A Comparison of Search Strategies for Geometric Branch and Bound Algorithms

Thomas M. Breuel

Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
tmb@parc.com

Abstract. Over the last decade, a number of methods for geometric matching based on a branch-and-bound approach have been proposed. Such algorithms work by recursively subdividing transformation space and bounding the quality of match over each subdivision. No direct comparison of the major implementation strategies has been made so far, so it has been unclear what the relative performance of the different approaches is. This paper examines experimentally the relative performance of different implementation choices in the implementation of branch-and-bound algorithms for geometric matching: alternatives for the computation of upper bounds across a collection of features, and alternatives the order in which search nodes are expanded. Two major approaches to computing the bounds have been proposed: the matchlist based approach, and approaches based on point location data structures. A second issue that is addressed in the paper is the question of search strategy; branch-and-bound algorithms traditionally use a "best-first" search strategy, but a "depth-first" strategy is a plausible alternative. These alternative implementations are compared on an easily reproducible and commonly used class of test problems, a statistical model of feature distributions and matching within the COIL-20 image database. The experimental results show that matchlist based approaches outperform point location based approaches on common tasks. The paper also shows that a depth-first approach to matching results in a 50-200 fold reduction in memory usage with only a small increase in running time. Since matchlist-based approaches are significantly easier to implement and can easily cope with a much wider variety of feature types and error bounds that point location based approaches, they should probably the primary implementation strategy for branch-and-bound based methods for geometric matching.

1 Introduction

Matching of points or other geometric primitives under geometric transformations is an important problem in many applications, including computer vision and robotics. Early work on matching has included the Generalized Hough Transform[2], heuristic search[8], and a variety of other techniques. Over the last decade, branch-and-bound techniques have become increasingly popular for geometric matching problems Such techniques work by recursively subdividing the space of parameters of the geometric transformations under consideration and computing upper bounds on the possible quality of match for any match possible within those subregions of parameter space[4]. Branch and bound techniques are attractive because they can guarantee optimality of the returned solutions to within specified numerical tolerances and because they appear to have complexities and running times similar to those of alignment methods[6].

Two major approaches to branch-and-bound techniques for geometric matching have emerged over the last decade. The first is an approach based on matchlists and (optionally) a depth-first search strategy[4]. The second is based on data structures for fast proximity searches or discrete Voronoi diagrams [14,12,9] and has its origins in earlier hierarchical matching schemes organized around multi-resolution image analysis [3,10].

Matchlist-based approaches to geometric matching are very easy to implement: all a user needs to supply is a quality-of-match function for a pair of features (and an upper bound function derived from the quality of match function). An efficient, self-contained matchlist-based branch-and-bound implementation takes a few hundred lines of C++ or Fortran code with no reference to complex data structures. This makes it easy to apply matchlist-based branch-and-bound techniques to complex matching tasks like the matching of line segments[6], text line models[7], and MDL descriptions[5]. In contrast, in approaches that rely on geometric data structures for fast proximity searches, using features like points with associated orientations, extended line segments, or groups of geometric features becomes increasingly difficult because the corresponding data structures for proximity searches can be difficult to implement.

However, it is not *a priori* obvious that matchlist-based approaches are particularly efficient. We might guess that the use of a point location or range query data structure could result in significant speedups, as some researchers have argued[14]. To date, no direct comparison of the two approaches has been carried out. This paper presents experimental comparisons of the two approaches, as well as a more in-depth analysis of the statistical properties of matchlists. It also presents a direct comparison of depth-first and best-first matching strategies.

2 Branch and Bound Algorithms for Geometric Matching

2.1 An Instance of the Geometric Matching Problem

To simplify the presentation for the purpose of this paper, we will be concrete and examine the geometric matching problem among point sets in two dimensions under translations and rotations; more general formulations can be found in the references[6]. Define a model to be a set of points $M = \{m_1, \ldots, m_r\} \subseteq \mathbb{R}^2$ and an image to be another set of points $B = \{b_1, \ldots, b_s\} \subseteq \mathbb{R}^2$. We consider a bounded set of isometric transformations $\mathbb{T}_{\text{all}}$; that is, the set of transformations $T : \mathbb{R}^2 \to \mathbb{R}^2$ consisting of translations and rotations and parameterize these transformations by a vector $(\Delta x, \Delta y, \alpha)^T \in \mathbb{T}_{\text{all}}$, where $\mathbb{T}_{\text{all}} = [\Delta_{\text{min}}, \Delta_{\text{max}}] \times [\Delta_{\text{min}}, \Delta_{\text{max}}] \times [0, 2\pi) \subseteq \mathbb{R}^3$ is the space of all transformations. We also assume a bounded error notion of quality of match $Q(M, B, \epsilon; T)$ under the Euclidean distance; where it is clear from context, we will omit the dependence of Q on M, B, and/or ϵ. That is, the quality of match assigned to a transformation T is the number of model points the transformation brings within an error bound of ϵ of some image point. If we write $\lfloor \textit{predicate} \rfloor$ for the standard indicator function, which assumes the value 1 if the *predicate* is true, 0 otherwise, we can write this quality of match function as

$$Q(T) = \sum_{m \in M} \max_{b \in B} \lfloor \, ||T(m) - b|| < \epsilon \, \rfloor. \tag{1}$$

