Interval Computations
No 2, 1993

# Solving Large Systems of Nonlinear Constraints with Application to Data Modeling

Gregory D. Hager

Many applications in remote sensing require fitting a model to observed data and making a decision based on the resulting model parameters and their uncertainty. When observation errors are bounded, interval-based constraint-solving methods provide a general, robust method for computing approximations to the set of models that fit observed data, and determining the appropriate decision to make about the model.

This paper formalizes the notion of decision-making from observed data. An abstract class of interval bisection algorithms is defined. These algorithms are shown to be correct for the entire class of decision-making problems, and to terminate in finite time for all but a small set of problems. A practical realization of the algorithm is described, and experimental results on simulated data are presented.

# Решение больших систем нелинейных ограничений с приложением к моделированию данных

Г. Д. Хагер

Многие приложения в дистанционном распознавании требуют подгонки модели к наблюдаемым данным и принятия решения, основанного на параметрах полученной модели и их неопределенности. Когда погрешности наблюдения ограничены, интервальные методы решения ограничений (равенств и неравенств) позволяют получить надежный общий метод вычисления приближений к множеству моделей, удовлетворяющих полученным в результате наблюдений данным, и определения подходящего для модели решения.

В статье формализуется понятие принятия решения исходя из полученных в результате наблюдения данных. Определен абстрактный класс интервальных алгоритмов деления пополам. Показано, что эти алгоритмы корректны для целого класса задач принятия решений и завершаются за конечное время для всех задач, за исключением небольшого множества. Описывается практическая реализация этого алгоритма; представлены экспериментальные результаты на моделированных данных.

# 1 Introduction

Many applications in the area of remote sensing require modeling and interpreting large sets of noisy sensor data. There is an extensive literature on the theory and practice of data modeling and interpretation when models are linear in their parameters and sensing errors are statistically well-behaved. However, many problems involve nonlinear models and ill-behaved sensor data errors. Achieving efficient, reliable data modeling for these problems can be extremely difficult. In addition, data interpretation often includes a calculation of the accuracy of the fitted model that is also complicated by nonlinearities and ill-behaved error statistics.

This article describes a reliable, simple-to-use algorithm for solving nonlinear, ill-behaved data interpretation problems. The major restriction on problems is that sensing error must be bounded. In this case, each observed datum imposes a constraint on the parameterization of a data model. These constraints are expressed by equations and inequalities which are solved using constraint satisfaction techniques. In addition, the algorithm accepts a *decision criterion* expressed as an inequality on model parameters. Constraint solving is performed until the satisfiability of the decision criterion is determined for every parameter value consistent with the observed data.

The original motivation for developing these algorithms comes from the domain of robotic sensing and manipulation. For example, in a manufacturing domain, there may be a wide variety of shafts, gears, and fasteners that are all exact copies of one another modulo specific parameters such as size, length, number of teeth, and so forth. Given a model for manufactured parts, the problem is to decide what the parts are, and to compare them with one another or to classify them. For example, the goal may be to order gears based on size, or to match gears with shafts of the appropriate size. Each decision involves solving for the set of model parameters from sensor data.

Problems of this type commonly have the following general attributes:

- There are many more equations than unknowns. It is common that 10 to 15 parameters are to be determined from well over 1,000 constraint equations.
- The precision to which the model parameters can be computed is ultimately governed by the precision and completeness of the sensor data. Sensor data errors are often bounded, but these bounds are typically several orders of magnitude larger than machine precision. The size of

these bounds as well as the density and/or completeness of the sensor data can be expected to vary widely in normal operating conditions.

- Model parameters need only be determined to the precision required to determine the satisfiability of a set of constraints describing task-specific decision criteria. Consequently, the amount of effort devoted to constraint solving can be expected to vary according to the size and quality of the data set as well as the physical attributes of the observed system.

The algorithm used to solve these problems employs interval bisection methods. The most important aspects of this algorithm are:

- The algorithm is *set-based*: the set of solutions to the systems of equations it manipulates are typically sets of nonzero measure in the model parameter space.

- As noted above, the goal of the algorithm is to determine set relationships, not to solve equations to an absolute precision. The algorithm can be shown to compute correct solutions. Given some additional conditions placed on the structure of the data modeling problem, the algorithm can be shown to terminate with a solution in finite time.

- The structure of the decision constraints is used to guide the bisection process so that the computation needed to reach a decision is reduced. The amount of computation performed varies adaptively based on the character of the data and the system of constraints.

The next section describes an example problem in detail and then formally states the general data modeling problem to be solved. Section 3 presents some basic results in interval methods for solving nonlinear equations. This section may be omitted on first reading and referred to as required. Section 4 presents an abstract algorithm for solving set-based decision-making problems, and shows that it is correct and that it terminates in finite time for all but a small set of problems. Section 5 describes an implemented algorithm. Section 6 describes some experimental results, and Section 7 summarizes the work and indicates possible extensions. Two related publications, [Hager 1992b] and [Hager & Lu 1993], describe more extensive experimental results with real data and the constraint solving software system, respectively.

# 2 Problem definition

In order to motivate the general class of problems addressed in this paper, consider the following simple example. A tray of ball bearings is imaged by a video camera. The images from the camera are processed to recover the outlines of the bearings in the image. The image of a single bearing in the camera image is a circle, and so it can be modeled by the equation

$$h(c_1, c_2, r, u, v) = (c_1 - u)^2 + (c_2 - v)^2 - r^2 = 0. \tag{1}$$

The vector $p = (c_1, c_2, r)$, consisting of the location of the center and the radius of the model, is referred to as the *parameter vector* of the model. Given the physical constraints of the imaging equipment, parameter vectors can normally be restricted to come from a set $\mathcal{P}$ referred to as the *parameter space* of the model. In the example above, the parameter space might be defined as $\mathcal{P} = \big\{(c_1, c_2, r) \,|\, |c_1| \leq 100, |c_2| \leq 100, 1 \leq r \leq 10\big\}$ indicating the expected range of circle locations and sizes.

The sensor — in this case the video camera and associated data processing algorithms — returns information about the location $z = (u, v)$ of points on the outlines of ball bearings in the camera image. However, a camera image is a discrete set of cells, so the sensor data is inherently quantized and has limited accuracy. If the width of a sensor cell is $2t$, then it can be assumed that an observed value $z = (u, v)$ only restricts the position of an outline to the set of locations $\mathbf{z} = \big\{(u', v') \,|\, |u - u'| \leq t, |v - v'| \leq t\big\}$. The latter is referred to as an observation's *uncertainty set*. If the tolerance $t$ is large enough, then the uncertainty set is guaranteed to contain the observation that would have been generated by a perfect (i.e. error-free) sensor.

The basic goal of data modeling in robotic sensing is to infer the parameterization of a data model from uncertain observations. The set of model parameters consistent with a single observation $z$ with uncertainty set $\mathbf{z}$ is:

$$S(h, \mathcal{P}, \mathbf{z}) = \bigcup_{z \in \mathbf{z}} \big\{p \in \mathcal{P} \,|\, h(p, z) = 0\big\}. \tag{2}$$

The parameter vectors consistent with a collection of observations $z_1$, $z_2$, ..., $z_n$ with uncertainty sets $\mathbf{z}_1$, $\mathbf{z}_2$, ..., $\mathbf{z}_n$ is

$$S^* = \bigcap_{1 \leq i \leq n} S(h, \mathcal{P}, \mathbf{z}_i). \tag{3}$$

This set, called the *solution set*, contains the parameter vectors of all surfaces that could have generated the complete data set. If the tolerance used to

define uncertainty sets has been selected correctly, then the physically correct parameters of the observed outline must be in the solution set. Choosing an error tolerance larger than necessary leads to unacceptably large solution sets, so the goal is to select a value which is no more than a slight overestimation of the minimal, correct value. It is also important to note that even if there were no errors in observed data ($t = 0$), *incompleteness* of the data could still cause the solution set to be large. For example, acquiring only two data points on a circle is not enough to fully specify model parameters, even if there are no data errors. In general, the size of the solution set depends on both the magnitude of data errors and the completeness of the observed data.

In this article the objective of data modeling is to make a decision about the observed object. For example, suppose the bearings mentioned above must be placed into a retainer with an outline of radius $s$. This is possible if and only if $r < s$. The solution set summarizes all the information known about $r$ from observations. Hence, it is only possible to guarantee that $r < s$ for the real outline if every parameter vector in the solution set satisfies this constraint. Conversely, if no parameter vector in the solution set satisfies this constraint, then the real outline definitely does not satisfy it. If neither case holds, then no decision can be made — the problem is fundamentally ambiguous.

