On the Use of Interval Arithmetic in Geometric Branch and Bound Algorithms

Thomas M. Breuel

Palo Alto Research Center
Palo Alto, CA 94304, USA
E-mail: tmb@parc.com

Abstract

Branch and bound methods have become established methods for geometric matching over the last decade. This paper presents techniques that improve on previous branch and bound methods in two important ways: they guarantee reliable solutions even in the presence of numerical roundoff error, and they eliminate the need to derive bounding functions manually. These new techniques are compared experimentally with recognition-by-alignment and previous branch and bound techniques on geometric matching problems. Novel methods for non-linear baseline finding and globally optimal robust linear regression using these techniques are described.

Key words: visual object recognition, geometric matching, interval arithmetic

1 Introduction

Extensive research has been carried out over the last few decades on algorithms for geometric matching (Grimson, 1990). The introduction of branch and bound algorithms provided perhaps the first practical algorithms for geometric matching that guarantee globally optimal results (Breuel, 1992; Huttenlocher et al., 1993; Hagedoorn and Veltkamp, 1997; Mount et al., 1999; Jurie, 1999; Olson, 2001). In parallel with the development of branch and bound algorithms in computer vision, interval arithmetic has been developed for applications in computer graphics, robotics, and global optimization (Jaulin et al., 2001).

This paper describes the implementation of branch and bound algorithms using interval arithmetic and presents experimental results showing that interval arithmetic-based branch and bound algorithms for geometric matching are practical. This yields geometric matching algorithm that are numerically reliable, easy to implement, and that can be used with a very wide variety of quality of match functions.

The paper briefly reviews the necessary techniques from branch and bound matching and interval arithmetic, giving the reader the information necessary to apply the method to their own matching problems.

2 Geometric Model Matching

To establish notation and review geometric matching, let us begin by defining the problem of geometric matching under translation and rotation (other matching problems will be considered below). Assume we are given a set of model features (points in the image plane), $M = \{m_1, \ldots, m_r\} \subseteq \mathbb{R}^2$ and a set of image features $B = \{b_1, \ldots, b_s\} \subseteq \mathbb{R}^2$. Optimal matching under a set of transformations $\mathbb{T}$ can then be written as

$$T_{\text{opt}} = \arg\max_{T \in \mathbb{T}} Q(T) \tag{1}$$

where we define the quality of match as a function $Q(T)$

$$Q(T) = \sum_{i=1}^{r} \max_{j=1\ldots s} \phi_\epsilon(||Tm_i - b_j||) \tag{2}$$

Here, ϕ_ϵ is a weighting function on the location error. The set of transformations $\mathbb{T}$ can be any of a wide variety of transformation spaces, such as the space of translations of $\mathbb{R}^2$, the space of translations and rotations of $\mathbb{R}^2$, or the space of translations, rotations, and scale changes of $\mathbb{R}^2$. For bounded error matching, corresponding to maximum likelihood solutions under a uniform error model, the weighting function is $\phi_\epsilon(x) = 1$ if $x < \epsilon; 0$ otherwise. For robust least square matching, corresponding to maximum likelihood solutions under a Gaussian error model (Wells III, 1997), the weighting function is:

$$\phi_\epsilon(x) = \max(0, 1 - \frac{x^2}{\epsilon^2}) \tag{3}$$

3 Branch and Bound Algorithms

With these definitions of quality of match functions, finding globally optimal maximal bounded error, robust least square, or maximum likelihood matches now amounts to a global optimization of the functions Q derived in the previous section. This can be accomplished using branch and bound optimization algorithms. Applications of branch and bound algorithms to matching problems have been described in the computer vision literature (Breuel, 1992; Huttenlocher et al., 1993; Hagedoorn and Veltkamp, 1997; Mount et al., 1999; Jurie, 1999; Olson, 2001). Let us briefly review how these algorithms work.

