DEPARTMENT OF ARTIFICIAL INTELLIGENCE
UNIVERSITY OF EDINBURGH

DAI Working Paper No. 205
Date: August 1987

Solving Algebraic Constraints In A Parallel Network, As Applied To Geometric Reasoning

Robert Fisher

Abstract:

This paper describes a network implementation of the SUP-INF method for solving sets of inequalities that has advantages over previous implementations. As algebraic inequalities are often a preferential representation for expressing relationships, efficient solution of sets of algebraic constraints is desirable. The cost of symbolic manipulation is transferred to compile-time and speed up at run-time is further enhanced by parallel evaluation. Further, allowing iteration in the network improves the competence of the method when working with non-linear expressions.

The network method is applied to implement a geometric reasoner for a computer vision program and is shown to meet the general requirements for such a geometric reasoner.

Analysis of typical geometric relationships demonstrates repeated algebraic substructure. This allowed the creation of standard network modules computing the algebraic relationships in the substructures. These modules are connected together to solve larger algebraic problems.

Acknowledgements

Funding for this paper was under Alvey grant GR/D/1740.3. The ideas in this paper benefited greatly from discussions with J. Aylett and M. Orr.

1 Introduction

Geometric reasoning is a necessary component of any competent vision system. It is needed for deriving and integrating position information (from data elements paired with model features) to deduce object positions. Other functions are the prediction of image locations (or other properties) and geometrically testing of proposed model-to-data pairings. This paper describes how these functions can be computed in a parallel network based on an algebraic representation of the geometric constraints between features. The theory behind the network and some programming details about their use and construction are described, and are illustrated in an example.

This network approach is interesting for several reasons:

- it is suitable for massive parallelism, with simple, well-defined processing elements and orderly connections,
- it is an interval constraining algorithm that operates over non-linear algebraic inequalities,
- it incorporates constructs to implement parallel case statements,
- it includes a signed reciprocal function allowing isolation of terms of unknown parity in algebraic inequalities and
- the network can be modularized for incremental construction.

Previously [Orr and Fisher 1987], we classified the tasks a vision-oriented geometric reasoner required, and found that only five fundamental geometric operations lay underneath these tasks, but there were a variety of approaches to implementing the functions. The operations are:

<u>LOCATE</u> - deducing position constraints from model-to-data feature pairings,
<u>MERGE</u> - integrating separate position constraints,
<u>TRANSFORM</u> - mapping a position into a new position-based reference frame,
<u>INVERT</u> - reversing the position mapping and
<u>PREDICT</u> - deducing data from model features and positions.

The formal relationships between position representations and these functions was precisely defined in an abstract data type definition.

Our recent model-based scene understanding (e.g. [Fisher 1986]) has concentrated on recognising and locating objects using 3D data features paired with 3D model features (such as edges, surface patches and volumes). (The SMS representation system [Fisher 1987a] has been used for object modeling.) This has led us to consider ([Fisher and Orr 1987], [Orr 1987]) what geometric position constraints are derivable from pairing the 3D model features with the 3D data features. This is complicated, because of (1) the variety of data and model features, (2) the variety of constraints types obtainable from pairing such features and (3) the existence of constraints that are only partial.

If the two main geometric entities were points and vectors, then our analysis showed that only two primitive position constraints were needed for representing all information obtainable from pairing a model geometric entity to a data geometric entity. The primitive constraints used (1) the angular distance between a transformed model vector and the corresponding observed data vector (possibly including parameterized vector positions) and (2) the spatial distance between a transformed model point and the corresponding data point (again with parameterizations).

More complex constraints could be deduced by combining typical groupings of these primitive constraints. The work also catalogued the set of geometric constraints resulting from the pairing of the likely model and data features, given the SMS models and 2 1/2D sketch data.

Another conclusion from this work was that algebraic inequalities over model, data and positional variables were a good representation for expressing the geometric constraints (following ACRONYM [Brooks 1981]). The key justifications for this were:

- The representation is incremental in two senses. First, new constraints can be added when each new piece of evidence is discovered. Second, new classes of constraints can be added when new visual relationships are understood.
- The representation is uniform, because it can represent a large range of visual relationships using the same mechanism.
- A priori knowledge of scene relationships (e.g. "the object must lie on the conveyor belt") is also expressible in algebraic form, allowing direct integration with observed and model relations.

On the other hand, methods are then needed for integrating the constraints, to provide estimates of the constrained variables (such as for object position).

This work has led us to now investigate mechanisms for implementing the geometric reasoning functions needed for vision, recognizing some of the difficulties as:

how to express and integrate partial constraints,
represent and compute with data errors,
perform the geometric computations quickly.

The body of this paper describes a parallel network geometric reasoning engine used for performing the computations. The algebraic constraints representing geometric relationships are compiled into a set of bounds on the position variables as a function of other position variables.

These bounds define a network of nodes representing expressions linked by their algebraic relationships that solves geometric problems as it relaxes to a state consistent with the input data, model properties and geometric relationships. The network can be pre-compiled before any observations, with sections activated only when appropriate evidence is obtained.

One advantage of the network formulation is it suggests direct parallel MIMD implementations of the computation, potentially leading to many orders of magnitude improvement in performance. A "competence" advantage of using the network formulation is that it allows iterative solution of non-linear geometric problems, which can lead to better solutions than current algebraic methods.

The structure of the network is constructed from the geometric constraints defining the problem. ACRONYM's [Brooks 1981] sup/inf constraint manipulation method (with some extensions [Fisher 1987b]) produces bounds on individual variables and reduces them to simpler form. Then, a network module is compiled from the simplified algebraic structure represented in the constraints. These modules are used as black boxes, representing a particular form of constraint, whose interfaces are connectable to the appropriate data during use. In particular, if model and data feature positions are input, then the output might be the transformation between the two. Alternatively, given the model and transformation, predicted data feature positions would be the output.

A key property of the network module is that a new instance can be allocated and linked with existing modules as needed. For example, when a new model-data feature pairing is discovered, then the appropriate network module can be created and linked to integrate the model position constraints of the new relationship with the existing positions. It might also signal that the proposed pairing was inconsistent with the previous position estimates.

This network module approach means that the slow symbolic manipulation is performed at network definition time, and allows fast and flexible execution during recognition.