The task of a geometric matching algorithm is to find a transformation T_{max} (usually not unique) that maximizes the quality of match for a given M, B, and ϵ:

$$T_{\text{max}}(M, B, \epsilon) = \arg \max_{T \in \mathbb{T}_{\text{all}}} Q(T; M, B, \epsilon) \tag{2}$$

This formulation can be easily generalized to other transformation spaces and other quality of match measures, most importantly, maximum likelihood matches under a Gaussian error model; for details, see [6].

2.2 Branch and Bound Algorithms

Let us now turn to a description of a branch and bound algorithm for geometric matching. As input, the algorithm is given a set of model points $M = \{m_1, \ldots, m_r\} \subseteq \mathbb{R}^2$ and a set of image points $B = \{b_1, \ldots, b_s\} \subseteq \mathbb{R}^2$. In terms of these points, we define a quality of match function $Q(T)$ as in Equation 1, as well as an upper bound $\hat{Q}(\mathbb{T}) \geq \max_{T \in \mathbb{T}} Q(T)$ (the details of how this upper bound is computed will be discussed below). For the purposes of this paper, we consider the space of transformations of $\mathbb{R}^2$ consisting of translations and rotations. Here, x_0, x_1, y_0, and y_1 define the minimal and maximal translations the search algorithm is going to consider.

Using these preliminaries, can organize the search for a globally optimal solution to Equation 2 as follows:

Algorithm 1

1. The algorithm maintains a priority queue of search states. When two search states have the same priority, the state with the lower depth in the search tree is preferred.
2. Initially, we add a node to the priority queue representing the space of all transformations, $\mathbb{T}_{\text{all}}$.
3. Each search state is associated with a subregion of transformation space $\mathbb{T}_k \subseteq \mathbb{T}_{\text{all}}$. It is further associated with an upper bound $\forall T \in \mathbb{T}_k : \; \hat{Q}_k = \hat{Q}(\mathbb{T}_k) \geq Q(T)$; the upper bound serves as the priority of the state.
4. The algorithm removes the state with the highest upper bound from the priority queue.
5. Pick some transformation $T \in \mathbb{T}_k$; if $Q(T) = \hat{Q}(\mathbb{T}_k)$, terminate the search and return T as a solution.
6. Otherwise, we split the region $\mathbb{T}_k$ into two disjoint subregions $\mathbb{T}_{2k}$ and $\mathbb{T}_{2k+1}$ along its largest dimension and insert these subregions back into the priority queue

This type of algorithm is known as a *branch and bound* algorithm in the global optimization and operations research literature. The particular variant described in Algorithm 1 is a best-first search. In order to obtain a depth-first search, we make a few modifications:

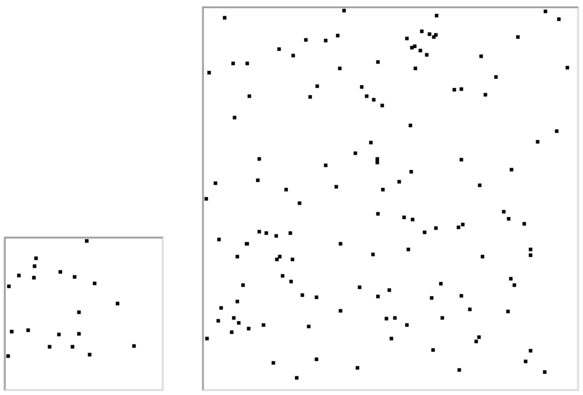

Fig. 1. A random instance of geometric matching problems used for various empirical performance measurements throughout the paper. The model is described in Section 3.1.

– Instead of using $\hat{Q}(T)$ as the priority in the priority queue, we use the depth of the search, k, as the priority.
– Instead of terminating in Step 5, we record the new solution if it is better than the previously best solution but do not terminate.
– In Step 6 we do not enqueue subregions if their upper bound is already worse the the best solution found so far.
– The algorithm terminates when the search queue is empty.