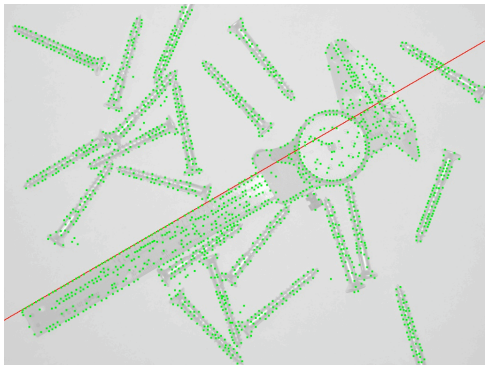
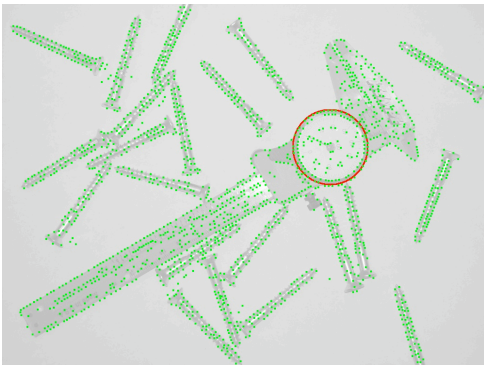
(a)  (b) 

Fig. 1. Maximum Likelihood Line and Circle Detection
Example of a globally optimal maximum likelihood match for a line and a circle. The image was captured with a digital camera and the features consist of 2248 points samples from edges extracted using a Canny edge detector. The likelihood function was calculated using the interval arithmetic method described in the paper.

Let the entire parameter space $\mathbb{T}$ consist of a Cartesian product of parameter ranges $[\underline{t}_0, \overline{t}_0] \times [\underline{t}_1, \overline{t}_1] \times \ldots \times [\underline{t}_n, \overline{t}_n]$. Here, the notation $\underline{t}$ and $\overline{t}$ just denotes two different real variables, $\underline{t} \leq \overline{t}$. We refer to such Cartesian products of intervals as *boxes* or *interval vectors*. The input to the algorithm consists of the likelihood or quality of match function to be optimized, $Q(T)$. Let $T_i \subseteq \mathbb{T}$ be a box – an axis-aligned, hyper-rectangular subset of the entire parameter space. Furthermore, let $\overline{q}_i = \hat{Q}(T_i)$ be an upper bound on the maximum of Q over T_i, that is $\hat{Q}(T_i) \geq \max_{T \in T_I} Q(T)$. The algorithm maintains a priority queue of boxes, ordered by the upper bounds $\overline{q}_i$:

(1) Let $T_0 = \mathbb{T}$, Compute $\overline{q}_0 = \hat{Q}(T_0)$ and insert $(T_0, \overline{q}_0)$ into the priority queue, using the upper bound of $\overline{q}_0$ as the priority.
(2) Remove the top element from the priority queue and call it $(T_i, \overline{q}_i)$.
(3) Check for termination, for example based on the desired numerical accuracy of the solution or other criteria (see below).
(4) Otherwise, subdivide T_i into smaller boxes, for example, by splitting it in half along a coordinate.
(5) For each T_s, compute $\overline{q}_s = \hat{Q}(T_s)$ and insert $(T_s, \overline{q}_s)$ into the priority queue, using the upper bound $\overline{q}_s$ as the priority.
(6) Continue at Step 2.

This approach to geometric matching has been demonstrated to work well in practice and has been used in a number of applications (Breuel, 1992; Huttenlocher et al., 1993; Hagedoorn and Veltkamp, 1997; Mount et al., 1999; Jurie, 1999; Olson, 2001). One technique that greatly simplifies its implementation is the use of *matchlists* (Breuel, 2002a). Briefly, in addition to the T_i, we also keep track, for each node in the priority queue, of a list of correspondences between image and model features that still result in a non-zero contribution to the overall quality of match function. For a more detailed discussion, the reader is referred to the references.

A key problem in its implementation is the derivation of the bounding function $\hat{Q}(T_i)$. Existing algorithms all rely on a case-by-case derivation. For example, Breuel (1992) relies on the linear relationship described in Baird (1985). Huttenlocher et al. (1993), Hagedoorn and Veltkamp (1997), and Mount et al. (1999) rely, explicitly or implicitly, on Lipshitz bounds or uniform continuity.

Another issue with a straightforward implementation of the algorithm is that it is not necessarily *reliable*, in the sense that floating point roundoff errors may cause the algorithm to lose solutions. In particular, without special precautions, a numerical evaluation of $\hat{Q}(T_i)$ is not guaranteed to be an upper bound for $\max_{T \in T_i} Q(T)$.