This paper describes the individual components of the network, how they are structured into modules, how the network is evaluated in parallel or serial, how the network modules are created and connected, and how the network implements the five primary geometric reasoning functions described above. An example network is also demonstrated.

In the discussion below, the term **symbolic CMS** means our adaptation and improvement of ACRONYM. The term **network CMS** means the compiled network, which implements the same function as the symbolic CMS in a different manner.

2 Related Work

The use of the algebraic inequalities to represent geometric constraints derives from Brooks' ACRONYM [Brooks 1981] work, as does the symbolic constraint manipulation methods. The network computation is similar to the many relaxation (e.g. [Rosenfeld 1978]) or constraint satisfaction (e.g. [Freuder and Quinn 1985]) algorithms that are suitable for parallel processing. However, it differs from the relaxation algorithms in that is not a probabilistic labeling computation and from constraint satisfaction in that an infinite continuous range of values is being reduced rather than selection from a finite set of discrete values. While the networks described below rely on connections between units, the computation is not in the distributed connectionist form, where the results are expressed as states of the network. Instead, the results are the values current at selected processors.

Davis [Davis 1987] has classified the types of constraint propagation systems. The engine described below is an interval label constraint machine applied over full algebraic constraints (with some transcendental operations). It is used for geometric reasoning without dependence on handling sin and cos (though it does this somewhat) by using a quaternion rotation representation. His complexity analysis indicates execution times may be doubly exponential and termination may not even occur (unless forced by truncating small changes, as is done here).

Here, the complexity does not appear to be a problem, and computation appears to be O(network size) with a small constant (e.g. 5). This may result from the truncated convergence of values. Davis also raises the problem of disjoint parameter intervals. We believe the geometry understanding embedded in the scene analysis program will detect most cases of this in advance (e.g. will know about n-fold symmetry) and create separate hypotheses with only single intervals.

The work presented here differs significantly from two other network based geometric reasoning systems. Hinton and Lang [Hinton and Lang 1985] learned and deduced positions of translated 2D patterns (alphabetic letters) using a distributed connectionist network. Its intermediate nodes represented object position and gated connections between iconic image and model representations. This model required specific computing units for each translation. Strengths of connections were learned through training. The network could recognize the patterns in the presence of disrupting noise, but was limited to a few translated iconic 2D patterns. It was probably also necessary to train the recognizer with the models in all positions.

Ballard and Tanaka [Ballard and Tanaka 1985] demonstrated a 3D reasoning network whose nodes represent instances of parameter values and whose connections represent consistency according to model-determined algebraic relationships. Thus, if the relationship a*b=c is given, then there would be a connection between nodes for specific values of a_i, b_j and c_k if they have the relationship. The algebraic relationships were based on stereo pairing of straight image edges to give a 3D wire frame and then pairing these to model edges to estimate a reference frame. The network deduced the object's reference frame through integration of evidence, much like their previous Hough transform work.

In both cases, patterns of network activity result, with the dominant pattern accepted as the answer (unlike here, where the result is explicit). Both systems also simultaneously select a model, which is treated separately in our analysis. Since the networks are omnidirectional, such an approach is also suitable for consistency analysis and feature prediction.

Of the two, I feel that the algebraic approach is more profitable, as well as providing 3D analysis capability. The approach presented below follows this somewhat, differing in using bounded variable units instead of one unit for each numerical value. Further, the algebraic structures used are generic, and are usable to express many geometric relationships.

3 Network Elements

The evaluable portion of the network consists of connected modules containing two types of nodes (value and operation) and the module interfaces. Each network module implements the algebraic reasoning implicit in the module's associated constraints.

The value nodes are associated with the variables in the network. They acquire inf and sup bounds on their associated algebraic quantity. The bounds are computed from the relationships with other value nodes, as defined by the operation nodes.

Operation nodes implement a simple unary or binary function and take their inputs from value nodes and other operation nodes. The operators implemented are: {"+", "*", "/", "sup_of_max", "sup_of_min", "inf_of_max", "inf_of_min", "extract_sup", "extract_inf", "constant", "cos", "sin", "sqrt", "<", "≤", ">", "≥", "and", "or", "not", "select" and "enable"}.

A parallel network based case construction is needed, because some operations produce different outputs according to conditions on their inputs. For example, if bounding "1/X", then we want different bounds depending on whether 0 is in the allowed range for X or not.

Two special types of operation nodes were added to implement the selection function. The first is an 'enable' operation, a function of a test argument and a result argument, whose output is the result argument only if its test argument is true. The second type is the 'select' operation, that returns the first of its arguments to become defined. Using these, the 'enable' operation turns on and off results, according to their applicability (as determined by the test arguments), and the 'select' operation passes through the 'true' value. (There should not be more than one 'enable'd in a properly designed module.) The logical value of the test argument is generated by using a numerical comparison operator (e.g. "<") or a logical operator (e.g. "and").

Modules are used to package together a set of related algebraic constraints. Each module consists of a local network computing its constraints. The module may be connected to other modules to form a larger network, by linking variables. Suppose the module is thought of as an electrical black box over a set of variables, which are identified with the connectors into and out of the black box. Then, integrating several constraints on several external variables (i.e. the computation's inputs and outputs) is equivalent to joining the appropriate connectors of one or more black boxes to the external variables as well as to each other.

The interface nodes are used for linking together the value nodes internal to separate modules with external variables (such as for observed data values and predicted object positions). This allows the sharing of constraints between multiple modules in one network.

The network is executed by computing the values of the operation nodes that bound the value nodes, which in turn may give new values to the operation nodes. This cycle of operation continues until no more nodes need updating, which must occur because the bounds on the value nodes converge monotonically and asymptotically and minor updates are truncated.

After convergence, bounds on the desired values are extracted from the associated value nodes.

4 Network Creation

This section summarizes how individual network modules are created from a set of constraints and how they are used in practice.

4.1 Algebraic Position Constraints and ACRONYM

The key geometric concept is the (relative) position, which is represented using (a) a variable for each degree of freedom and (b) a set of algebraic relationships (over the variables) between this and other positions. These constraints on positions may be formed by relating expressions in the variables to quantities measured from the image. Alternatively, they may represent model relationships (such as reference frame transformations).