A depth-first search can also be implemented without use of a priority queue[4], but the priority-queue based algorithm is presented here because it allows us to use essentially the same implementation for comparing the performance of depth-first and best-first strategies.

2.3 Geometric Computation of Upper Bounds

In the discussion so far, there has been no mention of how to compute the upper bounds $\hat{Q}(\mathbb{T})$. Under a bounded error model, computation of this upper bound amounts to computing, for each model feature, whether some image feature might fall within the given error bound of a transformed model feature. In different words, we bound each term in Equation 1.

The algorithm described in [4] takes advantage of special properties of linear error bounds under translation, rotation and scale [1] or affine transformations that allow this test to be carried out easily and efficiently using a small number of dot products, resulting in a tight upper bound. For other kinds of transformations and error measures, a tight upper bound is difficult to compute, but we do not actually require a tight upper bound for the algorithm to converge. Therefore, branch-and-bound algorithms can be formulated using more general upper bounds (e.g., [9]) that are easier to derive.

Briefly, we start by determining a bound on the location of each model point m_j under any transformation $T \in \mathbb{T}$. This bound can be expressed as any convenient geometric region, for example a bounding rectangle or a bounding circle in the image plane. For the purpose of exposition, let us assume that we express this bound as a circular error bound of diameter δ around some point $T_c(m)$ for some $T_c \in \mathbb{T}$ (preferably "central" in that region). T_c and δ can, in general, be dependent on m. That is, we choose T_c and δ such that $\forall T \in \mathbb{T} : \ ||T_c(m) - T(m)|| \leq \delta$. Then it is easy to see (using the triangle inequality) that

$$\forall T \in \mathbb{T}, b \in \mathbb{R}^2 : \ ||T(m) - b|| \leq ||T_c(m) - b|| + \delta \tag{3}$$

Finally, we can use this to bound $Q(\mathbb{T})$. Let T be any transformation in some subregion $\mathbb{T}$ of transformation space. Then, by bounding the terms of the sum individually using Equation 3:

$$Q(\mathbb{T}) = \max_{T \in \mathbb{T}} \sum_{m \in M} \max_{b \in B} \lfloor \, ||T(m) - b|| < \epsilon \, \rfloor \tag{4}$$

$$\leq \sum_{m \in M} \max_{b \in B} \lfloor \, ||T_c(m) - b|| < \epsilon + \delta \, \rfloor \tag{5}$$

Given this inequality, we can compute $\hat{Q}(\mathbb{T})$ fairly simply by computing T_c and δ and evaluating the summation and maximization over all pairs of model and image features.

In order to apply the algorithms presented in this paper, we need to obtain an upper bound $\hat{Q}(\mathbb{T}) \geq \max_{T \in \mathbb{T}} Q(T)$. We have generally expressed the connection between the two by deriving, given a set of transformations $\mathbb{T}$, a representative transformation T_c and an error bound δ such that we can choose $\hat{Q}(\mathbb{T}; \epsilon) = Q(T_c; \epsilon + \delta)$. Furthermore, the only sets of transformations $\mathbb{T}$ encountered by the algorithm are axis aligned hyperrectangles in transformation space $\mathbb{T}_{\text{all}}$.

A different way of looking at this is in terms of the region that a model point m traces out under transformations in the set $\mathbb{T}$, the *swept area* [9]. Formally, this set is written as as $\{Tm : T \in \mathbb{T}\} \in \mathbb{R}^2$. We are bounding this set by the circle centered at T_c with radius $\epsilon + \delta$.

2.4 Efficient Evaluation of Upper Bounds

We could, of course, evaluate the upper bound in Equation 4 directly. This requires $|B|\,|M|$ evaluations at each node in the search tree, a fairly expensive proposition. A simple optimization is to speed up the search for matching image points, that is, points b that fall within a distance of $\epsilon + \delta$ by using a point location data structure [12]. Then, the evaluation of Equation 4 can be carried out in $|M|$ steps. A particularly simple point location data structure is the distance transform, which has also been applied in branch-and-bound type geometric matching algorithms [3,10].

An alternative approach is based on *matchlists* [4]. In a matchlist-based approach, we associate with each state in the priority queue in Algorithm 1, not just a region in transformation space and a bound, but also a list of all the correspondences between model points and image features that are consistent with matching under the error bounds and the transformations represented by that region. As we subdivide regions in transformation space, we only need to examine correspondences that are consistent with the parent region.

A matchlist approach is simpler to implement because it does not require the introduction of a separate point location data structure. For example, a complete implementation of the branch and bound algorithm used in this paper is about 900 lines of C++ code, including data structures; the ANN library[11] is about 6900 lines of code.