It turns out that both problems can be addressed through the use of interval arithmetic. Mount et al. (1999), suggests "propagating uncertainty through" the computation of $Q(T)$ without elaborating further. Interval arithmetic provides us with a simple, well-defined, and well-studied means for doing this.

4 Interval Arithmetic

Interval arithmetic was originally proposed as a means of bounding the error of numerical computations. For detailed information about interval arithmetic and its implementation, the reader is referred to the references (Jaulin et al., 2001; Hickey et al., 2001). Here, we will limit ourselves to an informal introduction and a description of some of the aspects of interval arithmetic we need for the computation of $\hat{Q}(T_i)$.

Let $*$ be any of the standard arithmetic operations, $+$, $-$, or $\times$. Furthermore, let $\mathbb{I} = \{[\underline{u}, \overline{u}] | \underline{u} \leq \overline{u};\ \underline{u}, \overline{u} \in \mathbb{R}\}$ be the set of all closed finite intervals [1] Let $x = [\underline{x}, \overline{x}] \in \mathbb{I}$ and $y = [\underline{y}, \overline{y}] \in \mathbb{I}$ be two intervals. We extend $*$ to an operation over intervals $[*]$ by defining:

$$\begin{aligned} x[*]y &= [\underline{x}, \overline{x}]\ [*]\ [\underline{y}, \overline{y}] \\ &= [\min\{u * v | u \in [\underline{x}, \overline{x}], v \in [\underline{y}, \overline{y}]\}, \max\{u * v | u \in [\underline{x}, \overline{x}], v \in [\underline{y}, \overline{y}]\}] \end{aligned} \quad (4)$$

A function f has a natural extension to interval arithmetic, often denoted $[f]$, that is defined as follows:

$$[f]([\underline{x}, \overline{x}]) = [\inf\{f(u) | u \in [\underline{x}, \overline{x}]\}, \sup\{f(u) | u \in [\underline{x}, \overline{x}]\}] \quad (5)$$

In the rest of this paper, to simplify notation, we will normally make no explicit distinction between arithmetic operations and functions over the reals and their

[1] We will be using only a simple form of interval arithmetic in this paper that uses no infinities. For a more general treatment, please refer to the references.

interval equivalents. It should be noted that if we identify the real number x with the interval $[x, x]$, the above equations reduce to arithmetic over the real numbers; in that sense, interval arithmetic is an extension of arithmetic over the real numbers.

It is easy to see that we can express the standard arithmetic operators more easily than in Equation 4. For example, $[\underline{x}, \overline{x}] + [\underline{y}, \overline{y}] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$ and $[\underline{x}, \overline{x}] \times [\underline{y}, \overline{y}] = [\min(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}), \max(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y})]$. Furthermore, for a non-decreasing function f satisfying some additional technical conditions, $f([\underline{x}, \overline{x}]) = [f(\underline{x}), f(\overline{x})]$. This works for $\log$, $\exp$, and other functions. For functions like $\sin$, abs, $\tan$, sign and others, a more careful case-by-case analysis is necessary. Interval arithmetic can also be extended to interval arithmetic modulo 2π, simplifying the implementation of matching algorithms involving angles.

In these definitions of interval arithmetic operations, the bounds themselves are still real numbers, not finite precision floating point numbers, and as such are subject to roundoff errors. This means that we cannot carry out interval arithmetic exactly using floating point computations. However, it is quite easy to find intervals representable as floating point arithmetic that *include* the infinite precision solution. As it turns out, this is sufficient to guarantee numerical reliability of computations based on interval arithmetic using finite precision floating point numbers (Hickey et al., 2001).

For example, let $\oplus$ be floating point addition rounded to machine precision; this is what a modern processor will compute. Furthermore, let prevfp(x) be the largest floating point number representable that is smaller than, and different from, x and let nextfp(x) be the smallest floating point number representable that is larger than, and different from, x. Then it is easy to see that $a+b \in [\text{prevfp}(a \oplus b), \text{nextfp}(a \oplus b)]$. Furthermore, $[\underline{x}, \overline{x}] + [\underline{y}, \overline{y}] \subseteq [\text{prevfp}(\underline{x} \oplus \underline{y}), \text{nextfp}(\overline{x} \oplus \overline{y})]$. This technique can be easily extended to the other arithmetic operations, as well as standard transcendental functions (Hickey et al., 2001).