This basic problem setting can be generalized to cover a wide range of data interpretation problems. To frame the general problem more precisely, the complete set of available data values will be represented by the vector $x \in \Re^s$, and the corresponding vector of uncertainty sets will be denoted by $\mathbf{x}$. The function $g : \mathcal{P} \times \Re^s \to \Re^q$ will relate observation vectors and model parameters. Note that the example above can be placed into this framework by defining $x = (z_1, z_2, \ldots, z_s)$ and

$$g(p, x) = \begin{bmatrix} h(p, z_1) \\ h(p, z_2) \\ \vdots \\ h(p, z_s) \end{bmatrix}.$$

The *solution set* is now defined as:

$$S(g, \mathcal{P}, \mathbf{x}) := \begin{cases} \{p \in \mathcal{P} \mid g(p, x) = 0 \quad \text{for some } x \in \mathbf{x}\} & \mathbf{x} \neq \emptyset; \\ \{p \in \mathcal{P} \mid g(p) = 0\} & \text{otherwise.} \end{cases} \tag{4}$$

Let $a : \mathcal{P} \to \Re$ be a function such that a parameter vector $p$ is deemed to

support a decision if and only if $a(p) < 0$.[1] In the example described above, $a(c_1, c_2, r) = r - s$. Define the *supporting set* of $a$ as

$$D(a, \mathcal{P}) = \big\{ p \in \mathcal{P} \,|\, a(p) < 0 \big\}.$$

An *instantiation* of a decision problem is a four-tuple $I = \langle g, \mathcal{P}, \mathbf{x}, a \rangle$. The *decision value* of an instantiation $I = \langle g, \mathcal{P}, \mathbf{x}, a \rangle$ is defined as:

$$V(I) := \begin{cases} \textbf{true} & \text{if } S(g, \mathcal{P}, \mathbf{x}) \subset D(a, \mathcal{P}); \\ \textbf{false} & \text{if } S(g, \mathcal{P}, \mathbf{x}) \cap D(a, \mathcal{P}) = \emptyset; \\ \textbf{undecided} & \text{otherwise.} \end{cases}$$

The problem considered in this paper is the computation of $V$ from inputs $g$, $a$, $\mathcal{P}$, and $\mathbf{x}$.

# 3 Solving systems of nonlinear equations

## 3.1 Definitions and notation

The following notational conventions apply in the remainder of this article. The closed interval of real numbers from $u$ to $v$, $u \leq v$, is denoted by $[u, v]$. If $U$ and $V$ are vectors in $\Re^n$, $U_i \leq V_i, \quad i = 1 \ldots n$, then the set $[U, V] = [U_1, V_1] \times \cdots \times [U_n, V_n]$ is regarded as an interval vector in $\Re^n$. Point-valued and interval-valued variables are distinguished by writing the latter in boldface type. No special distinction is made between scalar and vector quantities. Given an interval or interval vector $\mathbf{x}$, $\underline{\mathbf{x}}$, and $\overline{\mathbf{x}}$ are the real-valued lower and upper values of $\mathbf{x}$, respectively. That is, $\mathbf{x} = [\underline{\mathbf{x}}, \overline{\mathbf{x}}]$. When a point value $x$ appears with interval values in an expression, $x$ should be thought of as the degenerate interval $\mathbf{x} = [x, x]$. The center of an interval $\mathbf{x}$ is defined as $\widehat{\mathbf{x}} := (\overline{\mathbf{x}} + \underline{\mathbf{x}})/2$, and the width of an interval $\mathbf{x}$ is defined as $w(\mathbf{x}) := \overline{\mathbf{x}} - \underline{\mathbf{x}}$. An interval vector is called *nontrivial* if the width of every component is nonzero.

Define an *interval covering* $\mathbf{P}$ of a set $\mathcal{P}$ to be a finite set of intervals such that $\mathcal{P} \subseteq \bigcup_{\mathbf{p} \in \mathbf{P}} \mathbf{p}$. The covering will be called *exact* if equality holds. In

[1] It is also possible for $a$ to be a vector function with the interpretation that a vector $p$ is in the supporting set if every component of $a(p)$ is negative. However, this modification complicates the analysis at some points without substantially altering the final results.

the sequel $\mathbf{P}$ will always denote an arbitrary interval covering of $\mathcal{P}$ unless otherwise noted.

Let $\|\cdot\|$ denote the $L_\infty$ norm. For sets $S_1$ and $S_2$ in $\Re^n$, define

$$d(S_1, S_2) := \sup_{p \in S_1} \inf_{q \in S_2} \|p - q\|.$$

If $S \subset \Re^n$ and $p \in \Re^n$, the expression $d(p, S)$ is shorthand for $d\big(\{p\}, S\big)$. For any set $S$, let $int(S)$, $cl(S)$, and $bdy(S)$ denote the interior, closure, and boundary of $S$, respectively.

It is assumed that all arithmetic evaluations in this article are carried out using exact arithmetic. The *exact interval evaluation* of a function $h$ at $\mathbf{x}$ is defined as

$$\tilde{h}(\mathbf{x}) := \big\{ y \,|\, y = h(x) \quad \text{for some } x \in \mathbf{x} \big\}.$$

It is assumed that functions with interval arguments are computed using standard definitions of interval arithmetic, e.g. that found in [Alefeld & Herzberger 1983] or [Neumaier 1990]. Such evaluations must satisfy two basic criteria for a function $h : A \to \Re^n$ evaluated at $\mathbf{x} \subseteq A$ :

- $\tilde{h}(\mathbf{x}) \subseteq h(\mathbf{x})$.
- There is a value $\beta > 0$ such that for all $\mathbf{x} \subseteq A$,

$$\|w(h(\mathbf{x}))\| \leq \beta \, \|w(\mathbf{x})\| \, .$$

The latter condition is true of functions that are Lipschitz continuous on $A$ and are evaluated using interval arithmetic [Neumaier 1990, Chapter 2].

## 3.2 Constraint solving applied to decision making

It is henceforth assumed that the components of any instantiation $I = \langle g, \mathcal{P}, \mathbf{x}, a \rangle$ have the following properties:

1. $\mathcal{P}$ is compact and admits an exact interval covering.
2. $S(g, \mathcal{P}, \mathbf{x})$, $D(a, \mathcal{P})$, and $D(-a, \mathcal{P})$ are non-empty.[2] Note that this, together with the restrictions stated above, implies that $S(a, \mathcal{P}, \emptyset) = bdy\big(D(a, \mathcal{P})\big)$ is also non-empty.

[2]This restriction is not essential, but requiring it simplies the later analysis without substantially altering the ultimate results.

3. There is an $\alpha > 0$ such that for all $p \in \mathcal{P}$ and $x \in \mathbf{x}$,
$$\alpha\, d\big(p, S(g, \mathcal{P}, \mathbf{x})\big) \leq \|g(p, x)\| \quad \text{and} \quad \alpha\, d\big(p, S(a, \mathcal{P}, \emptyset)\big) \leq \|a(p)\|\,.$$
4. There is an expression for the function $g(\mathbf{p}, \mathbf{x})$ such that the variables associated with the components of $\mathbf{x}$ each appear at most once in the expression.

In practice, satisfying conditions 1 and 2 does not generally constitute a problem. Note that the continuity of $g$ combined with the compactness of the parameter space implies that solution sets are compact.

For scalar functions with a single parameter, condition 3 rules out functions where both the function and its derivatives are simultaneously 0 on the boundary of the solution set. For example, suppose $g(p, x) = p^2 - x$. Condition 3 is not satisfied only for data values $\mathbf{x}$ where $\underline{\mathbf{x}} = 0$ or $\overline{\mathbf{x}} = 0$. If $\mathbf{x}$ is a random variable on a subset of the (interval) real line, this is a measure zero event and so condition 3 is satisfied with probability 1. For scalar functions of more than one variable, condition three is equivalent to requiring that the gradient of the function be nonzero on the boundary of the solution set. At present, it is not possible to characterize the set of vector functions that satisfy condition 3.

Condition 4 can often, but not always, be satisfied. The analysis in this article is carried out assuming condition 4 holds, however comments on the implications of not satisfying condition 4 appear at the end of Section 4.

Condition 4 has the following consequences:

**Lemma 3.1.** *For any* $\mathbf{p} \subseteq \mathcal{P}$, $g(\mathbf{p}, \mathbf{x}) = \bigcup_{x \in \mathbf{x}} g(\mathbf{p}, x)$.