Since the geometric constraints defining object position are defined algebraically, following ACRONYM [Brooks 1981], one might use ACRONYM's symbolic constraint manipulation system (CMS). This symbolic CMS could bound expressions over non-linear inequalities and achieved respectable performance through a combination of case analysis and considerable symbolic algebra.

The advantage of using symbolic algebra instead of merely bounding each term is seen in the following example. Suppose we wish to bound the expression "A*B", where A and B are functions of X, which must lie in the range [1,2]. Suppose A = X and B = 2 / X. Bounding A*B by bounding and multiplying the bounds on the individual terms gives the range [1,4]. However, using symbolic algebra to reduce A*B first gives the tighter bound [2,2].

4.2 Preprocessing the Constraints

Constraints on variables are represented as algebraic inequalities bounding isolated variables, such as:

$x \leq y + z$

Achieving this form is not always trivial, and requires several techniques that are applied by hand before processing. This is not a crucial point for the geometric reasoning described here as the constraint modules (section 7) are defined once-for-all.

If an equality is encountered, then it is split into two inequalities:

$x = Expr$

becomes

$x \leq Expr$ and $x \geq Expr$

If a product term is encountered during pre-processing, it is split into several cases using the signed reciprocal function:

$A*B \leq C$

becomes

$A \leq C * \mathrm{srecip}(B)$
$B \leq C * \mathrm{srecip}(A)$
$A \geq -C * \mathrm{srecip}(-B)$
$B \geq -C * \mathrm{srecip}(-A)$

This function has the definition:

```
if x > 0 then srecip(x) = 1/x
          else srecip(x) is undefined
```

This function then has the effect of turning on and off constraints according to the sign of the terms being divided. Then, we need not worry about whether the inequality should be reversed.

Some constraints reduce to the form:

$$x \leq f(x)$$

such as when

$$x*x \geq 1 - y*y$$

is reduced to

$$x \geq (1 - y*y)*\text{srecip}(x)$$
$$x \leq -(1 - y*y)*\text{srecip}(-x)$$

The bound simplification (section 4.3) has difficulties with these constraints and usually ends up reducing them to the weak form:

$$-\infty \leq x \leq +\infty$$

To prevent this, recursive constraints (i.e. those bounded by a function of themselves) are distinguished from the non-recursive bounds.

Geometric reasoning typically requires reasoning about angular quantities, but the original CMS contained little to support this. Though we are considering how to remedy this, the current rotation representation uses a quaternion formulation, which reduces all computations to vector form.

4.3 Simplification of Bounds

Assume we have a set of algebraic constraints over a set of variables. We could use these to directly create a network, as described below. However, it is desirable to make some optimizations, such as removing bounds that are uniformly weaker than others. Further, some expressions might be reduced to simpler expressions, such as by multiplying out constants or eliminating dependent variables. Merely applying the expression simplifier provides some reductions.

In general, this is a hard problem, but ACRONYM's [Brooks 1981] constraint manipulation system at least provides some simplifications (particularly over linear terms). We have also made some extensions [Fisher 1987b], notably for: square roots, powers of variables, products over expressions containing only free variables, the unsigned reciprocal function and undefined values.

The symbolic CMS is then used to derive symbolic bounds on non-observable values (e.g. object orientation) as a function of observable values (e.g. direction of a surface normal), which would initially be expressed as variables. Because the network modules are used to express the full relationship between quantities, the reverse mappings are also represented.

The CMS bounding is applied to all non-recursive and non-guarded constraints for each variable, over all other variables referenced in the constraint module. The constraints are a function of other variables, including those with unconstrained values (e.g. image observables). As we will want to add bounds to these variables later, (when a data observation is made), they should be left in the resulting constraint and hence not much bounding is possible. (If they were left out, the resulting constraint would be both weak and independent of any input data.) Some bounding is possible if uniformly stronger constraints are found, but this is quite difficult to do with non-linear inequalities.

The recursive and guarded constraints are then added to those that result from the application of the CMS to provide the inputs for network compilation (below).

The symbolic CMS could be used directly by the geometric reasoning program, instead of using the parallel network. This is inappropriate for two reasons:

(1) It is computationally expensive and is not suited for wide scale parallelism.

(2) Using the network provides better bounds than the CMS. This is because the CMS works best with linear constraints, whereas geometric reasoning also includes non-linear constraints. Here, the network can iterate to a better bound than that achieved by one pass through the CMS. An example of where the network performs better is finding sup(x) over:

$$
\begin{aligned}
x &\leq 1 + 1/y \\
y &\geq 1 + 1/x \\
0.1 &\leq x \leq 10 \\
0.1 &\leq y \leq 10
\end{aligned}
$$

The CMS (simplified somewhat) finds:

$$\sup(x) = 1 + 1/\inf(y) = 1 + 1/(1 + 1/\sup(x))$$

When it gets to the embedded sup(x), it uses the numerical upper bound 10 to produce:

$$\sup(x) = 1 + 1/(1 + 1/10) = 1.91$$

However, the network computation iterates to converge to the best bound:

$$\sup(x) = (1 + \mathrm{sqrt}(5)) / 2 = 1.62$$

4.4 Network Compilation

Value nodes are created for every individual variable occurring in the original constraints (which are given in the form "variable ≤ expression"). These nodes are then connected by various operator nodes, which extract values from Value nodes or from each other. The real details of network structure concern the connections used for the operator nodes. These connections are structured according to the types of expressions found in the constraints, and will be shown in that form below.

The network fragments for the following expressions are described:

Constant - an operation node is created that supplied the given constant

Variable - an operation node is created that extracts the sup or inf from the associated Value node.

Variable^N, where N is odd - a sequence of "times" operation nodes are created and linked to the sup (or inf) of the variable. The output of each "times" operation becomes an input to the next.

Variable^N, where N is even - If sup is the desired bound, then the network fragment calculates the Nth power of both sup(Variable) and inf(Variable) (using the same sequential technique as for the odd case above). Then, a final "max" function selects between the two possibilities.

If inf is the desired bound, then the construction is:

Condition	Compiled result
sup(Variable) < 0	sup(Variable)^N
inf(Variable) > 0	inf(Variable)^N
sup(Variable) ≥ 0 and inf(Variable) ≤ 0	0

plus - an operation node is created that adds the results from the recursively compiled subexpression nodes.

recip (i.e. 1/x) - if sup is the desired bound, then the construction is:

Condition	Compiled result
inf(Expression) > 0	1 / inf(Expression)
sup(Expression) < 0	1 / inf(Expression)
else	p_infinity

where "bound(Expression)" links to the recursively compiled subexpression.