These techniques can also be extended to vectors of intervals in the natural way (Jaulin et al., 2001). For example, a vector $(u, v) \in \mathbb{R}^2$ corresponds to the product of two intervals $[\underline{u}, \overline{u}] \times [\underline{v}, \overline{v}] \subseteq \mathbb{I}^2$ in interval arithmetic. We refer to such a Cartesian product of intervals as an *interval vector* or simply a *box*.

Following standard terminology, we say that an interval function $g : \mathbb{I}^n \to \mathbb{I}^m$ is an *inclusion function* for a real function $f : \mathbb{R}^n \to \mathbb{R}^m$ if

$$\forall x \in \mathbb{I}^n : \ \{v | v = f(u), u \in x\} \subseteq g(x) \tag{6}$$

That is, g is an inclusion function for a function f over real vectors if g is a function over intervals such that if an interval vector x includes a vector u, them image of x under g includes the image of the vector u under f. Let us define $\text{width}([\underline{x}, \overline{x}]) = \overline{x} - \underline{x}$. We then say that g is a *convergent inclusion function* if, for any sequence of intervals $x_i \in \mathbb{I}^n$ satisfying $\lim_{i \to \infty} \text{width}(x_i) = 0$, the inclu-

sion function satisfies $\lim_{i\to\infty} \text{width}(g(x_i)) = 0$; another standard theorem states that convergent inclusion functions also satisfy $g([x,x]) = f(x)$.

If a function $f : \mathbb{R}^n \to \mathbb{R}^m$ is expressed in terms of arithmetic operators and standard functions ($\sin$, $\exp$, etc.) and we symbolically replace the real arithmetic operators and standard functions with their equivalents in interval arithmetic, as defined in Section 4, we obtain an interval function $f_{\text{NI}} : \mathbb{I}^n \to \mathbb{I}^m$. It is a standard theorem in interval arithmetic that this interval function is an inclusion function for f, referred to as the *natural inclusion function*. Natural inclusion functions are not necessarily minimal: rewriting an expression into a mathematically equivalent form might give rise to a different inclusion function. The reason is that $[f \circ g](x) \subseteq ([f] \circ [g])(x)$, with equality not guaranteed. But natural inclusion functions are convergent if all the operators and functions involved in their construction are continuous. This turns out to be sufficient for ensuring convergence of the branch and bound geometric matching algorithm. When natural inclusion functions are expressed in finite precision arithmetic, using the rounding methods mentioned above, they become convenient means of computing reliable (though not necessarily tight) bounds for functions over intervals or boxes. The resulting functions also tend to have useful convergence properties, although this needs to be established on a case-by-case basis.

5 Implementing Geometric Matching with Interval Arithmetic

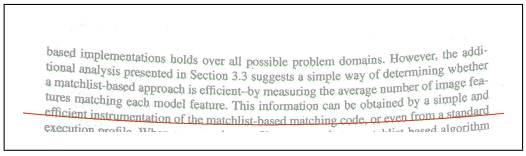
Given these preliminaries, it is now easy to apply interval arithmetic to the computation of the upper bound functions $\hat{Q}(T_i)$. Note that, by construction, the T_i are interval vectors. We can therefore compute the natural inclusion function Q_{NI} of the interval vector T_i.

Theorem. *Let* $[\underline{q}, \overline{q}] = Q_{\text{NI}}(T_i)$. *Then,* $\overline{q} \geq \max_{T \in T_i} Q(T)$.

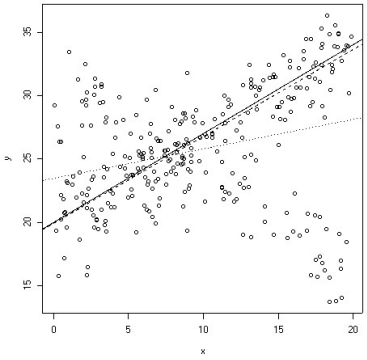
Proof. This follows directly from the properties of natural inclusion functions. •

Furthermore, when Q is continuous, then Q_{NI} is convergent, a property that is necessary to ensure correctness and convergence of the branch and bound algorithm.