In fact, the difficulty of implementing point-location based approaches for features or error bounds that are more complex than simple point features has driven the choice of benchmark problems used in this paper: by using simple point features in the benchmarks, we can use standard, highly-optimized implementations of point-location data structures like the ANN library. Using other feature types would have forced the use of heuristics or much more complex data structures in the point location-based implementations, putting point location based approaches at a disadvantage and complicating the interpretation of the results. Higher-dimensional transformation spaces (e.g., including scale changes or affine transformations) were not used in these experiments because matching unoriented unlabeled point features optimally in higher dimensional transformation spaces is costly using any known approach and because such transformation spaces often require more complex error models.

3 Experimental Results

To compare the performance of matchlist and point-location based approaches, this paper reports two set of experiments: a direct comparison of matchlist and point-location methods, and results from a more detailed instrumentation of a matchlist-based matching algorithm.

3.1 Datasets

In order to present actual results on running times and performance, we need to pick a space of geometric matching problems over which to measure and compare the performance of different matching algorithms. For this paper, we will be using two sets of data. The first consists of randomly constructed matching problems in which images and models consist of uniformly distributed points in subsets of $\mathbb{R}^2$. This model has the advantage of being easy to reproduce and easy to understand; it also has been used previously in the literature [12]. The second dataset consists of images from the standard COIL-20[13] database.

For the simulated datasets, each random problem instance is generated as follows. Each model consists of 20 points uniformly sampled from the rectangle $[-100, 100] \times [-100, 100]$. Each image consists of 10 points randomly selected from the model, rotated by a random angle in the interval $[0, 2\pi)$ and translated by a random translation drawn from $[0, 512] \times [0, 512]$. Each such model point is additionally perturbed by an error vector randomly and uniformly selected from the set $\{v \in \mathbb{R}^2 : ||v|| < 5\}$; this represents a 5% error on the location of model features in the image. Additionally, each image contains a random number (between 10 and 160) of background points, uniformly selected from the rectangle $[0, 512] \times [0, 512]$. The image points (both model and background points) are randomly permuted before being used as input to the matching algorithms.

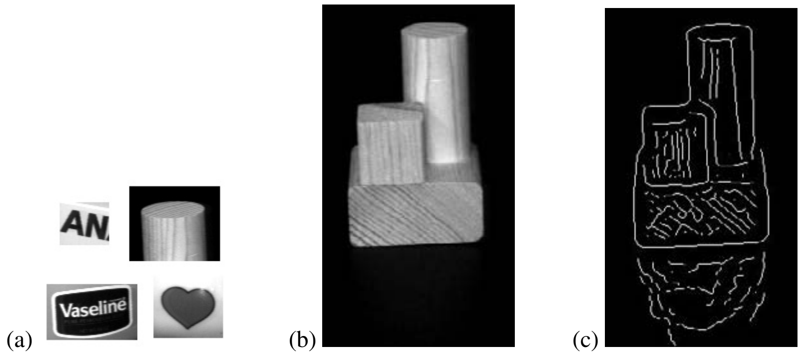

Fig. 2. Sample models and images from the COIL-20 database used in the performance measurements. Shown are (a) four models matched against the images, (b) a sample image, (c) edges extracted from the sample image and used by the matching algorithm.

In the results reported below, measurements are averages over 200 problem instances. Simulations were carried out on a 400MHz Pentium laptop.

There are a wide variety of possible choices for parameters in these kinds of simulations. The choices made above appear to represent the qualitative behavior of the algorithms reasonably well within a statistical model that is easy to explain and reproduce. No claims are made that this simple uniform model represents accurately the statistics of all real geometric matching problems, but some geometric matching problems appear to have distributions that are fairly uniform

For experiments with the COIL-20 database, edges were extracted using a Canny edge detector, the resulting edges were polygonized, and samples uniformly at 4 pixels. This gives rise to images consisting of between 82 and 283 edge samples. No gradient information was used in the matches (see the Discussion below for the rationale). As models, four object parts were selected manually and subjected to the same feature extraction process. Examples of these images are shown in Figure 2. For the benchmarks, each of the four parts was matched against the same set of 50 randomly selected images, resulting in a total of 200 runs of the algorithms.

3.2 Direct Comparison of Point Location and Matchlists Methods

To determine the relative performance of a point location approach and a matchlist approach, a branch and bound algorithm was implemented with two alternative upper bound functions. The first upper bound computation uses the point location approach described by Mount *et al.*[12]. The point location data structure (ANN) used in that implementation is publically available [11] and was used as the point location data structure for point-location-based upper bound computations in the experiments. As an additional optimization, the point location-based implementation took advantage of matchlists by eliminating non-matching model features for further consideration in child nodes in the search tree. It was verified that this resulted in some speed up compared to a naive implementation that requires time proportional to $|B|$.