*Proof:* Construct a function $g'$ by replacing each occurrence of a variable in an expression for $g$ with a new, unique variable name. Since the components of $\mathbf{x}$ appear only once in $g$, there is a one-to-one correspondence between these variables in $g$ and $g'$. Let the vector $\mathbf{p}'$ be the interval vector constructed by appropriate replication of the components of $\mathbf{p}$ so that for any $\mathbf{x}' \subseteq \mathbf{x}$, $g'(\mathbf{p}', \mathbf{x}') = g(\mathbf{p}, \mathbf{x}')$. Every component of both $\mathbf{p}'$ and $\mathbf{x}'$ occurs at most once in the expression of $g'$, so by [Neumaier 1990, Corollary 1.4.4], it follows that

$$g(\mathbf{p}, \mathbf{x}) = g'(\mathbf{p}', \mathbf{x}) = \big\{g'(p', x) \,|\, p' \in \mathbf{p}', x \in \mathbf{x}\big\} = \bigcup_{x \in \mathbf{x}} g'(\mathbf{p}', x) = \bigcup_{x \in \mathbf{x}} g(\mathbf{p}, x).$$

■

**Corollary 3.2.** *For any* $\mathbf{p} \subseteq \mathcal{P}$, *if* $0 \in g(\mathbf{p}, \mathbf{x})$ *then there is an* $x \in \mathbf{x}$ *such that* $0 \in g(\mathbf{p}, x)$.

*Proof:* By the previous lemma, if $0 \in g(\mathbf{p}, \mathbf{x}) = \bigcup_{x \in \mathbf{x}} g(\mathbf{p}, x)$ then there is some $x$ such that $0 \in g(\mathbf{p}, x)$. ■

Given an interval vector $\mathbf{p} \subseteq \mathcal{P}$, if $0 \notin g(\mathbf{p}, \mathbf{x})$ then it follows that $\mathbf{p}$ and the solution set are disjoint. Conversely, the fact that $0 \in g(\mathbf{p}, \mathbf{x})$ does not necessarily imply that $\mathbf{p}$ contains points in the solution set. However, given the properties stated above, it is possible to place an upper bound on the distance between $\mathbf{p}$ and the solution set.

**Lemma 3.3.** *Let* $S^* = S(g, \mathcal{P}, \mathbf{x})$. *There is a* $K > 0$ *such that for any* $\mathbf{p} \subseteq \mathcal{P}$, *if* $0 \in g(\mathbf{p}, \mathbf{x})$, *then* $d(\mathbf{p}, S^*) \leq K \left\| w(\mathbf{p}) \right\|$.

*Proof:* Consider two cases:

$\mathbf{x} = \emptyset$ : By assumption 3 above, for all $p \in \mathbf{p}$, $d(p, S^*) \leq \frac{1}{\alpha} \|g(p)\|$. But $0 \in g(\mathbf{p})$, so $\|g(p)\| \leq \|w(g(\mathbf{p}))\| \leq \beta \|w(\mathbf{p})\|$.

$\mathbf{x} \neq \emptyset$ : By assumption 3 above, for all $p \in \mathbf{p}$, and $x \in \mathbf{x}$, $d(p, S^*) \leq \frac{1}{\alpha} \|g(p, x)\|$. But $0 \in g(\mathbf{p}, \mathbf{x})$, so by Corollary 3.2 there is some $x^* \in \mathbf{x}$ such that $0 \in g(\mathbf{p}, x^*)$. Therefore

$$\|g(p, x^*)\| \leq \|w(g(\mathbf{p}, x^*))\| \leq \beta \|w(\mathbf{p})\|.$$

In both cases, by defining $K = \beta/\alpha$, it follows that

$$d(\mathbf{p}, S^*) = \max_{p \in \mathbf{p}} d(p, S^*) \leq \frac{\beta}{\alpha} \|w(\mathbf{p})\| = K \|w(\mathbf{p})\|.$$

■

Lemma 3.3 is of tremendous practical importance because, as will be seen shortly, it implies that arbitrarily good approximations to the solution set can be constructed without expending constraint solving effort on the vector of observed values, $\mathbf{x}$. Since $\mathbf{x}$ is typically quite large in comparison to $\mathbf{p}$, this property substantially reduces the amount of computational effort expended in constraint solving.

Recall that $\mathbf{P}$ represents an arbitrary, exact interval covering of $\mathcal{P}$. Define

$$S(g, \mathbf{P}, \mathbf{x}) := \{p \,|\, p \in \mathbf{p} \in \mathbf{P} \text{ and } 0 \in g(\mathbf{p}, \mathbf{x})\}$$

and

$$D(a, \mathbf{P}) := \{p \,|\, p \in \mathbf{p} \in \mathbf{P} \text{ and } a(\mathbf{p}) \not\geq 0\}.$$

Note that it follows from the properties of interval arithmetic that

$$S(g, \mathcal{P}, \mathbf{x}) \subseteq S(g, \mathbf{P}, \mathbf{x}) \quad \text{and} \quad D(a, \mathcal{P}) \subseteq D(a, \mathbf{P}).$$

Hence, these operators can be thought of as computing an outer approximation to the solution set and supporting set from an interval covering, respectively.

Define a covering $\mathbf{P}$ to be *$\delta$-fine* if $\|w(\mathbf{p})\| \leq \delta$ for all $\mathbf{p} \in \mathbf{P}$. With these definitions, Lemma 3.3 can be applied to interval coverings as follows:

**Corollary 3.4.** *For any $\epsilon > 0$:*

1. *There is a $\delta_1 > 0$ such that for any $\delta_1$-fine covering $\mathbf{P}$ of $\mathcal{P}$,*

$$d\Big(S(g,\mathbf{P},\mathbf{x}), S(g,\mathcal{P},\mathbf{x})\Big) \leq \epsilon.$$

2. *There is a $\delta_2 > 0$ such that for any $\delta_2$-fine covering $\mathbf{P}$ of $\mathcal{P}$,*

$$d\Big(D(a,\mathbf{P}), D(a,\mathcal{P})\Big) \leq \epsilon.$$

*Proof:* For 1, note that

$$d\big(S(g,\mathbf{P},\mathbf{x}), S(g,\mathcal{P},\mathbf{x})\big) \leq \max_{\mathbf{p}\in\mathbf{P}} d\big(\mathbf{p}, S(g,\mathbf{P},\mathbf{x})\big).$$

By Lemma 3.3, for any $\mathbf{p} \in \mathbf{P}$, $d\big(\mathbf{p}, S(g,\mathbf{P},\mathbf{x})\big) \leq K \left\| w(\mathbf{p}) \right\|$. Choosing $\delta_1 = \epsilon/K$, it follows that

$$d\big(\mathbf{p}, S(g,\mathbf{P},\mathbf{x})\big) \leq K\delta_1 \leq K\frac{\epsilon}{K} = \epsilon.$$

For 2, note that $S(a,\mathcal{P},\emptyset) = bdy\big(D(a,\mathcal{P})\big)$ is nonempty by assumption. Therefore

$$d\big(D(a,\mathbf{P}), D(a,\mathcal{P})\big) \leq d\big(S(a,\mathbf{P},\emptyset), S(a,\mathcal{P},\emptyset)\big)$$

and the conclusion follows from 1. ■

The ultimate aim of this analysis is to make judgements about intersection and inclusion relationships among sets. The following states the central result used in the remainder of this paper:

**Theorem 3.5.** *For arbitrary functions $g$ and $a$,*

$$S(g,\mathcal{P},\mathbf{x}) \cap cl\big(D(a,\mathcal{P})\big) = \emptyset$$

*if and only if there exists a $\delta > 0$ such that for any $\delta$-fine covering $\mathbf{P}$ of $\mathcal{P}$,*

$$S(g,\mathbf{P},\mathbf{x}) \cap D(a,\mathbf{P}) = \emptyset.$$

*Proof:* In the reverse direction, the result follows directly from the fact that $S(g,\mathcal{P},\mathbf{x}) \subseteq S(g,\mathbf{P},\mathbf{x})$ and $D(a,\mathcal{P}) \subseteq D(a,\mathbf{P})$.

In the forward direction, two compact, disjoint sets must be separated by some positive distance $\eta$. Choose a positive value $\epsilon < \eta/2$. From Corollary 3.4, there is a $\delta > 0$ such that for any $\delta$-fine covering $\mathbf{P}$,

$$d\big(S(g,\mathbf{P},\mathbf{x}), S(g,\mathcal{P},\mathbf{x})\big) \leq \epsilon \quad \text{and} \quad d\big(D(a,\mathbf{P}), D(a,\mathcal{P})\big) \leq \epsilon.$$

But, then for any $p_1 \in S(g,\mathbf{P},\mathbf{x})$ and $p_2 \in cl\big(D(a,\mathbf{P})\big)$,

$$\|p_1 - p_2\| \geq \eta - 2\epsilon > 0.$$

Hence, $S(g,\mathbf{P},\mathbf{x}) \cap cl\big(D(a,\mathbf{P})\big) = \emptyset$. ■

**Corollary 3.6.** *Let* $S'(g,\mathbf{P},\mathbf{x}) = \big\{\mathbf{p} \in \mathbf{P} \,|\, 0 \in g(\mathbf{p},\mathbf{x})\big\}$. $S(g,\mathcal{P},\mathbf{x})$ *and* $cl\big(D(a,\mathcal{P})\big)$ *are disjoint if and only if there is a* $\delta > 0$ *such that for any* $\delta$*-fine covering* $\mathbf{P}$ *of* $\mathcal{P}' \subseteq \mathcal{P}$, $\quad D\big(a, S'(g,\mathbf{P},\mathbf{x})\big) = \emptyset$.