If $q_0 = [\underline{q}_0, \overline{q}_0]$ and $q_1 = [\underline{q}_1, \overline{q}_1]$ are the quality of match values computed for the top two entries in the priority queue, we know that q_0 corresponds to a globally optimal solution when $\underline{q}_0 > \overline{q}_1$. The reason is that, because the priority queue is ordered by upper bounds, we know that any remaining interval $[\underline{q}_i, \overline{q}_i]$ must have an upper bound that is lower than than $\overline{q}_1$. When the intervals q_0 and q_1 are still overlapping, or when $\overline{q}_0 - \underline{q}_0$ is not small enough for the desired numerical accuracy, we need to subdivide rectangles further until one interval is strictly greater than the other (assuming that the two corresponding local optima have unequal quality of match values). If this procedure does not find a solution, the search is termi-



(a)



(b)

Fig. 2. Nonlinear Baseline Matching and Robust Regression

In Figure 2(a), the largest nonlinear baseline is found in a digital camera image of a page with significant page curl using the algorithm described in the text (additional baselines can be found with an n-best version of the algorithm). Figure 2(b) shows linear regression using the algorithm described in the text. The solid line indicates ground truth. The dashed line shows the optimal match found using the technique described in the text. The dotted line is a standard least square regression.

nated when a predetermined limit of numerical accuracy is reached, that is, when $\mathrm{diam}(T) < \delta$ for some chosen δ (δ might, for example, be determined by floating point machine precision). When terminated like that, a reliable numerical algorithm may return two (or more) incomparable optima together with their associated quality of match intervals. The meaning of such a solution is that the quality of match function cannot be evaluated sufficiently accurately under the given numerical precision to determine which of the solutions is actually optimal. Note that this is a limitation of the implementation of the quality of match function using finite precision arithmetic, not the search algorithm itself. In practice, this case is very rare for common choices of δ.

By combining interval arithmetic with branch and bound algorithms for geometric matching in this way, we arrive at a global optimization algorithm very similar to the global optimization algorithms described in Hansen (1980). In fact, many of the results on convergence described in that work carry over. Geometric branch and bound algorithms using interval arithmetic differ from the original algorithms describe in Hansen (1980) in that the geometric algorithms use matchlists or point location data structures to speed up the computation of Q and that a number of practically important techniques have been worked out in the geometric case to allow, for example, the computation of n-best matches.

Computation of the natural inclusion function $[Q](T)$ corresponding to a quality of match function $Q(T)$ is very simple in many modern programming languages using features like overloading: $Q(T)$ is written down in terms of standard arithmetic operators and functions, but using interval data types, and the compiler automatically generates the corresponding natural inclusion function. Given suitable definitions

of 2D vectors, real intervals, and 2D interval vectors, this is how a term in the quality of match function from Equation 1 is computed in a C++ implementation:

```
interval eval_likelihood_term(vec2 image,vec2 model,ivec2 translation,ivec2 rotation) {
  return phi_epsilon(norm(cross(rotation,model)+translation-image));
}
```

This, and a range of parameters, is essentially the only information a user of the algorithm needs to change in order to adapt the method to different recognition or matching problems.

6 Experimental Results

6.1 Performance of 2D Matching

The bounds obtained from an interval arithmetic computation might be considerably worse than those obtained from a manual derivation of bounds, due to problems like the "wrapping problem" (Jaulin et al., 2001). We need to establish experimentally that this does not present a problem in the application of the algorithm. This paper therefore compares branch and bound methods with manually derived bounds and interval arithmetic; recognition by alignment (Huttenlocher and Ullman, 1987) is also used as a control.

The first method in these experiments, recognition by alignment, picks two points from the model, two points from the image, and computes a transformation that aligns the model pair with the image pair; the transformation is used to project the remaining model points into the image and count the number of matching features. Note that this is only an approximation to the bounded error matching problem, not an exact solution. In the alignment algorithm implemented for these benchmarks, two point location data structures were used to speed up alignment, one to limit the set of image feature pairs considered to those consistent with the implied scale after picking a pair of model features, and a second data structure for quickly finding matching image features when all model features are transformed using the alignment transformation; it was verified experimentally that use of these data structures indeed resulted in a substantial speedup.