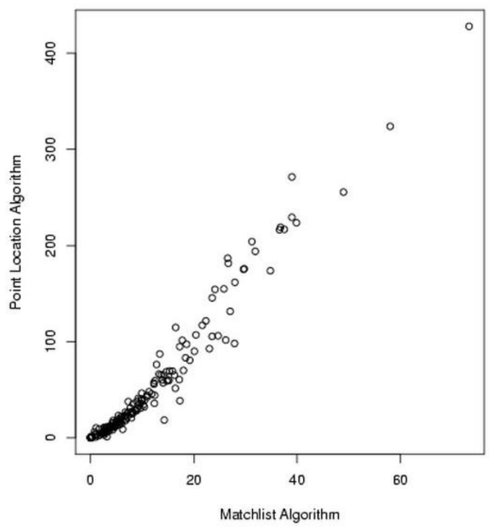

Fig. 3. Relative performance of the matchlist method compared to the point location method. Both axes show running times in seconds. Note the different scales of the axes. The slope of the curve is 3.7, meaning that the matchlist method runs on average 3.7 times faster than the point location method.

On the randomly generated problem instances, using the matchlist based approach resulted in a *speedup* of a factor of 2.43 and 3.75 (for 120 and 10 points of clutter, respectively) compared to the point location approach using the ANN library. These results are reproduced by the experiments on the COIL-20 data set. Figure 3 shows a scatter plot of running times of the matchlist-based method compared to the point location-based method (note the different scales of the axes). Each datapoint represents a run of each algorithm on the same pair of model and image. The slope of the line is 3.7, meaning that the matchlist based approach runs on average 3.7 times faster than the point location method on this dataset.

3.3 Statistical Properties of Matchlists

These results are perhaps surprising at first glance. In fact, the initial matchlist in a matchlist based approach is of size $|M|\,|B|$, and the initial few subdivisions do not reduce the matchlist at all. Only once $\mathbb{T}_k$ becomes small enough so that the uncertainty $\epsilon+\delta$ of the projected model features will become sufficiently small so as not to include all image features, will the match list be reduced below that size. The efficiency of a matchlist based approach to evaluating $\hat{Q}(\mathbb{T})$ then depends on the size of the average match list that occurs during the search. If most search states are expanded with small matchlists, a matchlist based approach is efficient. If most search states were expanded with large matchlists, the matchlist based approach would be hopelessly inefficient compared to

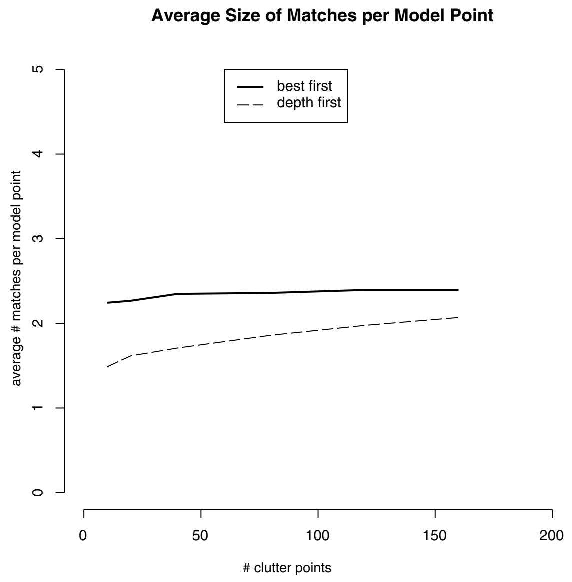

Fig. 4. Size of the average matchlist. The matching problem used is as given in Section 3.1. As in the other results, the x-axis gives the amount of clutter; the y-axis shows the number of correspondences on the average matchlist.

approaches based on point location. To address this question, the branch and bound matching algorithm was instrumented to keep statistics about the size of matchlists at various depths in the search tree.

The question of the size of matchlists can then be easily answered by collecting statistics from actual runs of the algorithm. For the problem defined in Section 3.1, these results are shown in Figure 4. As we can see, the average size of the matchlist $\bar{l}$ is a little over two, meaning that there are, on average, a little over two image features on the matchlist for each model feature. This makes it very difficult to outperform the matchlist based approach using a point location data structure, since the lookup performed by the point location data structure would have to take less time than the time for two computations of $||m' - b||$ (plus a small amount of bookeeping overhead). In fact, this result was already suggested early on by execution profiling of implementations in [4], which showed that the system spent roughly equal amounts of time in the computation of $T_c(m)$ and $||m' - b||$, making further reductions in the time spent on distance computations of limited benefit.