The proof follows directly from the previous theorem and the definitions of $S'$ and $D$.

Finally, a result about finding points in the solution set:

**Lemma 3.7.** *If there is a nontrivial interval* $\mathbf{p} \subseteq \mathcal{P}$ *such that* $\mathbf{p} \subseteq S(g,\mathcal{P},\mathbf{x})$ *then there is a* $\delta > 0$ *such that any* $\delta$*-fine covering of* $\mathcal{P}$ *contains an element* $\mathbf{q}$ *with* $0 \in g(\widehat{\mathbf{q}},\mathbf{x})$.

*Proof:* Since $\mathbf{p} \subseteq S(g,\mathcal{P},\mathbf{x})$, $g(p,\mathbf{x}) = 0$ for every $p \in \mathbf{p}$. Define $\delta$ to be one-half the minimum width of the components of $\mathbf{p}$. Since $\mathbf{p}$ is nontrivial, $\delta > 0$. Furthermore, any $\delta$-fine covering has an element $\mathbf{q} \subset \mathbf{p}$, and it follows that $0 \in g(\widehat{\mathbf{q}},\mathbf{x})$. ■

# 4 A bisection algorithm for decision making

The previous section established an upper bound on the distance between a solution set and an interval-based approximation to it when using a simple interval-based consistency test. In practice there are procedures for testing interval vectors that can produce more accurate approximations. This motivates the introduction of a *reduce* operator $R$. $R$ accepts an interval vector of model parameters, an interval vector of observations and a constraint function, and returns an interval vector of model parameters subject to the following conditions:

1. $R(g, \mathbf{p}, \mathbf{x}) \subseteq \mathbf{p}$.
2. $\mathbf{p} \cap S(g, \mathcal{P}, \mathbf{x}) = R(g, \mathbf{p}, \mathbf{x}) \cap S(g, \mathcal{P}, \mathbf{x})$.
3. If $0 \notin g(\mathbf{p}, \mathbf{x})$, then $R(g, \mathbf{p}, \mathbf{x}) = \emptyset$.

Define

$$S_R(g, \mathbf{P}, \mathbf{x}) := \bigcup_{\mathbf{p} \in \mathbf{P}} R(g, \mathbf{p}, \mathbf{x}).$$

**Lemma 4.1.** *For all interval coverings* $\mathbf{P}$ *of* $\mathcal{P}$,

$$S(g, \mathcal{P}, \mathbf{x}) \subseteq S_R(g, \mathbf{P}, \mathbf{x}) \subseteq S(g, \mathbf{P}, \mathbf{x}).$$

*Proof:* The fact that $S(g, \mathcal{P}, \mathbf{x}) \subseteq S_R(g, \mathbf{P}, \mathbf{x})$ follows directly from conditions 1 and 2 above. To show the remaining inclusion, consider an arbitrary $\mathbf{p} \in \mathbf{P}$, such that $R(g, \mathbf{p}, \mathbf{x}) \neq \emptyset$. By condition 3, $0 \in g(\mathbf{p}, \mathbf{x})$. But this means that $R(g, \mathbf{p}, \mathbf{x}) \subseteq \mathbf{p} \subseteq S(g, \mathbf{P}, \mathbf{x})$. Since $\mathbf{p}$ was arbitrary, it follows that $S_R(g, \mathbf{P}, \mathbf{x}) \subseteq S(g, \mathbf{P}, \mathbf{x})$. ■

An analogous operator $E$ can be defined for inequalities. It must satisfy the following conditions:

1. $E(a, \mathbf{p}) \subseteq \mathbf{p}$.
2. $\mathbf{p} \cap D(a, \mathcal{P}) = E(a, \mathbf{p}) \cap D(a, \mathcal{P})$.
3. If $a(\mathbf{p}) \geq 0$ then $E(a, \mathbf{p}) = \emptyset$.

Define

$$D_E(a, \mathbf{P}) := \bigcup_{\mathbf{p} \in \mathbf{P}} E(a, \mathbf{p}).$$

In this case, the following result holds:

**Lemma 4.2.** *For all interval coverings* $\mathbf{P}$ *of* $\mathcal{P}$,

$$D(a, \mathcal{P}) \subseteq D_E(a, \mathbf{P}) \subseteq D(a, \mathbf{P}).$$

The proof is analogous to that of Lemma 4.1.

Let $\texttt{bisect}(\mathbf{n})$ denote a function that returns $\emptyset$ if $\mathbf{n} = \emptyset$, and otherwise returns the set of two subintervals resulting from bisecting some component of $\mathbf{n}$. Let $\texttt{remove}(\mathcal{Q})$ be a function that removes an element from the set $\mathcal{Q}$ and returns that element. Assume that $\mathcal{P}$ is itself an interval vector. An algorithm for computing the decision value of a problem instantiation is:

**Algorithm 4.1.**
$\texttt{Bisect}(I)$:

1. *Initialization*:
   (a) $\mathcal{Q} := \{\mathcal{P}\}$.
2. *Termination Test*:
   (a) if $D_E(-a, \mathcal{Q}) = \emptyset$ return **true**.
   (b) if $D_E(a, \mathcal{Q}) = \emptyset$ return **false**.
   (c) if there are elements $\mathbf{p}_1$ and $\mathbf{p}_2$ in $\mathcal{Q}$ such that:

$$0 \in g(\widehat{\mathbf{p}}_1, \mathbf{x}) \quad \text{and} \quad a(\widehat{\mathbf{p}}_1) < 0,$$
$$0 \in g(\widehat{\mathbf{p}}_2, \mathbf{x}) \quad \text{and} \quad a(\widehat{\mathbf{p}}_2) > 0.$$

   return **undecided**.
3. *Reduction and Bisection*:
   (a) $\mathcal{Q} := \mathcal{Q} \cup \texttt{bisect}(R(\texttt{remove}(\mathcal{Q}), g, \mathbf{x}))$.
   (b) Go to step 2.

The implementation details of this algorithm appear in the next section. The following terms apply to Algorithm 4.1:

- Algorithm 4.1 is *fair* if, for any $\delta > 0$, there is a value $M_\delta$ such that after $M_\delta$ iterations $\mathcal{Q}$ is a $\delta$-fine covering.
- Algorithm 4.1 is *correct* for a set of instantiations $\mathcal{I}$ if for every $I \in \mathcal{I}$ for which $\texttt{Bisect}(I)$ terminates,

$$\texttt{Bisect}(I) = V(I).$$

- Algorithm 4.1 is *finitely complete* for a set of instantiations $\mathcal{I}$ if it terminates within a finite number of iterations for every $I \in \mathcal{I}$.

Not unexpectedly, Algorithm 4.1 is not finitely complete for every instantiation. To see why, consider a problem where the solution set and the supporting set are disjoint, but their closures are not. In this case, it is never possible to separate the approximations to the solution set and the supporting set. Thus, the algorithm will compute forever without reaching an answer. This property is captured in the following:

**Definition 4.1.** *An instantiation $I$ is undecidable if*

$$S(g,\mathcal{P},\mathbf{x}) \cap bdy\big(D(a,\mathcal{P})\big) \neq \emptyset \quad \textit{and} \quad int\big(S(g,\mathcal{P},\mathbf{x})\big) \cap bdy\big(D(a,\mathcal{P})\big) = \emptyset.$$

*An instantiation is decidable if it is not undecidable.*

**Theorem 4.3.** *Algorithm 4.1 is correct for any problem instantiation. If the algorithm is fair, it is finitely complete for all decidable instantiations.*

*Proof:* Correctness follows directly from Lemmas 4.1, 4.2 and Corollary 3.6.

For completeness, consider an arbitrary $I = \langle g, \mathcal{P}, \mathbf{x}, a\rangle$ and let $S^* = S(g,\mathcal{P},\mathbf{x})$, $D^* = D(a,\mathcal{P})$, and $D^c = D(-a,\mathcal{P})$. There are three cases.

$V(I) =$ **false**: Then $S^* \cap D^* = \emptyset$. By Corollary 3.6, there exists a $\delta > 0$ such for that any $\delta$-fine covering $\mathbf{P}$ of $\mathcal{P}' \subseteq \mathcal{P}$, $D\big(a, S'(g,\mathbf{P},\mathbf{x})\big) = \emptyset$. This property is preserved by the operators $R$ and $E$. Since Algorithm 4.1 is fair, there is some number $M_\delta$ such that after $M_\delta$ iterations, $\mathcal{Q}$ is a $\delta$-fine covering. Hence the algorithm will terminate after no more than $M_\delta$ iterations with value **false**.