If inf is the desired bound, then the construction is:

Condition	Compiled result
inf(Expression) > 0	1 / sup(Expression)
sup(Expression) < 0	1 / sup(Expression)
else	n_infinity

srecip (i.e. signed reciprocal defined by: if x>0 then srecip(x) = 1/x, else srecip(x) is undefined.) - if sup is the desired bound, then the construction is:

Condition	Compiled result
inf(Expression) > 0	1 / inf(Expression)
else	undefined

where "bound(Expression)" links to the recursively compiled subexpression.

If inf is the desired bound, then the construction is:

Condition	Compiled result
inf(Expression) > 0	1 / sup(Expression)
else	undefined

times - if sup(A*B) is desired, a network fragment for:

```
max(inf(A)*inf(B), inf(A)*sup(B), sup(A)*inf(B),
             sup(A)*sup(B))
```

is created, where "times" nodes calculate the products of the values calculated in the recursively defined subfragments and a max node selects the largest of the products. Special cases are used when A or B is a constant.

If inf(A*B) is desired, a network fragment for:

```
min(inf(A)*inf(B), inf(A)*sup(B), sup(A)*inf(B),
             sup(A)*sup(B))
```

is instead created. Again, special cases are used when A or B is a constant.

square root - here the positive square root is assumed. If sup is the desired bound, then the construction is:

Condition	Compiled result
sup(Expression) ≥ 0	sqrt(sup(Expression))
else	p_infinity

where "bound(Expression)" links to the recursively compiled subexpression.

If inf is the desired bound, then the construction is:

Condition	Compiled result
inf(Expression) ≥ 0	sqrt(inf(Expression))
else	0

max (or min) - sup(max(List)) is compiled to be max(sup(List)). This creates subfragments for each expression in the list and then links these to a chain of connected binary 'max' operation nodes. The compilation removes any duplicated entries from the lists. Different types of min and max operation nodes are used depending on whether sup or inf is selected. The difference is reflected in the default evaluation actions, as described in section 5.

guard - compiles an 'enable' node implementing:

Condition		Result
Expression1		sup(Expression2)
	or	
Expression1		inf(Expression2)

and (or 'or') - compiles a logical 'and' (or 'or') of the subexpressions, without regard for sup or inf.

greater (or greatereq) - compiles a node checking if inf(subexpression1) > (or ≥) sup(subexpression2)

less (or lesseq) - compiles a node checking if sup(subexpression1) < (or ≤) inf(subexpression2)

not - compiles a node evaluating not(subexpression)

zero - compiles a node checking if inf(subexpression) ≤ 0 and sup(subexpression) ≥ 0

notzero - compiles a node checking if inf(subexpression) > 0 or sup(subexpression) < 0

Since subexpressions often reappear in constraints on different variables in the same constraint module, it is sensible to not duplicate the network fragments for the common subexpressions. Hence, the recursive compilation of expressions always returns a previous compilation for an expression if it exists. Attention is paid to whether the sup or inf of the expression is desired, as (e.g.) the network fragments would not be the same for the sup and inf bounds of "A+B".

To reduce the duplication involved in multiple bounds on a variable, the network structure compiled as described above is simplified somewhat afterwards. The simplification reduces sets of upper bounds on V of the form: {V < a, V < b, V < c, ...} to "V < min(a, b, c, ...)". A similar simplification applies to the lower bounds using the max function.

Propagation delays are reduced by using balanced 'min' and 'max' operation subtrees (which does not increase the module size).

Ordinarily, the network does not contain any value nodes for the many intermediate expressions occurring in the constraints. However, they can be included for debugging. This creates more value and operation nodes, and also increases run-time, but provides a better impression of what's being calculated.

4.5 Network Module Formation

The function of a module is to group together all constraints of a particular type into a prototypical network fragment. Then, when a problem is being analyzed, modules of the appropriate types can be copied and linked to implement the deduced constraints. These modules are compiled individually, in advance, and then loaded when processing is started. The module interfaces supply all the information needed for connecting up modules to each other and external variables.

We illustrate the network creation process by showing the structure for a single module for the constraint "A ≤ B - C" over the given variables (whether the variables represent model, position or observable values is not important). Two other equivalent relations are "C ≤ B -

A" and "B ≥ A + C" (e.g. we bound all variables by expressions in other variables). The extended symbolic CMS [Fisher 1987b] is then used to calculate supremum and infimum expressions over these variables, giving:

$$\sup(A) \leq \sup(B - C) \leq \sup(B) - \inf(C)$$
$$\sup(C) \leq \sup(B - A) \leq \sup(B) - \inf(A)$$
$$\inf(B) \geq \inf(A + C) \geq \inf(A) + \inf(C)$$

This creates value nodes for: {A, B, C} and also for {B-C, B-A, A+C} if debugging information is desired.

Moreover, the bounding expression for "sup(B - A)" is a function of two other bounds, which introduces the operation nodes into the network. This compilation results in the network in figures 1.

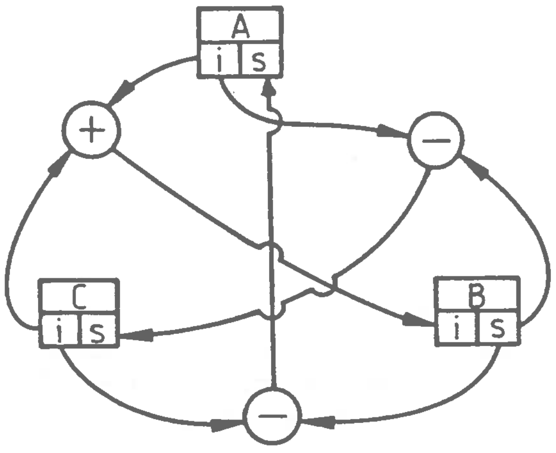

Figure 1: Network Module for "A ≤ B - C"

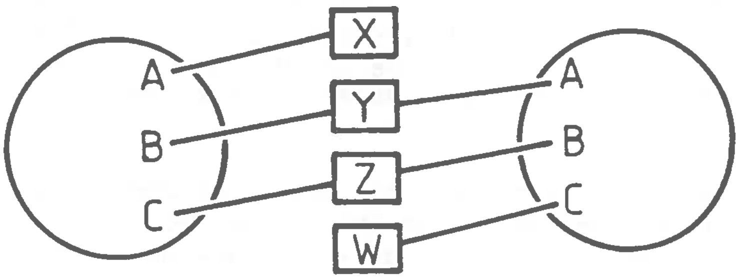

Figure 2: Connected Network Modules for Combining Constraints

Now, suppose we had two instances of this form of constraint

$$x \leq y - z$$
$$y \leq z - w$$

over the observed variables $x = x_o$, $y = y_o$ and $w = w_o$. We could express these two constraints in the network form using two instances of the module M defined in figure 1, connected as in figure 2. For the first constraint, the pairings x->A, y->B and z->C are made as shown. The second module represents a second constraint between the variables, and is connected as shown.

