Comparing Random Starts Local Search with Key Feature Matching *

J. Ross Beveridge Christopher R. Graves Jim Steinborn
Computer Science Department
Colorado State University
Fort Collins, CO 80523

Abstract

A new variant on key feature object recognition is presented. It is applied to optimal matching problems involving 2D line segment models and data. A single criterion function ranks both key features and complete object model matches. Empirical studies suggest that the key feature algorithm has run times which are dramatically less than a more general random starts local search algorithm. However, they also show the key feature algorithm to be brittle: failing on some apparently simple problems, while local search appears to be robust.

1 Introduction

To find an object, recognition algorithms often first seek small sets of features which predict the object's presence. This ideas is articulated by Roberts [Roberts, 1965] and is the heart of local feature focus [Bolles and Cain, 1982]. It suggests the value of perceptual organization [Lowe, 1985] and is fundamental to the alignment approach [Huttenlocher and Ullman, 1990]. It is also a basic component of Geometric Hashing [Lamdan *et al.*, 1990] and continues to be refined [Olson, 1995].

Here we present a variant upon this general theme which searches for an optimal match by first searching for good matches between triples of object model and image features. For simplicity, we call this our key feature algorithm since the search for a good triple may be thought of as a search for a key feature which determines the rest of the match. The key feature algorithm uses the same criterion function as our local search line matching algorithms [Beveridge *et al,* 1990; Beveridge, 1993; Beveridge *et al,* 1997] and this enables us to report on a set of side-by-side comparisons between the two. Since these two algorithms search the space of possible matches in very different ways, the comparison highlights strengths and weaknesses of each.

*This work was sponsored by the Defense Advanced Research Projects Agency (DARPA) Image Understanding Program under grants DAAH04-93-G-422 and DAAH04-95-1-0447, monitored by the U. S. Army Research Office, and the National Science Foundation under grants CDA-9422007 and IRI-9503366

The greatest difference between the two algorithms is in how they initiate search. The key feature algorithm initiates indepdent searches from each of a set of ranked key features. Local search starts a number of independent trials from randomly selected matches. The key feature algorithm relies on one or more of the key features belonging to the best match. In contrast, local search makes no effort to find consistent initial matches and instead relies upon iterative improvement guided by updated global alignment to move search from the random match to one which is good. Randomness is important to local search because the probability of failing to find a good solution over multiple trials drops exponentially as the number of trials increases. This same random sampling methodology has been used in the RANSAC algorithm to find consistent subsets of object model and image features [Fischler and Bolles, 1987].

We begin by reviewing our formulation of 2D line segment matching as a combinatorial optimization task which has as its goal finding matches such as the one shown in Figure 1. Next the random-starts local search and key feature algorithms are described and their performance is compared on the example just shown as well as on a series of controlled test problems.

2 Optimal Matching

Let M be a model of 2D line segments and D a set of segments representing image data. A match is a many-to-many correspondence mapping c where:

$$c \in C \qquad C = 2^S \qquad S \subseteq M \times D \tag{1}$$

An objective function E defines the best match c^*.

$$E(c^*) \leq E(c) \quad \forall c \in C \tag{2}$$

The match error E has two terms [Beveridge, 1993]:

$$E(c) = E_{\text{fit}}(c) + E_{\text{om}}(c) \tag{3}$$

$$= \left(\frac{1}{\sigma^2}\right) E_{\text{Res}}(c) + E_{\text{om}}(c) \tag{4}$$

$E_{\text{fit}}(c)$ measures how well the model fits the data. $E_{0m}(c)$ measures how much of the model is omitted from the match.

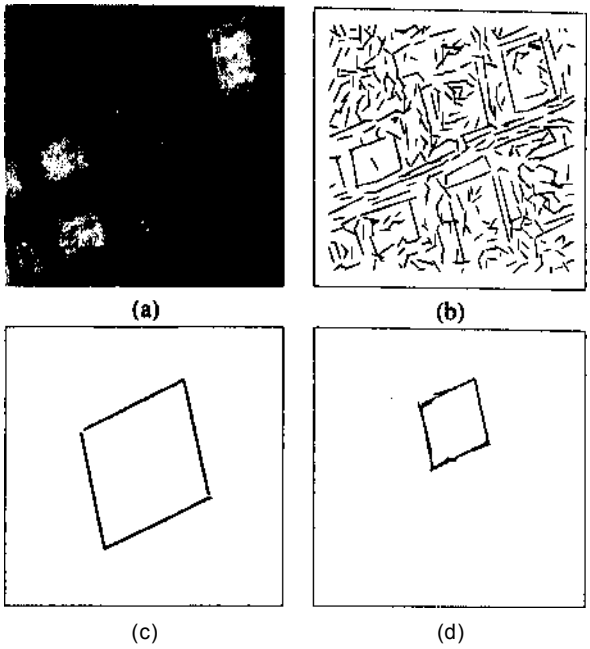

Figure 1: Optimal match to fragmented data, a) aerial photograph, b) segments [Burns *et al.,* 1986], c) model, d) best match.

To evaluate *E,* a 2D similarity transformation best fitting the model to the data is computed. The fitting criterion is a weighted sum of integrated perpendicular distance between infinitely extended model lines and their corresponding data line segments: $E_{Res}(c)$ is a normalized function of the residual error after fitting. The best fit for any c, neglecting underconstrained cases, is computed by solving a quadratic polynomial. *Eom(c)* is computed by transforming the model to the best-fit configuration and measuring how well the data covers the model. The weighting term *a* dictates how far a data segment can be from a model segment and still be included in an optimal match. If, for instance, $\sigma = 3$, then the data may be up to 3 pixels from the model.