$V(I) =$ **true**: Note that for decidable problems, $S^* \cap D^c = \emptyset$ if and only if $S^* \cap cl(D^c) = \emptyset$. Hence, $S^* \subset D^*$ if and only if $S^* \cap D^c = \emptyset$. Repeat the argument for the previous case with $D^c$ in place of $D^*$.

$V(I) =$ **undecided**: Since $I$ is decidable, the interior of $S^*$ contains points from both $D^*$ and $D^c$. Thus, it is possible to choose two nontrivial intervals $\mathbf{p}_1 \subset S^* \cap D^*$ and $\mathbf{p}_2 \subset S^* \cap D^c$.

By Lemma 3.7 there is a $\delta > 0$ such that any $\delta$-fine covering of $S^*$ contains an element $\mathbf{q}_1 \subseteq \mathbf{p}_1$ with $0 \in g(\widehat{\mathbf{q}}_1, \mathbf{x})$ and an element $\mathbf{q}_2 \subseteq \mathbf{p}_2$ with $0 \in g(\widehat{\mathbf{q}}_2, \mathbf{x})$. Furthermore, $a(\widehat{\mathbf{q}}_1) < 0$ and $a(\widehat{\mathbf{q}}_2) > 0$. Since Algorithm 4.1 is fair, there is some number $M_\delta$ such that after $M_\delta$ iterations, $\mathcal{Q}$ is a $\delta$-fine covering which contains $S^*$. Hence the algorithm will terminate after no more than $M_\delta$ iterations with value **undecided**.

■

Except for certain artificial constructions that seldom occur in practice, for fixed $g$, $a$, and $\mathcal{P}$, the set of observation vectors leading to undecidable instantiations is a set of measure zero in the set of all possible observation vectors. Consequently, Algorithm 4.1 will terminate for all but a vanishing small set of problem instantiations.

It is worth considering what would happen if condition 4, stated at the beginning of Section 3.2 failed, i.e. if $g(p, \mathbf{x}) \neq \tilde{g}(p, \mathbf{x})$. Lemma 3.7 would fail making it possible that the algorithm would return **undecided** when the correct decision is **true** or **false**. However, the algorithm will still never return **true** when the answer is **false** or vice-versa. The possibility of spurious **undecided** answers could be minimized by using more advanced means, e.g. Gauss-Newton style methods, of checking for the existence of roots at step 2c of Algorithm 4.1. Note that any problem instantiation not satisfying condition 4 can be transformed to one that does by a process of renaming observation variables so that they appear uniquely, and replicating the components of $\mathbf{x}$ appropriately. The new instantiation would satisfy condition 4, and hence Algorithm 4.1 will terminate with an answer unless the problem instantiation is undecidable. Thus, the only effect of dropping condition 4 is to increase the number of problem instantiations for which the algorithm returns **undecided**.

# 5 A realization of bisection-based decision making

The previous section described the essential structure of a decision-making algorithm based on interval bisection. In order to implement the algorithm, the operators $R$, `remove`, and `bisect` must be implemented. In practice, the construction of these procedures has a substantial effect on the running time of the algorithm. This section discusses some practical considerations governing these choices, and describes methods found to work well in practice.

## 5.1 The $R$ operator

The most important consideration for the $R$ operator is its ability to reject intervals that are outside the solution set as quickly as possible. A poor choice of $R$ can lead to excessive bisection and a combinatorial explosion of the number of intervals in $\mathcal{Q}$. This behavior is exacerbated by

the fact that the solution set typically has finite measure in the model parameter space. Thus, there is a potentially large set of intervals that lie on or near the boundary of the solution set. Allowing these intervals to multiply without rejection or reduction can exponentially increase the number of intervals that must be processed before a decision can be reached. For this reason, implementing the $R$ operator based on the definition given in Section 4 is impractical. For problems of more than a few parameters, the number of intervals to be processed grows rapidly and convergence is extremely slow.

Much of the literature devoted to solving systems of nonlinear equations focuses on variations of Newton-like equation solving methods [Alefeld & Herzberger 1983, Neumaier 1990]. The major advantage of these algorithms is their asymptotically quadratic convergence. However, as noted previously, most of the problems that arise in the domain of data modeling have many more equations than unknowns, and they often involve a mix of equalities and inequalities. Furthermore, the system of equations cannot be guaranteed to be of full rank. These considerations eliminate most Newton-like algorithms that rely on generating and inverting the full system Jacobian.

[Alefeld & Herzberger 1983, Chapter 22] describes Newton-like methods that do not require an explicit inverse Jacobian, but still have asymptotically quadratic convergence. With minor modifications, these methods can be generalized to solve rectangular systems of equations. However, testing of these methods on a variety of systems showed their *initial* rate of convergence to be extremely slow. Since in practice many decisions do not require extremely accurate approximations of the solution set, the initial convergence rate of any method is as important as its asymptotic performance. The slow initial convergence of Newton-like methods is fundamentally due to the fact that the Taylor expansion of a system of equations can greatly overestimate the range of a function until the parameter intervals are quite small.

If Taylor series expansions are to be avoided, then the only choice is to use some type of constraint testing based on interval evaluation of $g$. The major observation leading to the operator used in practice is that direct constraint testing can work well *if bisection is done well*. Unfortunately, there is no way of guaranteeing that bisection will always be done well — that is, there is no way of predicting how to best bisect interval vectors to improve a solution set approximation. However, by performing all bisections and testing each of the resulting interval vectors, an algorithm is guaranteed of capitalizing on any possible interval rejection due to bisection. Furthermore,

while performing this lookahead, it is possible to gather information about the efficacy of further bisection. These observations motivated the following definition of $R$:

R$(g, \mathbf{p}, \mathbf{x})$

1. For each component $i$, $1 \leq i \leq n$, trisect[3] $\mathbf{p}$ in $i$, yielding interval vectors $\mathbf{p}_{1,1}, \mathbf{p}_{1,2}, \ldots, \mathbf{p}_{n,2}, \mathbf{p}_{n,3}$.
2. For each $\mathbf{p}_{i,j}$, if $0 \notin g(\mathbf{p}_{i,j}, \mathbf{x})$ then $\mathbf{p}_{i,j} := \emptyset$.
3. $\mathbf{p} := \bigcap_{1 \leq i \leq n} \bigcup_{1 \leq j \leq 3} \mathbf{p}_{i,j}$.

Empirical tests performed on a variety of multivariate polynomials, geometric forms such as circles, ellipses, ellipsoids, and superquadrics [Hager 1992c, Solina & Bajcsy 1990], and the models used in this article have confirmed that, during the initial iterations of bisection, this operator causes a faster reduction in the volume of the solution set approximation than any Newton-like operator investigated. The operator has the additional advantage that it can be easily modified to accommodate strict and nonstrict inequalities. It is simple to parallelize since constraint testing can be performed independently on the $3n$ subintervals created.

## 5.2 Choosing the dimension to bisect

The main purpose of bisection is to enhance the ability of $R$ to perform constraint solving. For example, suppose that $\mathbf{g}(\mathbf{x}) = \mathbf{x}_1 + 2 * \mathbf{x}_2$ with $\mathbf{x}_1 = \mathbf{x}_2 = [-1, 1]$. Bisecting $\mathbf{x}_1$ leads to values for $\mathbf{g}$ of $[-3, 2]$ and $[-2, 3]$ whereas bisecting $\mathbf{x}_2$ yields $[-3, 1]$ and $[-1, 3]$. The latter bisection has less overlap between the computed intervals and is more likely to enable rejection of some interval vector of parameters. Thus, a good quantity to base bisection on is the change in the widths of the components of $g$ relative to changes in the widths of the input vector. This is estimated by computing a *sensitivity* value as follows: for an interval vector $\mathbf{p}$, let $\mathbf{p}_i^c$ be $\mathbf{p}$ with component $i$ replaced by $\widehat{\mathbf{p}}_i$. Define the sensitivity for component $i$ as

$$s_i(g, \mathbf{p}, \mathbf{x}) = \max_{1 \leq j \leq q} \left[ 1 - \frac{w\big(g_j(\mathbf{p}_i^c, \mathbf{x})\big)}{w\big(g_j(\mathbf{p}, \mathbf{x})\big)} \right].$$

[3]Trisection is the division of a node into three equal parts by division of single component.

Sensitivity values range from 0 to 1 with values near 1 indicating that the parameter component has a relatively large effect on the constraint function. Bisection is performed on the parameter component with maximum sensitivity. Intuitively, sensitivity-based bisection will lead to interval shapes that "balance" the effect of different components of $g$. Note that if $s_i(g, \mathbf{p}, \mathbf{x}) = 0$, then the system of constraints is independent of $\mathbf{p}_i$, and there is no reason to bisect this parameter.