The network modules will usually implement much more complicated constraints (section 6).

5 Network Evaluation

The values at each node are computed using the values of the connecting nodes. For example, the computation for the supremum (e.g. sup(A)) picks the minimum of each bounding expression on the supremum, including the current supremum, because the bounds can never get worse (assuming a bound is always valid). Hence, if:

$$\sup(A) \leq X$$

$$\sup(A) \leq Y$$

then:

$$\sup(A_{t+1}) \leftarrow \min(\sup(A_t), X_t, Y_t)$$

is the updating function for the supremum of A. This computation is implemented by a operation node taking the "min" of X and Y, which then may update sup(A), if appropriate. A similar computation updates the inf of values.

The value at an operation node is recomputed whenever one or more of its inputs changes. As a result, changes to value and operator nodes propagate throughout the network until asymptotic convergence occurs.

Ordinarily, an operator will not be evaluated until all arguments have values. This causes problems when using the functions that may not evaluate, such as the 'srecip' function. A problem also occurs at initial startup, because not all operators have all arguments ready, which may block the evaluation of other nodes, which may in turn block the evaluation of the operator, etc, resulting in deadlock. The problem occurs often, because the bounds on value nodes are max and min operators of many arguments.

To solve this problem, the max and min operators are evaluated differently according to whether the sup or inf is desired. The sup_of_max (inf_of_min) operator does not evaluate until all arguments are ready, because a higher (lower) bound may be necessary as other arguments become ready, and the sup function is only allowed to decrease (increase). However, the inf_of_max (sup_of_min) operator can evaluate when one argument is ready, because later arguments either have no effect or improve the bound. This allows the results to propagate through sections of the network not yet evaluated or never able to be evaluated.

Complete networks are formed by connecting together sets of network modules designed to be evaluated in parallel. Ideally, each value, operation, interface and variable node could be stored in a separate processor. The whole network could be evaluated synchronously or asynchronously in a MIMD processor with non-local connectivity, because expression relationships are not regular. Each processor would continuously poll its inputs and update its value when appropriate. Since the size of the expression relating two values is typically less than 50 terms, and since it appears to only take a few iterations for the networks to converge, a complete computation (thresholding the allowable changes) is expected to occur rapidly.

So far, we have only evaluated the networks serially. This does not affect the final outcome, as the final bounds always represent the best achievable by the network and the bounds can only get tighter, irrespective of the order of bounding.

To improve serial efficiency, rather than continuously re-compute all nodes, dependency linkages record which nodes need re-computing whenever a value changes. Moreover, value nodes are updated only when their bounds improve, thus limiting the propagation of changes. The serial evaluation of a network is started by declaring all constants and external variables to have changed.

It is easy to show that the networks must converge asymptotically, that is, not oscillate. At any time the current bounds on a value V must be true. Then, if an upper bounding expression increases in value, the original bounds on V must still hold, as it then makes no sense to increase the range of potential values for V. Hence, the bounds can only get tighter, though perhaps asymptotically. As the bounds can only be equal (inconsistency is declared if they cross), each bound has a limit, so must converge. In practice, when the change in a value is below a threshold, no change is recorded. Also, when a value becomes too large or small, it is treated as plus or minus infinity. These force convergence (by limit theorems).

6 Prototype Constraint Modules

The intention behind the use of constraint modules is to precompile their algebraic relationships into network modules in advance (an slow process) and then allocate, link and evaluate these dynamically during execution (a much faster process). The use of modules also makes network construction much cleaner.

For visual geometric reasoning, the number of necessary primitive constraint modules is surprisingly small - six. Analysis of the types of primitive relationships showed that the following mappings are enough (though more complicated constraints could be derived in situations supplying several quantities - such as when a vector is measured at a known point). The constraints are:

(1) parameter limit - bounds the allowable range of a scaler model parameter M

$$| M_{obs} - M_{nominal} | \leq \delta_M$$

(2) vector constraint - stating that two vectors are within an an-

gular tolerance:

$$\underline{V}_1 \cdot \underline{V}_2 \geq \varepsilon_1$$

equivalent to

$$| \underline{V}_1 - \underline{V}_2 | \leq \varepsilon_2$$

(3) point location constraint - stating that two points are within a spatial tolerance:

$$| \underline{P}_1 - \underline{P}_2 | \leq \varepsilon$$

(4) position transform - links two reference frame positions by the transformation (another relative position) between them

$$T_t(T_1) = T_2$$

(5) vector transform - links four vectors by a reference frame transformation (relative position)

$$T(\underline{V}_1) = \underline{V}_2 \qquad T(\underline{V}_3) = \underline{V}_4$$

Both pairs of vectors need not be used, but are necessary for completely deducing the transform.

(6) point transform - links two points by a reference frame transformation (relative position)

$$T(\underline{P}_1) = \underline{P}_2$$

For each of these six algebraic relations, a set of algebraic constraints has been specified (appendix A). The sets can then be compiled into a prototype network modules. Thus, during scene analysis, whenever a new constraint is added (such as when a new model vector is paired with a data vector), then the appropriate modules are copied and linked together.

The size of the modules is summarized in table 1.