It is instructive to look at these statistics in a little more detail. Figure 5 contains two types of curves. The solid curve indicates what fraction of the total number of search states that are expanded by the algorithm are expanded below a certain depth. We see that only a small fraction of the states is expanded below depth 12. The broken line indicates the average size of the matchlist for search states at a certain depth. It shows that the average size of the matchlist becomes small below the depth at which most states occur during the search.

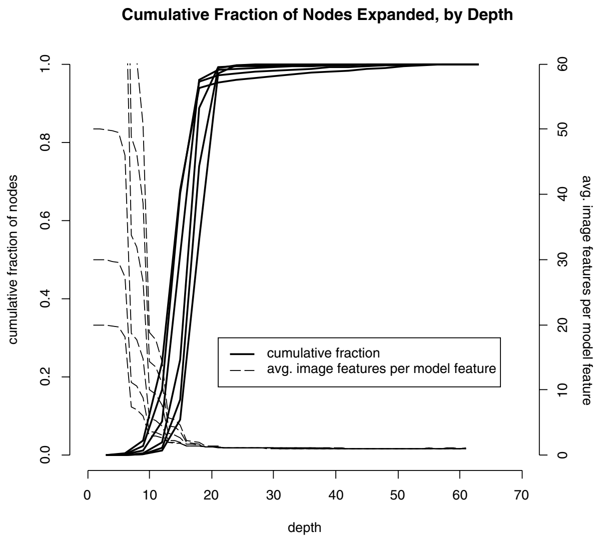

Fig. 5. Fraction of nodes expanded at different depths, and average number of image features matching each model feature at different depths. These results are for the problem described in Section 3.1; the different curves are at a clutter of 10, 20, 40, 80, 120, and 160. The solid curve indicates what fraction of the total number of search states that are expanded by the algorithm are expanded below a certain depth. The broken line indicates the average size of the matchlist for search states at a certain depth.

There is a simple heuristic explanation for this behavior: the algorithm must keep subdividing regions in transformation space recursively until it can start pruning the search tree, leading to an initial exponential growth in the number of nodes, However, pruning is possible only once the matchlist has shrunk significantly. Therefore, exponential growth of the number of nodes will tend to occur somewhat beyond the point where the matchlist has shrunk, resulting in the observed complexity. The results in Figure 6 suggest a nearly perfectly exponential growth up to a certain depth, followed by the onset of pruning.

3.4 Best First vs. Depth First

So far, we have remained within a traditional branch-and-bound framework, by expanding the most promising search node, no matter at what depth in the search tree it may occur. We might call this a "best first" search algorithm. An alternative approach is to use a "depth first" approach. A depth-first search can be implemented by using search depth instead of the upper bound $\hat{Q}$ as the priority in the priority queue, as described in Section 2.2.

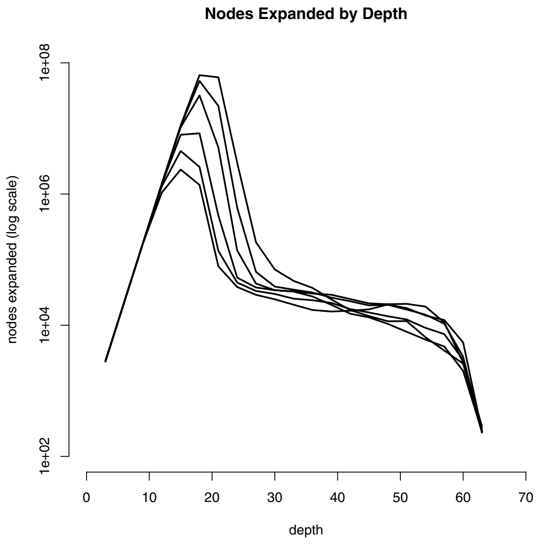

Fig. 6. Number of nodes expanded at different depths (logarithmic scale). Notice the initial exponential growth. These results are for the problem described in Section 3.1; the different curves are at a clutter of 10, 20, 40, 80, 120, and 160.

With this modification, we are no longer guaranteed anymore that the first solution that satisfies our acceptance criteria in Step 5 is, in fact optimal. The algorithm therefore needs to continue searching until the complete search tree has been examined.