The second method is globally optimal branch and bound matching using matchlists as described in Section 3 and the references. The geometric component of the computation of the likelihood function was carried out using manually derived, circular upper bounds on the swept area.

The third method uses the same code as the second method, except that it substitutes the use of interval arithmetic for the manually derived formula for the computation of bounds on the swept area.

nclutter	alignment	bbound, manual	bbound, interval	bbound, opt.
50	0.676	0.62	2.1	1.1
100	2.16	1.8	5.9	3
150	4.56	3.5	11	5.9
200	7.87	6.1	20	10

Fig. 3. Running Times on 2D Matching Task

The table shows a comparison of running times of three different algorithms on a 2D matching task. The running times are given in seconds and represent averages of 200 trials. "nclutter" is the number of random, non-model-derived background features.

The fourth method is similar to the third method but simulates the performance of an optimizing compiler for interval arithmetic (see below).

Each problem instance was generated by picking 20 points from a spatially uniform distribution in a 100×100 pixel area, deleting ("occluding") 10 points picked at random, translating and rotating the remaining points with a uniformly randomly chosen translation and rotation into an area of size 512×512, perturbing the resulting locations by adding random vectors of magnitude smaller than the error bound, and inserting a variable number (`nclutter`) of clutter points with uniform distribution. This simple model of random instances of the geometric matching problem has been used by a number of authors (e.g., Mount et al., 1999; Breuel, 2002a) and has the advantage that it is easy to describe and reproduce. While it does not represent accurately every conceivable statistical distribution of image feature locations, it appears to be reasonably predictive of the performance of geometric matching algorithms on many real world problems involving significant amounts of clutter.

The running times of the methods were determined each on the same 200 problem instances. It was verified that all the results returned by the branch and bound methods were identical to each other for each instance. The results of these benchmark runs are shown in Table 3. Results are average running times per problem instance on a modern 1GHz desktop PC running Linux and using the GNU C compiler version 3.0.2. These results show an average overhead of a factor of 3.5 of the interval arithmetic implementation relative to the branch and bound implementation using manually derived bounds, and an overhead of 2.6 relative to the alignment algorithm [2] Execution profiling of the implementations suggests that the interval arithmetic method spends up to 40% of its time in rounding operations. This overhead can largely be eliminated with a better interval arithmetic library or by using a compiler with built-in support for interval arithmetic (e.g., available commercially from

[2] It should be noted that the features in these random problem instances have no orientation information associated with them. This makes the optimal matching problem considerably harder than it would otherwise be.

Sun Microsystems). Such implementations of interval arithmetic change rounding modes in only a few places, rather than invoking rounding operations for every arithmetic operation. We can estimate the performance of such an optimized implementation by removing rounding operations from the implementation, and the overhead of using interval arithmetic drops to an average of 1.7 compared to manually derived bounds.

Note that the method still works and still returns useful solutions if rounding operations are disabled in the interval arithmetic package. It still retains the advantage of being easy to implement and easy to adapt to new problems. Disabling the rounding operations just means that we cannot be certain anymore that no solutions are lost to roundoff errors.

In summary, these benchmarks on 2D geometric matching suggest that global optimization of likelihood functions as they arise in geometric matching problems using branch and bound methods and interval arithmetic are practical, even if there is some overhead compared to alignment methods and branch and bound methods with manually derived bounds. However, in return, the implementation based on interval arithmetic guarantees globally optimal solutions even in the presence of numerical roundoff errors, simplifies the implementation of bounding functions, and hence arguably reduces the risk of programming mistakes.

6.2 Line Detection

Globally optimal line finding using a branch and bound algorithm was previously described in Breuel (1996). That approach required a careful derivation of error bounds and careful attention to how parameter space was explored. In contrast, using the techniques described in this paper, globally optimal line finding under a Gaussian error model required only expressing in C++ the computation of a single term of the log likelihood function $Q(\theta, r) = \sum_i \phi_\epsilon(\text{dist}(p_i, l_{\theta,r}))$, where $\text{dist}((x, y), l_{\theta,r}) = |(x, y) \cdot (\cos\theta, \sin\theta) - r|$. An example of a match found using this implementation is shown in Figure 1. In the example, the image was captured using a digital camera and scaled down to 1024×768 pixels and 2248 edge points were extracted. The interval arithmetic algorithm found the globally optimal robust least square line match under, shown in Figure 1(a), an error bound of 5 pixels in under a second.

