraint Solving in 2D*. PhD thesis, Dept. LSI, Universitat Politècnica de Catalunya, Barcelona, Catalonia, 1998. Written in Catalan.

[36] A. Törn and A. Žilinskas. *Global Optimization*, volume 350 of *Lecture Notes in Computer Science*. Springer-Verlag, 1989.

[37] A. Verroust, F. Schonek, and D. Roller. Rule-oriented method for parameterized computer-aided design. *Computer-Aided Design*, 24(10):531–540, October 1992.