Could sensitivity bisection constantly neglect splitting some component of an interval vector? To frame this question precisely, think of each application of bisection as a node in a tree. The parent is the original interval vector, and the two children are the interval vectors returned from bisection. The infinite tree constructed by repeated application of bisection will be termed the *bisection tree*. Intervals appearing anywhere below an interval $\mathbf{p}$ will be referred to as *descendents* of $\mathbf{p}$. Bisection will be termed *fair* on an interval $\mathbf{p}$ if, for every $\delta > 0$, there is a level $M_\delta$ in the bisection tree where the descendents at that level are a $\delta$-fine covering of $\mathbf{p}$.

**Lemma 5.1.** *Sensitivity bisection is fair on any interval vector* $\mathbf{p}$ *where, for any nontrivial* $\mathbf{p}' \subseteq \mathbf{p}$, $\quad 0 < s_i(g, \mathbf{p}', \mathbf{x}) < 1, \quad 1 \le i \le n.$

*Proof:* Suppose that sensitivity bisection is not fair on an arbitrary interval vector $\mathbf{p}$. Then there is some $\delta > 0$ and nontrivial interval $\mathbf{q} \subseteq \mathbf{p}$ such that $\|w(\mathbf{q})\| > \delta$, and $\mathbf{q}$ can be expressed as two subvectors $\mathbf{q} = (\mathbf{b}; \mathbf{n})$ where $\mathbf{b}$ contains only components of $\mathbf{q}$ that are fairly bisected, and $\mathbf{n}$ contains only components that are never bisected below $\mathbf{q}$. Let $i_1, \ldots, i_n$ be the indices of components of $\mathbf{n}$ in $\mathbf{q}$ and $j_1, \ldots, j_m$ be the indices of the components of $\mathbf{b}$ in $\mathbf{q}$. For $\mathbf{b}' \subseteq \mathbf{b}$, define

$$\underline{s}(\mathbf{b}') = \min_{k=i_1 \ldots i_n} s_k\big(g, (\mathbf{n}; \mathbf{b}'), \mathbf{x}\big) \quad \text{and} \quad \overline{s}(\mathbf{b}') = \max_{k=j_1 \ldots j_m} s_k\big(g, (\mathbf{n}; \mathbf{b}'), \mathbf{x}\big).$$

Note that if $\overline{s}(\mathbf{b}') - \underline{s}(\mathbf{b}') < 0$, it follows that

$$s_k\big(g, (\mathbf{n}; \mathbf{b}'), \mathbf{x}\big) > s_l\big(g, (\mathbf{n}; \mathbf{b}'), \mathbf{x}\big), \quad k = i_1, \ldots, i_n \text{ and } l = j_1, \ldots, i_m.$$

Hence, some component of $\mathbf{n}$ will be chosen for bisection.

By definition, if $\|w(\mathbf{b}')\| = 0$ then $\overline{s}(\mathbf{b}') = 0$ and $\underline{s}(\mathbf{b}') = K > 0$. By Lipschitz continuity of $g$, $\overline{s}(\mathbf{b}') - \underline{s}(\mathbf{b}')$ is a continuous function of $\mathbf{b}'$. Hence, there is a value $\epsilon$ such that if $\|w(\mathbf{b}')\| < \epsilon$ then $\overline{s}(\mathbf{b}') - \underline{s}(\mathbf{b}') < 0$. But, the components represented by $\mathbf{b}'$ are fairly bisected, so there is some level below

**q** when the nodes of the tree form an $\epsilon$-fine covering of the subvector **b**. This is a contradiction since at this point a component of **n** would be chosen for bisection. ■

The conditions of this result are not restrictive. Only constraints depending on only one model parameter will generate a sensitivity value of 1. In practice, any component with a sensitivity near 1 is virtually assured of being reduced without bisection. Hence, only considering components with sensitivity values smaller than some threshold near 1 avoids unfair behavior in this case. Interval vectors with components that receive a sensitivity of 0 are independent of that parameter component, and hence the solution set computation does not need to refine that component.

## 5.3 Choosing elements from the active queue

The simplest fair method for choosing elements from the queue is to choose the element with largest width at every step. However, in practice this choice has the undesirable property that a single element or a small group of elements may be chosen several times before their largest component(s) are bisected or reduced. Using volume as an ordering criterion turns out to work better since each iteration reduces the volume of an element by at least a factor of 2. In combination with the bisection method described above, volume ordering leads to a fair algorithm. Hence, Algorithm 4.1 is correct and finitely complete using the $R$ operator of Section 5.1, sensitivity bisection, and volume ordering of $\mathcal{Q}$.

Experimentation has shown that this method is often extremely inefficient. To see why, consider an extreme case where an interval is contained in the solution set. Such an interval is unaffected by $R$, so every time it is chosen, it is bisected, leading to two new intervals that cannot reduce. By comparison an interval vector lying on the boundary of the solution set may be reduced in size by $R$ and then bisected. Both of the children are much smaller in volume than the children of the interval in the solution set. Hence, intervals in the solution set may be processed several times while the intervals on or near the boundary — exactly the intervals that should be processed — remain dormant on the queue. Ideally, such behavior should be avoided.

The ordering algorithm used in practice is based on the observation that constraint testing can be used to classify any interval **p** as *supporting* a decision if $a(\mathbf{p}) < 0$, *not supporting* it if $a(\mathbf{p}) > 0$, or *undecided* if neither

condition holds.[4] Furthermore, in order to reach a **true** or **false** decision, all undecided intervals must be eliminated. If undecided intervals are eliminated, then the solution set must consist of all supporting or all refuting intervals. In cases where the correct decision is "obvious" (the boundaries of the supporting and solution sets are separated by a large distance), all intervals of one "type" (supporting or refuting) can be quickly eliminated while the number of intervals of the other type is unbounded. These observations lead to the following instantiation of `remove`:

1. Classify all intervals in $\mathcal{Q}$ as supporting, refuting, or undecided.
2. If there are intervals labeled undecided choose the largest (by volume measure), otherwise;
3. If there are fewer refuting intervals than supporting, choose the largest refuting interval, otherwise;
4. Choose the largest supporting interval.

In most cases, this algorithm quickly classifies interval vectors as supporting or refuting, and then concentrates on eliminating one set. If, for example, all refuting intervals lie outside the solution set, the number of supporting intervals quickly exceeds the number of refuting intervals, and the algorithm then concentrates on eliminating refuting intervals at the expense of working on supporting intervals.

With this procedure, it is no longer possible to ensure fairness as it was defined in Section 3. However, it is still possible to ensure finite completeness.

**Theorem 5.2.** *Algorithm 4.1 using the* `bisect` *and* `remove` *procedures described above is finitely complete for decidable instantiations.*

*Proof:* Suppose $V(I) =$ **true**. $I$ is decidable, so by Theorem 3.5, there is a $\delta > 0$ such that an interval vector $\mathbf{p}$ cannot be labeled undecided if $\|w(\mathbf{p})\| < \delta$. `Remove` orders undecided elements of $\mathcal{Q}$ by volume and therefore chooses among them fairly. By Lemma 5.1, `bisect` is fair. Hence, every undecided element of $\mathcal{Q}$ will reduced to the point that it must be labeled as supporting or refuting.

[4] In general, an interval is refuting if $a(\mathbf{p}) \geq 0$. However, this definition avoids certain technical complications and special cases, and doesn't change the completeness of the algorithm.

Since $S(g, \mathcal{P}, \mathbf{x}) \subset D(a, \mathcal{P})$, the number of supporting intervals that can appear on $\mathcal{Q}$ is unbounded. Consequently, the number of times a refuting interval is chosen is also unbounded. However, the choice of refuting intervals is fair, and bisection is fair. Hence, for any $\delta > 0$, the covering consisting of all refuting intervals in $\mathcal{Q}$ will eventually become $\delta$-fine. But, there is a $\delta' > 0$ such that for any refuting interval $\mathbf{p}$ with $\|w(\mathbf{p})\| < \delta'$, $R(g, \mathbf{p}, \mathbf{x}) = \emptyset$. Hence, the algorithm will terminate with value **true**.

A similar argument addresses the case where $V(I) =$ **false**.