Table 1: Primitive Geometric Module Sizes

Relation	Value Nodes	Operation Nodes
parameter limit	3	19
vector constraint	8	793
point constraint	7	264
position transform	21	2239
vector transform	22	1535
point transform	13	1167

+ - lists number of value and operation nodes

If a relationship is more complicated, then it is usually expressible using several of these modules. For example, to express the relationship between a transformed (T) model vector ($\underline{V}_m$) and an observed data vector ($\underline{V}_d$) we use the following modules:

$T(\underline{V}_m) = \underline{V}_t$ (Module 5)

$| \underline{V}_d - \underline{V}_t | < u$ (Module 2)

If parameters are included in a model (such as positional degrees of freedom or size variations), then constraint (1) applies in addition to any other deriving from observed geometric relationships. Further, a priori position information can usually be expressed using these constraints.

Additional constraint modules for other geometric constraints can be created as necessary.

Initially, vector and point transform modules were designed to relate only a single pair of features. This was sufficient for deducing (e.g.) a second vector given the first vector and the transformation. However, a single pair of vectors can only partially bound the transformation. Using a pair of these modules with connected transformation ports allows some bounding information to propagate between modules, but, unfortunately, not enough. Each individual module communicates only the bounds on the components of the rotation quaternion, but not the relationships between components and thus loses information, even though the bounds were intersected. More information could be communicated (such as the plane that the vector part of the quaternion must lie in), but the simplest solution is to provide a module for the simultaneous rotation of two vectors.

7 Implementation of the Geometric Reasoning Functions in the Network

In the introduction, we introduced the five important geometric reasoning functions. Here, we describe how they are implemented when using the network:

LOCATE - the model-to-data feature pairings provide the data for constraint modules expressing the relationships between positions and model and data features. The positions are implicit in the constraints and numerically bounded in the network.

MERGE - separate position constraints are integrated by connecting their respective constraint modules appropriately. Inconsistent constraints cause the network to reach an inconsistent state, which signals incorrect model selections or model-to-data pairings.

TRANSFORM - constraint module (4) expresses this function.

INVERT - all connectors on the constraint modules are simultaneously both inputs and outputs, so constraint modules (4), (5) and (6) express both the forward and inverse position mapping.

PREDICT - connected constraint modules express the relationships between positions and model and data vectors and points and thus produce the data parameters consistent with a set of position bounds and model parameters.

The implementation of these ideas used connected instances of the modules described previously. It is easy to create subroutines allocating modules given the connecting interface variables. Standard configurations of modules are also easily allocated. When the model-based image understanding program made feature-model pairings, instances of the modules were allocated and connected according to the model and reference frame relationships. Interface variables represent the input and output values, and the reference frame. Evaluation tests consistency and resolves the reference frame, as well as predicts image feature positions.

8 Example

This section illustrates the possibilities for doing geometry in a network with an example of estimating an object's 3D orientation. Assume the following model direction vectors

$$\underline{m}_1 = (-0.51,0.83,0.22)$$
$$\underline{m}_2 = (0.68,-0.23,0.69)$$

are rotated rigidly by position T to give the vectors $T(\underline{m}_1)$ and $T(\underline{m}_2)$. Then, assume we observe two data vectors

$$\underline{d}_1 = (-0.40,0.91,0.04)$$
$$\underline{d}_2 = (-0.52,-0.67,0.51)$$

that are paired with $\underline{m}_1$ and $\underline{m}_2$ respectively. (This pairing would be hypothesized by the model-based scene analysis. An example pairing might arise from using a 3D orientation discontinuity and the normal of a planar surface patch.)

The pairings are represented using an instance of module (5) from above, linked to the $\underline{m}_i$ and $\underline{d}_i$ vectors. Evaluating this simple network, the following bounds are achieved on the rotation (represented as a quaternion):

Low	(0.729,0.249,-0.622,-0.141)
High	(0.731,0.252,-0.618,-0.139)

where the true value is

$$(0.73,0.25,-0.62,-0.14)$$

The result required 46 network update cycles with 3919 operation node evaluations, where all evaluations in each cycle could be executed in parallel. (As most network units are simple, something like 1 microsecond would be a nominal value for a network cycle time.) This gives the time necessary for values to completely propagate through the layers of simple function units several times before convergence.

Now, suppose instead we anticipate that the vectors $\underline{d}_i$ are displaced from their true position by some observational error of maximum magnitude ε. We then know

$$|\ T(\underline{m}_i) - \underline{d}_i\ | \leq \varepsilon$$

so we can now use two instances of module (2).

Evaluating this network with r = 0.05 produces the new bounds on the estimated rotation:

Low .0.615,0.149,-0.810,-0.220)
High (0.804,0.370,-0.438,-0.076)

which contains the correct rotation given above (63 network update cycles involving 7591 node evaluations).

An alternative to using module (2) entails setting the bounds on the $\underline{d}_i$ directly, which allows for non-isotropic errors. More exact error relationships could be represented algebraically and linked by other modules.

To make a third model-to-data vector pairing, we connect new instances of modules (2) and (5). This final network is shown in figure 3. Suppose this new vector is observed with ε = 0.02. Then, the new estimated rotation is:

Low (0.652,0.173,-0.805,-0.194)
High (0.799,0.341,-0.464,-0.097)

which has reduced variance (52 network update cycles involving 15411 node evaluations). The average estimated parameter value is:

(0.725,0.257,-0.634,-0.145)

which compares favourably with the true value. These bounds give the full range of allowable variation, instead of a statistical estimate.

This example demonstrates combining solutions to several constraint problems to fully constrain a larger problem. One design criterion behind the network solution was extendibility, which this modular approach allows. Hence, as more evidence is obtained, more network modules can be connected to further refine parameter estimates, especially as the pre-defined modules are usable for a variety of geometric relationships.

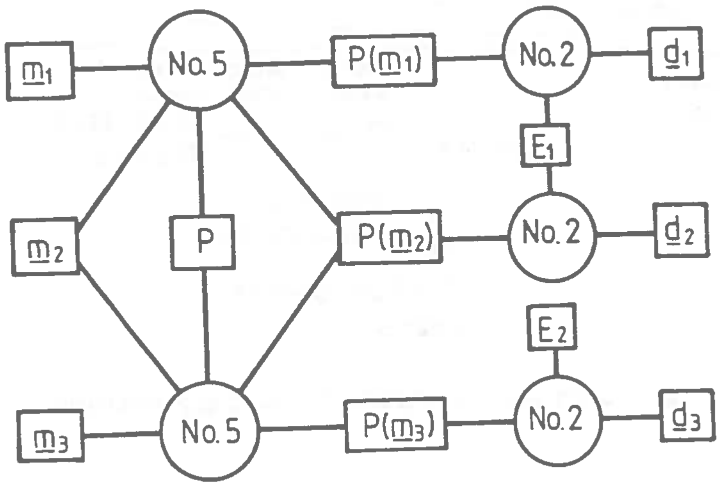

Figure 3: object orientation from three pairs of direction vectors

9 Discussion and Conclusions

The goal of this work was to produce a geometric reasoning engine suitable for massive parallelism - for which this network structure is ideally suited. The processing elements are simple and the connections orderly and well defined. The module structure allows easy packaging of standard reasoning functions, with a simple connection mechanism.