To compare the overhead of the non-interval arithmetic methods with the interval arithmetic methods on this task, edge samples were extracted from 400 images from the COIL-20 image database. The task was to find the globally optimal robust least square match of a line model against these points. The error bound used was 2 pixels, and the required numerical accuracy of the solutions was 10^{-2} for location and 10^{-5} for orientation (these are commonly used parameter values). The two algorithms for line finding were implemented independently in C++. It was veri-

fied that the two implementations returned the same solutions. Running times were measured and the ratio of running times for the two implementations on each of the 400 images were determined. The geometric average of this ratio is 1.3, showing that matching using interval-arithmetic has a modest overhead compared to the manually derived bounds.

6.3 Circle Detection

Similar to line finding, circle finding was implemented using the log likelihood function $Q(\theta, r) = \sum_i \phi_\epsilon(\text{dist}(p_i, c_{u,v,r}))$ and using the weighting function from Equation 3 with $\epsilon = 5$. The parameter space for circles is three-dimensional, and in addition to allowing the center to fall anywhere within the image, the range of radii considered was 30 pixels to 300 pixels. Running times on this data set were a few seconds on a desktop PC, and the optimal match is shown in Figure 1(b), suggesting that the method is practical. The only other globally optimal method for circle finding appears to be the branch and bound method described in Olson (2001); a performance comparison between that method and the method described in this paper remains to be done.

6.4 Novel Applications

To show that the use of interval arithmetic makes it easy to create novel geometric matching methods, two problems for which previously only locally optimal or heuristic methods were available were solved using interval arithmetic methods. In both cases, the implementation was no more difficult than expressing the likelihood function using C++ operators inside a generic driver routine.

Robust Nonlinear Baseline Detection Nonlinear baseline finding is an important problem in document image analysis. It occurs, for example when pages are captured with digital cameras, which often exhibit significant lens distortion, or when pages have significant page curl. An example is shown in Figure 2(a). An image was captured using a digital camera and thresholded using an adaptive thresholding algorithm. As is common for baseline finding algorithms, the lower center of the bounding box of each connected component was used as a reference point (indicated in the figure), and a quadratic baseline-and-descender model of the text line was fitted (Breuel, 2002b). The best match is indicated in Figure 2(a) by a line; additional matches can be extracted using the n-best techniques described in the references (Breuel, 2002a).

Robust Linear Regression To perform robust linear regression, we attempt to optimize the quality of match function $Q(m, b) = \sum_i \phi_\epsilon(|y_i - (mx_i + b)|)$. This is frequently done using expectation-maximization (EM) methods, which are not guaranteed to return globally optimal solutions. A globally optimal robust linear

regression algorithm was implemented using the techniques described above by expressing $Q(m, b)$ with interval arithmetic and optimizing over b and m using the methods described above. The test case shown in Figure 2(b) consists of a mixture of two lines $y = 0.7x + 20 + \nu_2$ and $y = -0.7x + 30 + \nu_2$, where ν_2 is a noise component with a Gaussian distribution and $\sigma = 2$. The technique described in this paper achieved a nearly perfect separation of the two components, returning an approximation of $y = 0.69x + 19.9$ for the first component (shown as a dashed line in the figure). In contrast, a standard least square regression returns a best fit of $y = 0.23x + 23.5$ (shown as a dotted line in the figure).

7 Discussion

This paper has presented an approach to geometric matching using branch and bound methods and interval arithmetic. The shift from an explicit derivation of bounding functions, as used in prior work on branch and bound matching algorithms, has a number of important implications.

First, in many geometric matching problems of practical interest, deriving bounding functions and implementing them correctly can be difficult and error prone. For example, in the implementation of the line finding algorithm described in Breuel (1996), the need for the interior circular arc segment was only noticed after extensive testing. The experience with implementations of globally optimal algorithms to real-world matching problems described above, several of which are for previously unsolved problems, shows that creating branch and bound matching methods becomes as simple as expressing a term of the likelihood function when interval arithmetic is used, and the resulting bounds are automatically correct.