Finally, suppose $V(I) =$ **undecided**. Since $I$ is decidable, there is a $\delta > 0$ such that any $\delta$-fine partition of $\mathcal{P}$ has an element $\mathbf{p}$ such that $\mathbf{p} \subseteq S(g, \mathcal{P}, \mathbf{x})$ and $\mathbf{p} \cap bdy\big(D(a, \mathcal{P})\big) \neq \emptyset$. Such an element would be labeled undecided, and has the property that every bisection of it or its descendents generates at least one child labeled undecided. All descendents are, of course, in the solution set. By Corollary 3.4 there is a $\delta' > 0$ such that any $\delta'$-fine partition of $\mathbf{p}$ has at least one element that would be labeled supporting and one element that would be labeled refuting by the algorithm. Since bisection is fair and selection within the group of undecided elements is fair, the set of undecided elements in $\mathcal{Q}$ will eventually become $\delta'$-fine. But, by this point one interval vector labeled supporting and one labeled refuting must have been generated. The centers of both elements are within the solution set. Therefore, the algorithm will terminate with value **undecided**. ■

# 6 Experimental results

This section describes the results of applying the algorithm of the previous section to a variety of problems. In order to illustrate the behavior of the algorithm in a controlled fashion, only simulation results are presented. Experimental results on real data can be found in [Hager 1992a, Hager 1992b]. We note that none of the experiments presented in this section were implemented so that $g(p, \mathbf{x}) = \tilde{g}(p, \mathbf{x})$.

**Deciding the size of an ellipse.** The standard equation for an ellipse in the plane can be written as:

$$\begin{aligned} u' &= R(\theta)(u - c) \\ 0 &= \left(\frac{u_1'}{a}\right)^2 + \left(\frac{u_2'}{b}\right)^2 - 1 \end{aligned}$$

where $R$ is a rotation matrix, $u$ is a point on the boundary of the ellipse, and the model parameters are orientation $\theta$, location $c$, and axis lengths $a$ and $b$.

This form of the equation has the undesirable property that rotation and translation are coupled. An alternate form that generally leads to much better performance can be derived by choosing a particular data point and deriving the equation for an ellipse relative to that data point. Letting $u^*$ denote this distinguished point, the equation for an ellipse can be rewritten as:

$$\begin{aligned} u' &= R(\theta)(u - u^*) \\ 0 &= \left(\frac{u'_1}{a} + \cos(\alpha)\right)^2 + \left(\frac{u'_2}{b} + \sin(\alpha)\right)^2 - 1. \end{aligned}$$

Note that this form has one less parameter than the previous expression. It introduces a new parameter $\alpha$, that describes the direction of the distinguished point $u^*$ from the center of the object. This form also decouples rotation from translation (which is no longer explicitly represented). For this reason, this expression is much simpler to solve than that given above. Also note that the data variables do not appear uniquely due to the presence of the rotation matrix $R$.

In order to test the bisection algorithm, 30 data points distributed evenly about the boundary of an ellipse were generated. The interval $[-0.1, 0.1]$ was added to both coordinates to simulate data uncertainty. The initial range on model parameters was

$$\begin{aligned} \alpha &\in [0, \pi/2] \\ \theta &\in [-\pi, \pi] \\ a &\in [10, 50] \\ b &\in [10, 50]. \end{aligned}$$

Performing 100 iterations of bisection for this problem consumed 6.7 CPU seconds on a Sun Sparc 10/41.[5]

The decision-making goal was to determine whether $a > 30$ and $b > 20$. Several experiments were conducted with different values of $a$ and $b$. Other parameter values were held fixed at $\alpha = 0.7$ and $\theta = 0.0$. For each trial, the number of iterations required to reach a decision and the final result were noted. Table 1 summarizes the results.

[5]These timings include some overhead for input and data generation.

| | | b | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 15 | | 20 | | 25 | | 30 | | 35 | | 40 | |
| | 15 | 36 | **F** | 41 | **F** | 34 | **F** | 29 | **F** | 29 | **F** | 29 | **F** |
| | 20 | 48 | **F** | 52 | **F** | 59 | **F** | 62 | **F** | 62 | **F** | 62 | **F** |
| a | 25 | 48 | **F** | 76 | **F** | 109 | **F** | 209 | **F** | 209 | **F** | 209 | **F** |
| | 30 | 52 | **F** | 274 | **U** | 328 | **U** | 113 | **U** | 113 | **U** | 113 | **U** |
| | 35 | 52 | **F** | 109 | **U** | 121 | **T** | 278 | **T** | 278 | **T** | 278 | **T** |
| | 40 | 50 | **F** | 117 | **U** | 92 | **T** | 70 | **T** | 70 | **T** | 70 | **T** |

Table 1

Note that the spatial arrangement of **T**'s, **F**'s, and **U**'s reflects the structure of the supporting set for this problem. As expected, the amount of computation varies greatly. For **T** and **F** decisions it is roughly inversely proportional to the distance between the supporting set and the solution set. **U** decisions clearly require more effort. This is at least partially due to the fact that the interval selection and bisection methods have not been optimized to find points within the solution set.

| | | b | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 15 | | 20 | | 25 | | 30 | | 35 | | 40 | |
| | 15 | 273 | **F** | 189 | **F** | 143 | **F** | 143 | **F** | 143 | **F** | 143 | **F** |
| | 20 | | * | 469 | **U** | 1646 | **U** | 217 | **U** | 217 | **U** | 217 | **U** |
| a | 25 | 746 | **U** | 292 | **U** | 365 | **U** | 1181 | **U** | 1181 | **U** | 1181 | **U** |
| | 30 | 476 | **U** | 157 | **U** | 145 | **U** | 93 | **U** | 93 | **U** | 93 | **U** |
| | 35 | 179 | **U** | 132 | **U** | 552 | **U** | 4024 | **T** | 4024 | **T** | 4024 | **T** |
| | 40 | 56 | **U** | 122 | **U** | 1588 | **T** | 106 | **T** | 105 | **T** | 52 | **T** |

Table 2

In order to illustrate the effects of missing data and larger data errors, the same trials were performed with an error bound of $[-0.5, 0.5]$ and only a 220 degree arc of data. Table 2 summarizes the results. The effects of the additional model uncertainty can be clearly seen by looking at the increase in the number of cases marked **U**. In addition, in nearly all cases the amount of computation required to solve the problem increased significantly. In one trial, marked by *, no decision was reached in 10000 iterations.

In practice, the increase in running time can be handled in many ways. In most applications, the algorithm is run with a time bound derived from application-specific time constraints. If that bound is exceeded, it can be presumed that the problem is in some respect ambiguous. The problem is assigned the value **U** and the action appropriate for undecidable situations is taken. In particular, additional sensor readings can be added to the data vector. Additional data often make up for "gaps" in the initial data set, and also add redundant information which, when data is noisy, serves to reduce the size of the solution set. [Atiya & Hager 1991, Hager et al. 1993] discuss this issue in more detail.

**Comparing gears.** For a somewhat more complex and realistic example, consider the problem of isolating the largest of a collection of gears based on outer diameter. Since gears are surfaces of revolution, it suffices to consider a planar model. Unless otherwise noted, all distances in this section will be in millimeters.

The contours of the gear in nominal position can be defined as two explicit functions of angle:

$$\begin{aligned} v_1(\theta) &= \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} \Big( d + k \sin(K\theta + \theta_0) \Big) \\ v_2(\theta) &= \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} h. \end{aligned}$$

Here, $d$ is the average outer gear diameter, $h$ is the hub diameter, $k$ is the depth of the gear cut, $K$ is the number of teeth in the gear, and $\theta_0$ is the phase of the gear cut. Note that $K$ is an integer value. Figure 1 shows the data generated by sampling these contours.

With some manipulation, it is possible to derive an implicit form of these equations:

$$(c_1 - u_1)^2 + (c_2 - u_2)^2 - \left[ d + k \sin\left( \theta_0 + K \operatorname{atan}\left( \frac{u_2 - c_2}{u_1 - c_1} \right) \right) \right]^2 = 0 \quad (5)$$

$$(c_1 - u_1)^2 + (c_2 - u_2)^2 - h^2 = 0. \quad (6)$$

As before, $c$ is a vector representing the location of the center of the gear. Once again, this form of the equations is not well-suited for interval solution. Assuming a point $u^*$ on the inner hub of the gear can be reliably isolated,

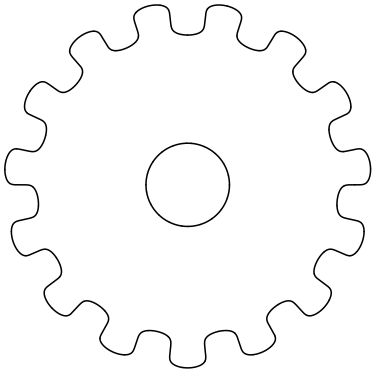

Figure 1: The contours generated by the gear model.