The methodology we are investigating is summarised in figure 4 below. At the top are the sets of algebraic constraints associated with particular geometric relationships. Image observables are represented by variables at this stage. These constraints are then algebraically manipulated to produce a set of inequalities bounding each variable. An extended symbolic CMS based on ACRONYM's CMS is then used to form tighter bounds on each variable. The simplified inequalities are then used to define a network fragment, that are associated with object hypotheses to form complete networks (with inter-linkages through shared variables). When observable variables get bound to measured values the other variables (e.g. model or position variables) are forced into consistency.

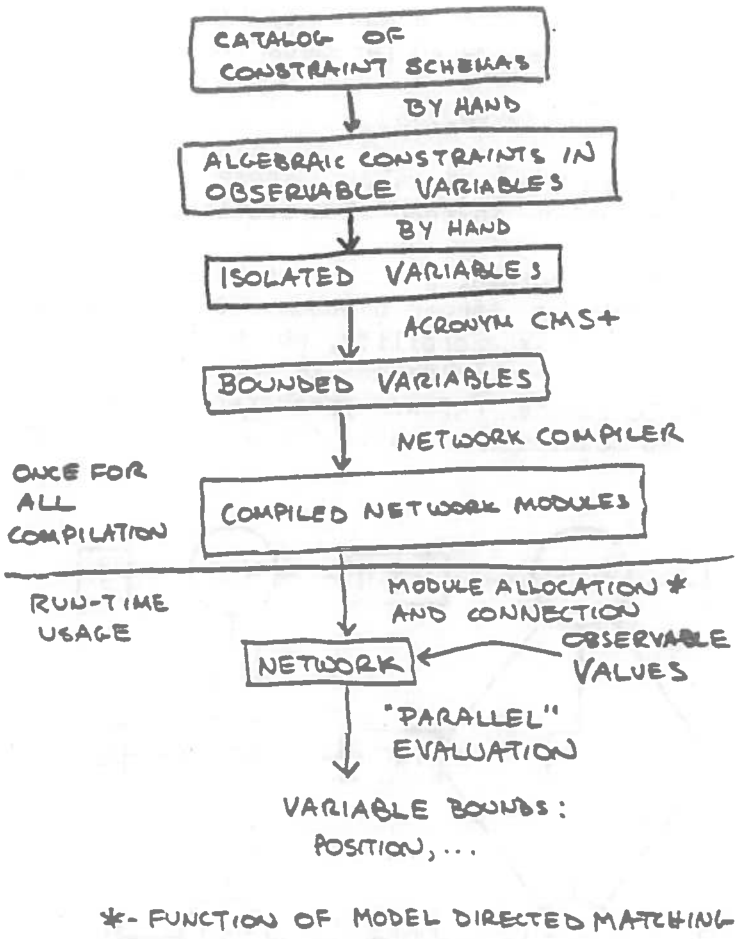

Figure 4: Network Definition from Geometric Constraints

Both defining the full constraint set and applying the symbolic CMS to produce bounding expressions can be done at model-compilation time. This may be slow, because there may be many variables in the expressions at this stage. However, this is unimportant, because this stage is done off-line. The only occasions for repeating the analysis are:

new constraints types become understood and generated, and
new CMS techniques are implemented.

ACRONYM's CMS was optimal when producing numerically bounded variables (i.e. not resulting in expressions in other variables) over sets of linear constraints, based on work by Bledsoe [Bledsoe 1974] and Shostak [Shostak 1977]. Since we reproduce the symbolic reasoning in the network, but substitute values for the data variables later, it is likely to have the same performance on linear constraint sets, but we have not proved this result. Unfortunately, as the geometric constraints are generally non-linear, we cannot expect optimality. However, the extensions to the symbolic CMS should improve performance.

It also turned out that using the network provided improved behaviour over simply using the extended symbolic CMS. This was because the iterative evaluation in the network allowed recursive calculations to converge.

While the network structure is capable of implementing many computations, its use here was for geometric reasoning. Six modules were needed for the standard geometric reasoning functions. The sizes of several of the modules was larger than hoped for - implying substantial copying costs if dynamically created.

The various aspects of the use of the network to support the geometric reasoning functions were demonstrated with an example of estimating an object's 3D reference frame.

10 References

[Ballard and Tanaka 1985] Ballard, D. and Tanaka, H., "Transformational Form Perception in 3D: Constraints, Algorithms and Implementation", Proc. 9th Int. Joint Conf. on Artif. Intel., pp 964-968, 1985.

[Bledsoe 1974] Bledsoe, W. W., "The sup-inf method in Presburger arithmetic", Memo ATP 18, Dept. of Math. and Comp. Sci., Univ. of Texas at Austin, 1974.

[Brooks 1981] Brooks, R. A., "Symbolic Reasoning among 3D Models and 2D Images", Stanford AIM-343, STAN-CS-81-861, 1981.

[Davis 1987] Davis, E., "Constraint Propagation with Interval Labels", Artificial Intelligence, Vol 32, No 3, July 1987, pp281-331.

[Fisher 1985], Fisher, R., "SMS: A Suggestive Modeling System for Object Recognition", University of Edinburgh, Department of Artificial Intelligence Working Paper 185, 1985.

[Fisher 1986] Fisher, R. B., "From Surfaces to Objects: Recognizing Objects Using Surface Information and Object Models", PhD Thesis, University of Edinburgh, 1986.

[Fisher 1987a] Fisher, R. B., "SMS: A Suggestive Modeling System for Object Recognition", Image and Vision Computing, Vol 5, No 2, May 1987, pp97-104.

also available as: Dept. of Artificial Intelligence Research Report 298, University of Edinburgh, 1987.