Furthermore, the use of interval arithmetic also makes geometric matching numerically reliable. That is, unlike other methods for solving such problems, these algorithms guarantee that the algorithm terminates and no solutions are lost due to roundoff errors [3] .

In addition, the use of interval arithmetic allows us to apply a variety of numerically reliable and efficient local optimization methods like the Interval Newton method (Hansen and Greenberg, 1983). This may improve on prior local search methods (e.g., Jurie, 1999) in that it permits uniform treatment of global and local search. This remains to be explored in future work.

The experiments presented in this paper on 2D matching and line finding show that, even though it makes stronger guarantees, using interval arithmetic has comparable complexity and is only moderately slower compared to manually derived bounds.

[3] If two potential solutions cannot be discriminated to representable machine precision, both solutions are returned and their incomparability is indicated.

Implementations of branch and bound geometric matching using interval arithmetic also demonstrate that it yields useful and efficient, globally optimal matching algorithms in a variety of applications. Note that the applications to robust least square regression and robust non-linear baseline finding are novel; prior algorithms for these two problems were both considerably more complicated and not guaranteed to return optimal results.

Overall, the combination of interval arithmetic with the implementation techniques described in Breuel (2002a) appears to be currently the best choice for many geometric matching problems: it results in simple implementation that return numerically reliable, globally optimal solutions and are practical and effective for many geometric problems.

References

Baird, H. S., 1985. Model-Based Image Matching Using Location. MIT Press, Cambridge, MA.

Breuel, T. M., 1992. Fast Recognition using Adaptive Subdivisions of Transformation Space. In: IEEE Conference on Computer Vision and Pattern Recognition. pp. 445–451.

Breuel, T. M., 1996. Finding Lines under Bounded Error. Pattern Recognition 29 (1), 167–178.

Breuel, T. M., 2002a. A comparison of search strategies for geometric branch-and-bound algorithms. In: European Conference on Computer Vision. p. (in print).

Breuel, T. M., 2002b. Robust least square baseline finding using a branch and bound algorithm. In: Document Recognition and Retrieval VIII, SPIE, San Jose. pp. 20–27.

Grimson, E., 1990. Object Recognition by Computer. MIT Press, Cambridge, MA.

Hagedoorn, M., Veltkamp, R. C., 1997. Reliable and efficient pattern matching using an affine invariant metric. Technical Report RUU-CS-97-33, Dept. of Computing Science, Utrecht University.

Hansen, E., 1980. Global optimization using interval analysis – the multi-dimensional case. Numerische Mathematik 34, 247–270.

Hansen, E., Greenberg, R. I., 1983. An interval newton method. Appl. Math. Comput. 12, 89–98.

Hickey, T. J., Ju, Q., van Emden, M. H., 2001. Interval arithmetic: From principles to implementation. To appear in the Journal of the ACM; also available at `http://www.cs.brandeis.edu/~tim/`.

Huttenlocher, D., Klanderman, G., Rucklidge, W., 1993. Comparing images using the hausdorff distance. IEEE Transactions on Pattern Analysis and Machine Intelligence 15 (9), 850–63.

Huttenlocher, D. P., Ullman, S., June 1987. Object Recognition Using Alignment.

In: International Conference on Computer Vision. IEEE, Washington, DC, London, England, pp. 102–111.

Jaulin, L., Kieffer, M., Didrit, O., Walter, E., 2001. Applied Interval Analysis. Springer Verlag, Berlin.

Jurie, F., 1999. Solution of the Simultaneous Pose and Correspondence Problem Using Gaussian Error Model. Computer Vision and Image Understanding 73 (3), 357–373.

Mount, D., Netanyahu, N., Le Moigne, J., 1999. Efficient algorithms for robust feature matching. Pattern Recognition 32 (1), 17–38.

Olson, C. F., June 2001. Locating geometric primitives by pruning the parameter space. Pattern Recognition 34 (6), 1247–1256.

Wells III, W., 1997. Statistical approaches to feature-based object recognition. International Journal of Computer Vision 21 (1/2), 63–98.