(6) can be rewritten as

$$\left(\frac{(u_1 - u_1^*)}{h} + \cos(\theta)\right)^2 + \left(\frac{(u_2 - u_2^*)}{h} + \sin(\theta)\right)^2 - 1 = 0 \qquad (7)$$

where $\theta$ is the direction of $u^*$ from the center. Given an arbitrary data point $u$, it is not generally known if $u$ comes from the hub contour, in which case (7) applies, or if it comes from the outer contour, in which case (5) applies. This ambiguity is addressed by multiplying the left hand sides of the two expressions and setting the result equal to 0. The multiplication forces only one of the two constraints to be satisfied for any data point. Finally, the location of the center of the gear can be computed from $u^*$, $h$, and $\theta$ as:

$$\begin{aligned} c_1 &= u_1^* - h\cos(\theta) \\ c_2 &= u_2^* - h\sin(\theta) \end{aligned}$$

and this value of $c$ can be substituted into (5) to improve performance. Also, the value of $K$ is known to be an integer. This information is used in computations by noting that any interval value of $K$ with nonintegral endpoints can be reduced to an interval with integral endpoints.

When processing several gears, the full system parameter vector contains a separate subvector of parameters for each gear. In order to be chosen as

the largest, gear $i$ must satisfy the criterion:

$$\bigwedge_{j \neq i} [d_i > d_j] .$$

This criterion must be replicated and evaluated in parallel for each gear. While not discussed in this paper, the actual algorithm employed in these tests has been extended to handle problems where multiple decisions must be considered simultaneously. See [Hager 1992b] for details. These problem attributes — multiple contours, rotational components, translation, integer-valued variables, and multiple decisions — are all commonly encountered in practice.

As before, artificial gear data was generated with an uncertainty of $[-0.1, 0.1]$ in both coordinates. 20 points were generated from the hub, and 100 points were generated from the outer gear contour. The initial range on parameters was

$$\begin{aligned}
\theta &\in [-\pi, \pi] \\
\theta_0 &\in [-\pi/10, \pi/10] \\
c_1 &\in [-10000, 10000] \\
c_2 &\in [-10000, 10000] \\
h &\in [1, 10] \\
k &\in [1, 10] \\
d &\in [20, 100] \\
K &\in [20, 50].
\end{aligned}$$

Performing 100 iterations of bisection for this problem consumed 84.4 CPU seconds on a Sun Sparc 10/41.[6]

The algorithm was first applied to the problem of determining the largest of two gears. One gear was held to a fixed diameter of 40 mm while the size of the other was varied in millimeter increments from 40 to 48. Other parameters were held fixed at $c_1 = c_2 = 0$, $h = 8$, $\theta = \theta_0 = 0$, $k = 5$, and $K = 25$. In order to demonstrate the effects of data uncertainty, the number of iterations until a decision was made was recorded for four different uncertainty levels. Decisions requiring more than 1000 iterations were assigned the value **U**. All cases that terminated within 1000 iterations received the value **T**. The results appear in Table 3.

[6]These timings include some overhead for input and data generation.

| | | Gear Size | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 |
| Uncertainty Level | 0.05 | $> 1000$ | 86 | 50 | 58 | 34 | 34 | 33 | 29 | 26 |
| | 0.1 | $> 1000$ | 208 | 70 | 50 | 50 | 42 | 38 | 30 | 26 |
| | 0.2 | $> 1000$ | $> 1000$ | 475 | 210 | 190 | 121 | 91 | 66 | 48 |
| | 0.3 | $> 1000$ | $> 1000$ | 666 | 522 | 344 | 146 | 124 | 80 | 75 |

Table 3: The number of iterations to a decision when comparing two gears of similar size.

As would be expected, the larger the data uncertainty, the more difficult the comparison task becomes, as measured by the number of iterations of the algorithm. However, it is interesting to note that the amount of work needed to make more obvious decisions does not depend heavily on the level of data uncertainty.

When given several gears to sort, the algorithm need only process a specific gear until it is guaranteed that some other gear is larger. The algorithm was given gears with common parameters $c_1 = c_2 = 0$, $h = 8$, $\theta = \theta_0 = 0$, $k = 5$, $K = 25$, and values of $d$ of 20, 25, 30, 32, 34, 40, 50, 50, 60, and 70. Note that the ambiguity between the two gears in the middle of the spectrum should not lead to an undecidable outcome since both are smaller than the largest gear. Table 4 summarizes the number of iterations devoted to each size gear before it was determined not to be the largest.

| | | Gear Size | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 20 | 25 | 30 | 32 | 34 | 40 | 50 | 50 | 60 |
| Uncertainty Level | 0.05 | 12 | 12 | 12 | 12 | 13 | 14 | 16 | 16 | 20 |
| | 0.1 | 12 | 12 | 12 | 12 | 15 | 15 | 16 | 16 | 26 |
| | 0.2 | 12 | 12 | 12 | 12 | 15 | 15 | 22 | 22 | 38 |
| | 0.3 | 16 | 16 | 23 | 23 | 23 | 26 | 46 | 46 | 58 |

Table 4: The number of iterations to a decision when comparing several gears at once. The largest gear has size 70.

The important point of this table is to again note that the effect of increasing data uncertainty has little effect on algorithm performance except in cases where reaching a decision depends on making fine distinctions among data models.

# 7 Conclusions

This paper has presented a simple-to-use algorithm for making decisions about sensor data. The major features of this algorithm are its simplicity, its flexibility, and the fact that for most problems, it will terminate in finite time with a correct answer (modulo assumptions about the range of errors in the observed data). Although not described in this paper, the algorithm has been parallelized using the Linda [Scientific Computing Associates 1990] distributed programming system. Near linear speedup in constraint-solving performance has been observed for problems of the appropriate granularity.

From a practical perspective, the major hurdle when implementing problems has been ensuring that the data conform to the bounded error model, and setting up the system of constraints so that they are efficiently solvable. Techniques for filtering data to remove data outliers have been developed, and some protection against data outliers has been incorporated into the algorithms themselves. With these additions, the current implementation has proven to be robust to such problems on real data.

The major drawback of the algorithm is the number of iterations it takes to reach a decision in ambiguous situations. However, it can be argued that this behavior is preferable to returning an incorrect answer within a specified time bound. Moreover, in many applications it is reasonable to specify a time bound on the algorithm. If no decision can be made within that time bound, the problem can be declared ambiguous, and the appropriate action taken as if the algorithm had returned **U**. In particular, in sensing applications the obvious solution is to seek out more data which further constrains the data model and permits a decision to be reached.

Work is proceeding on several improvements to the algorithms. Most importantly, a data selection method has been developed so that it is possible to choose a small subset of the available constraints to be used in the $R$ operator. For large data sets, selection can be performed much more quickly than performing constraint solving on the entire data set. Consequently, when the speed of convergence can be maintained, adding a data selection step can result in substantial time savings. Work is also being done to improve the performance of the $R$ operator, as well as the interval selection and bisection choices.

# Acknowledgements

The author would like to thank Stan Eisenstat and the anonymous reviewers for many helpful comments and suggestions, Dana Angluin and Willard Miranker for reading earlier drafts of this paper, and Hemant Tagare for several useful conversations. This research was supported by DARPA grant N00014-91-J-1577, by National Science Foundation grants IRI-9109116 and DDM-9112458, and by funds provided by Yale University.

# References

[1] Alefeld, G. and Herzberger, J. *Introduction to interval computations.* Academic Press, New York, 1983.

[2] Atiya, S. and Hager, G. D. *Real-time vision-based robot localization.* In: "Proceedings of the 1991 IEEE International Conference on Robotics and Automation", IEEE Computer Society Press, 1991, pp. 639–643.

[3] Hager, G. D. and Lu, J. *C++ interval and constraint solving package.* DCS RR-953, Yale University, New Haven, CT, 1993.

[4] Hager, G. D. (1992a) *Constraint solving methods and sensor-based decision making.* In: "Proceedings of the IEEE International Conference on Robotics and Automation", IEEE Computer Society Press, 1992, pp. 1662–1667.

[5] Hager, G. D. (1992b) *Task-directed computation of qualitative decisions from sensor data.* DCS RR-921, Yale University, New Haven, CT, 1992. Submitted for review to the IEEE Transactions on Robotics and Automation.

[6] Hager, G. D. (1992c) *Task-directed computation of qualitative decisions from sensor data.* Submitted for review to the IEEE Transactions on Robotics and Automation, 1992.

[7] Hager, G. D., Engelson, S., and Atiya, S. *On comparing statistical and set-based methods in sensor data fusion.* In: "Proceedings of the IEEE International Conference on Robotics and Automation", IEEE Computer Society Press. To appear in 1993.

[8] Neumaier, A. *Interval methods for systems of equations.* Cambridge Univ. Press, 1990.

[9] *C-Linda reference manual.* Scientific Computing Associates, 246 Church St., Suite 307, New Haven, CT, 1990.

[10] Solina, F. and Bajcsy, R. *Recovery of parametric models from range images: the case for superquadrics with global deformations.* IEEE Transactions on Pattern Analysis and Machine Intelligence **12** (2), pp. 131–147.

Department of Computer Science
Yale University
P.O. Box 2158 Yale Station
New Haven, CT 06520
USA
E-mail: `hager@cs.yale.edu`