At first sight, this may seem to result in a much larger number of node expansions and therefore might be considerably less efficient. What makes this algorithm practical is the observation that when the algorithm has found a candidate solution in Step 5, we need not expand further any states that we encounter whose upper bound estimate is less than, or equal to, the quality of the best solution we have found so far. In practice, this means that the additional cost in terms of runtime of a depth-first search strategy is often modest. Experimental results demonstrating this point are shown in Figure 7. Furthermore, experimentally, the relative overhead becomes smaller the larger the geometric matching problem becomes, meaning that larger overhead is paid on small problems which are already solved quickly.

Why might we want to adopt a depth-first search strategy? Because the amount of memory it requires is very much smaller than that of a best first strategy. Over a common range of parameters explored in our experiments, the depth first search approach requires between 1/50 and 1/200 the amount of memory of the best first strategy (Figure 8). On systems with limited amounts of memory, a depth-first approach may be the only feasible approach for implementing branch-and-bound algorithms. Even on general purpose workstations with large amounts of memory, the moderate increase in running times associated with a depth first search may be justifiable in return for greatly reduced memory requirements when implementing these algorithms on embedded systems. A

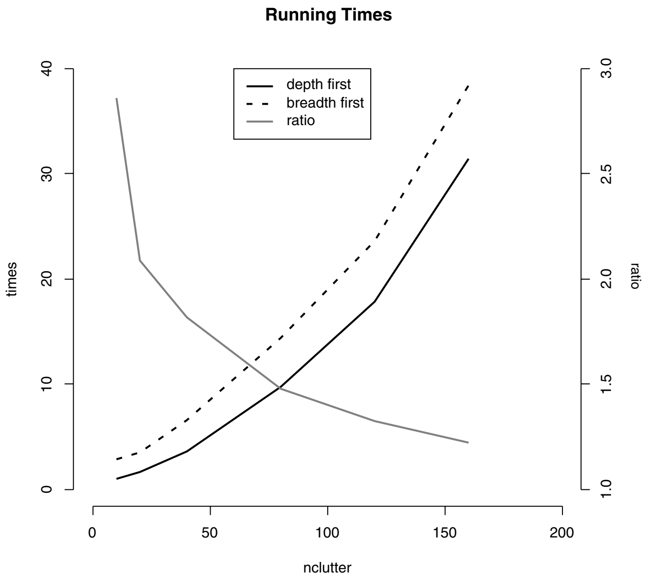

Fig. 7. Running times of the depth first and best first search strategies compared. The problem is as described in Section 3.1, with the number of clutter points given by the x-axis. The left y-axis shows the running time in seconds, the right y axis shows the ratio of the depth-first running times to the best-first running times.

depth-first approach also makes it easier to tolerate the larger memory footprint associated with splitting regions $\mathbb{T}$ into 2^n subregions, rather than using binary splits in Step 6 in Algorithm 1, resulting in performance that can equal or surpass that of a best-first search with binary splits (data not shown).

4 Discussion

This paper has compared alternative implementations of two fundamental aspects of branch-and-bound algorithms for geometric matching. The first is the question of how to compute upper bounds on the quality of match function $Q(T)$ over a given set of transformations. The second is the question of which order to expand search nodes in.

The experimental results presented in this paper suggest that on common and non-trivial problems, namely uniformly distributed model and image features under bounded error, a matchlist-based approach outperforms a point-location based approach. It was also demonstrated that tese results also carry over to matching experiments with real image data extracted from the COIL-20 database.

Of course, results from empirical performance measurements on two benchmark problems does not guarantee that that the advantage of matchlist based implementations holds over all possible problem domains. However, the additional results presented in Section 3.3 show a simple way of determining whether a matchlist-based approach is efficient: by measuring the average number of image features matching each model

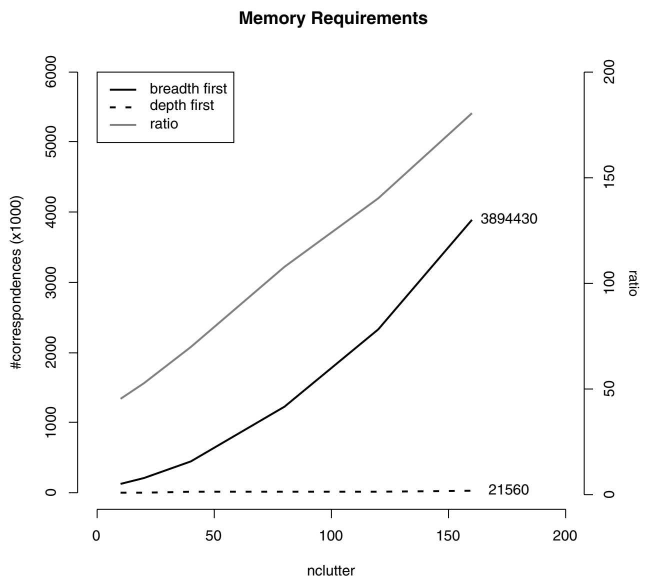

Fig. 8. Comparison of the memory requirements of the depth first and best first search strategies. The problem is as described in Section 3.1, with the number of clutter points given by the x-axis. The left y-axis shows the number of correspondences (in thousands) that need to be retained in the search queue (proportional to the memory requirements) seconds, the right y axis shows the ratio of the best-first to the depth-first memory requirements.