[Fisher 1987b] Fisher, R. B., "A PROLOG Version of ACRONYM's CMS (Version 1.1)", Dept. of Artificial Intelligence Software Paper ***, University of Edinburgh, 1987.

[Fisher and Orr 1987] Fisher, R. B., Orr, M., J., L., "Geometric Constraints from 2 1/2D Sketch Data and Object Models", Dept. of Artificial Intelligence Research report 318, University of Edinburgh, 1987.

also presented at 1986 IBM Conference on Geometric Reasoning.

[Freuder and Quinn 1985] Freuder, E. C., Quinn, M. J., "Taking Advantage of Stable Sets of Variables in Constraint Satisfaction Problems", Proc. 9th IJCAI, pp 1076-1078, 1985.

[Hinton and Lang 1985] Hinton, G. and Lang, K., "Shape Recognition and Illusory Conjunctions", Proc. 9th Int. Joint Conf. on Artif. Intel., pp 252-259, 1985.

[Marr 1982] Marr, D., Vision, W. H. Freeman, San Francisco, 1982.

[Orr 1987] Orr, M.J.L., "Geometric constraints in 3D computer vision", Dept. of Artif. Intel., working paper ***, Edinburgh University, 1987b.

[Orr and Fisher 1987] Orr, M. J. L., Fisher, R. B., "Geometric Reasoning for Computer Vision", Image and Vision Computing, Vol 5, No 2, pp233-238, 1987.

[Rosenfeld 1978] Rosenfeld, A., "Iterative Methods in Image Analysis", Pattern Recognition, Vol 10, pp181, 1978.

[Shostak 1977] Shostak, R. E., "On the sup-inf method for proving Presburger formulas", J. Assoc. Comput. Mach., 24, pp529-543, 1977.

Appendix A. Algebraic Expressions for the Six Constraint Modules

The following gives the algebraic relationships expressed in the six constraint modules. Some modules express the relationships in different ways, according to convenience, because sometimes the network solves the problem more easily with one form instead of another. There may also be inequality bounds on an variable, even though the exact relationship is also present, to help convergence.

(1) parameter limit

$| M_{obs} - M_{nominal} | \leq \delta_M$

$0 \leq \delta_M \leq +\infty$

(2) vector constraint

$\underline{V}_1 \cdot \underline{V}_2 \geq \varepsilon_1$

$| \underline{V}_1 - \underline{V}_2 |^2 \leq \varepsilon_2 * \varepsilon_2$

$2*(1 - \varepsilon_1) = (\varepsilon_2)\hat{}2$

$| \varepsilon_1 | \leq 1$ and $| \varepsilon_2 - 1 | \leq 1$

$| V_{1x} - V_{2x} | \leq \varepsilon_2$ and $| V_{1y} - V_{2y} | \leq \varepsilon_2$ and $| V_{1z} - V_{2z} | \leq \varepsilon_2$

$\underline{V}_1 \cdot \underline{V}_1 = 1$ and $\underline{V}_2 \cdot \underline{V}_2 = 1$

$| V_{1x} | \leq 1$ and $| V_{1y} | \leq 1$ and $| V_{1z} | \leq 1$

$| V_{2x} | \leq 1$ and $| V_{2y} | \leq 1$ and $| V_{2z} | \leq 1$

(3) point location constraint

$| \underline{P}_1 - \underline{P}_2 |^2 \leq \varepsilon^2$

$| P_{1x} - P_{2x} | \leq \varepsilon$ and $| P_{1y} - P_{2y} | \leq \varepsilon$ and $| P_{1z} - P_{2z} | \leq \varepsilon$

$0 \leq \varepsilon \leq +\infty$

(4) position transform

$T_t(T_1) = T_2$

where:

$T_i = (r_i, t_i)$,
r_i are quaternions with unit norm (rotations)
t_i are quaternions representing pure vectors (translations)
@ denotes quaternion multiplication
' denotes quaternion complement

then:

$r_1 = r_t' @ r_2$

$r_2 = r_t @ r_1$

$r_t = r_2 @ r_1'$

$t_1 = r_t' @ (t_2 - t_t) @ r_t$

$t_2 = r_t @ t_1 @ r_t' + t_t$

$t_t = t_2 - r_t @ t_1 @ r_t'$

$r_1 @ r_1' = r_2 @ r_2' = r_t @ r_t' = 1$

$t_1 @ t_1' = (t_2 - t_t) @ (t_2' - t_t')$

(5) <u>vector transform</u>

$T(\underline{V}_1) = \underline{V}_2$ and $T(\underline{V}_3) = \underline{V}_4$

where:

$T = (r,t)$
r is a quaternion with unit norm (rotations) = $(r_0,\underline{r})$
t is a quaternion representing a pure vector (translations)
@ denotes quaternion multiplication
' denotes quaternion complement
$\times$ = cross product

$r_0(\underline{V}_2 - \underline{V}_1) = \underline{r} \times (\underline{V}_2 + \underline{V}_1)$

$r_0(\underline{V}_4 - \underline{V}_3) = \underline{r} \times (\underline{V}_4 + \underline{V}_3)$

$r @ r' = 1$

$r_0 \geq 0$ (implies only one solution)

$\underline{V}_1 = r' @ \underline{V}_2 @ r$

$\underline{V}_2 = r @ \underline{V}_1 @ r'$

$\underline{V}_3 = r' @ \underline{V}_4 @ r$

$\underline{V}_4 = r @ \underline{V}_3 @ r'$

(6) <u>point transform</u>

$T(\underline{P}_1) = \underline{P}_2$

where:

$T = (r,t)$
r is a quaternion with unit norm (rotations)
t is a quaternion representing a pure vector (translations)
@ denotes quaternion multiplication
' denotes quaternion complement

$\underline{P}_2 = r \mathbin{@} \underline{P}_1 \mathbin{@} r' + t$

$t = \underline{P}_2 - r \mathbin{@} \underline{P}_1 \mathbin{@} r'$

$\underline{P}_1 = r' \mathbin{@} (\underline{P}_2 - t) \mathbin{@} r$

$| \underline{P}_1 | = | \underline{P}_2 - t |$

$r \mathbin{@} r' = 1$

$(\underline{P}_2 - t) \mathbin{@} r = r \mathbin{@} \underline{P}_1$