feature. This information can be obtained by a simple and efficient instrumentation of the matchlist-based matching code, or even from a standard execution profile. This allows us to use a simple matchlist-based approach first. Only when an execution profile suggests that a matchlist-based algorithm spends a large amount of time in, or a large number of invocations of, feature distance computations compared to geometric transformations of model features, would we need to consider other implementation strategies.

The empirical fact that matchlist-based approaches are often more efficient than point-location based approaches is good news because point location based methods are more difficult to implement even in the simplest cases (they require an extra data structure), and become increasingly complex for the kinds of feature types and error bounds encountered in real-world matching problems. For example, implementing point location datastructures for features associated with gradient information, extended features like line segments, or error bounds that depend on, and vary with, the transformation (e.g., error bounds that are large perpendicular to the gradient and small parallel to the gradient) requires work that goes significantly beyond traditional k-D tree or trie methods. In contrast, these real-world complexities are easily handled in a matchlist based approach because, using matchlists, an implementor only needs to supply a quality-of-match function comparing an individual model feature that has already undergone transformation with an image feature (and possibly an upper bound function). Another way of looking at matchlist based approaches is that they construct, implicitly during the branch-and-bound search itself, a proximity search data structure that is well-adapted to the needs of the matching problem.

The second set of experimental results deals with the tradeoff between depth-first and best-first (i.e., a priority-queue based approach) search strategies. It is not surprising that a depth-first approach uses much less space than a best-first search, but it has not been clear until now whether the potentially larger number of nodes that needs to be explored in a depth-first approach would put such an approach at a severe disadvantage. The results presented in this paper suggest that the overhead of a depth-first approach is modest and the reduction in memory usage is large. A depth-first approach therefore warrants serious consideration in the implementation of branch-and-bound methods for geometric matching.

Of course, it is impossible to test and compare all the different conditions under which a geometric matching algorithm might be used. The author hopes that other practitioners will see this paper as an incentive for comparing different approaches to the implementation of branch-and-bound algorithms for geometric matching problems.

References

1. Henry S. Baird. *Model-Based Image Matching Using Location.* MIT Press, Cambridge, MA, 1985.
2. D. H. Ballard. Generalized hough transform to detect arbitrary patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(2):111–122, 1981.
3. G Borgefors. Hierarchical chamfer matching: A parametric edge matching algorithm. *PAMI*, 10:849–865, 1988.
4. T. M. Breuel. Fast Recognition using Adaptive Subdivisions of Transformation Space. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 445–451, 1992.
5. T. M. Breuel. Higher-Order Statistics in Object Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 707–708, 1993.
6. T. M. Breuel. Branch-and-bound algorithms for geometric matching problems. Submitted to *Computer Vision and Image Understanding*, 2001.
7. T. M. Breuel. Robust least square baseline finding using a branch and bound algorithm. In *Document Recognition and Retrieval VIII, SPIE, San Jose*, 2001.
8. E. Grimson. *Object Recognition by Computer.* MIT Press, Cambridge, MA, 1990.
9. M. Hagedoorn and R. C. Veltkamp. Reliable and efficient pattern matching using an affine invariant metric. Technical Report RUU-CS-97-33, Dept. of Computing Science, Utrecht University, 1997.
10. D.P. Huttenlocher, G.A. Klanderman, and W.J. Rucklidge. Comparing images using the hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):850–63, 1993.
11. D. M. Mount and S. Arya. ANN: A library for approximate nearest neighbor searching. In *CGC 2nd Annual Fall Workshop on Computational Geometry*, 1997.
12. D.M. Mount, N.S. Netanyahu, and J. Le Moigne. Efficient algorithms for robust feature matching. *Pattern Recognition*, 32(1):17–38, 1999.
13. S. Nene, S. Nayar, and H. Murase. Columbia object image library: Coil. Technical Report CUCS-006-96, Department of Computer Science, Columbia University, 1996.
14. Clark F. Olson. Locating geometric primitives by pruning the parameter space. *Pattern Recognition*, 34(6):1247–1256, June 2001.