3 Random Starts Local Search

Perhaps the simplest algorithm is steepest-descent on a 'Hamming-distance-T neighborhood. This is so named because any correspondence mapping *c* may be represented by a bitstring of length n, where $n \in |S|$. A 'I' *V* in position j of the bitstring indicates that the jth pair in the set 5 is part of the match c. The *n* neighbors of c are generated by successively toggling each bit. Hence, the neighborhood contains all matches created by either 1) adding a single pair of model-data features not already in the match or 2) removing a single pair currently in the match.

Steepest-descent local search using this neighborhood computes *E* for all n neighbors of the current match c, and moves to the neighbor yielding the greatest improvement: the greatest drop in *E.* Search terminates at a local optimum when no neighbor is better than the current match.

Because local search often becomes stuck at undesirable local optima, it is common practice to run multiple trials. Each trial is started from a randomly chosen initial match c_i. The random selection of c_i is biased to choose, on average, λ data segments for each model segment. Specifically, let h_m be the number of pairs in *S* which contain a model segment TO. Each of these pairs is included in c_i with independent probability $\frac{\lambda}{h}$. Our experience suggests $\lambda = 4$ is a good choice, thus binding on average 4 data segments to each model segment.

Over *t* trials, the probability of failing to find a good match drops as an exponential function *oft.* Let P_s be the probability of finding a good solution on a single trial. The probability of failing to find a good match in *t* trials is:

$$Q_f = (1 - P_s)^t \tag{5}$$

For geometric models, there is often a well defined and known best solution and we can characterize algorithm performance in terms of the probability P_9 of finding the best match in a given trial. From P_8, the required number of trials t_9 needed to solve a particular problem to a preset level of confidence Q_9 may be derived from equation 5:

$$t_s = \lceil \log_{P_f} Q_f \rceil \quad Q_f = 1 - Q_s \quad P_f = 1 - P_s \tag{6}$$

4 Key Feature Matching

The local search algorithm just presented can be turned into a key feature matching algorithm by initializing search from *k* carefully selected key feature matches. Let *F* be the set of key features, and let a specific key feature f_i be defined as follows:

$$f_i \in F \qquad f_i = \{s_{i1}, s_{i2}, s_{i3}\} \qquad s_{ij} \in S \tag{7}$$

The local search neighborhood is modified to consider only the addition of pairs. To understand this simplification, consider local search initiated from a key feature f_i^* fully contained within the optimal match:

$$f_i^* = \{s_{i1}^*, s_{i2}^*, s_{i3}^*\} \qquad s_{ij}^* \in c^* \tag{8}$$

Simply by adding pairs local search will typically arrive at the best match c*. Search is further simplified by noting that each match to a key feature $f_i \in F$ constrains where the model is placed relative to the data. Thus, search can be carried out in a constrained search space which includes only pairs $s \in S$ consistent with the specific model placement defined by f_i.

As a final step, the top 5 matches found by searching from each of the *k* key features are used to initialize a single trial of random starts local search. This final step can both add and remove pairs of segments and proves to be important in some cases. Without this step, key feature matching can be close to the optimal match and still miss it. In principle, this final pass of local search could allow the key feature algorithm to find

the best match even when no key feature satisfies equation 8. However, as a rule of thumb, the key feature algorithm will fail to find the optimal match when this condition is not met, i.e. no key feature is a subset of the optimal match c*.

4.1 Choosing Key Features

A variety of strategies has been suggested for defining sets of key features. [Lowe, 1985] strongly advocated broad and general perceptual invariants derived from the basic rules of physics and imaging. [Huttenlocher and Ullman, 1990] took a more model-based approach: predicting which features are key in part upon specific object geometry.

However, both Lowe and Huttenlocher separated the task of selecting possible key features from that of matching complete models to image data. In contrast, we use the same criterion function, E, to both rank our selection of possible key features and to evaluate the optimality of final matches.

As in past work, the ability of a key feature to constrain the pose of the object is critical. For 2D line matching, where models can be rotated, translated and scaled, the two obvious choices are pairs of matched segments and triples of matched segments. Triples are more reliable predictors of pose.

4.2 Spatially Proximate Triples

If there are *n* possible pairings between model and data features in the set 5, then there are n^3 possible triples. For typical problems presented below, $n \approx 1{,}000$. It is impractical to enumerate and rank 1,000,000,000 triples; clearly, some filter must be used. We use a very general heuristic: *proximal lines in a model are likely to be proximal in the data*. This idea is by no means new, [Lowe, 1985] suggests this approach and then dismisses it as too unfocused for his purposes.

Filtering by spatial proximity will generate on the order of *n* ranked triples. Model and data segments are analyzed independently to find the nearest neighbors of each. For each model line $m_i \in M$, determine the closest two neighbors m_{i1} and m_{i2} as defined by Euclidean distance δ:

$$\delta(m_i, m_{i1}) \leq \delta(m_i, m_k) \; \forall \, m_k \in M - \{m_i\}$$
$$\delta(m_i, m_{i2}) \leq \delta(m_i, m_k) \; \forall \, m_k \in M - \{m_i, m_{i1}\}$$

Analogous nearest neighbors d_{j1} and d_{j2} for each data line segment $d_j \in D$ are found.

When matching segments *M* to D, each pair of segments $(m_i, d_j) \in S$ form two spatially proximate triples f_1 and f_2:

$$f_1 = ((m_i, d_j), (m_{i1}, d_{j1}), (m_{i2}, d_{j2}))$$
$$f_2 = ((m_i, d_j), (m_{i1}, d_{j2}), (m_{i2}, d_{j1}))$$

Since each of the n pairs of model and data segments in *S* leads to 2 triples, there are *2n* spatially proximate triples in the initial set of key features *F*.

In keeping with the assumption that some key features are better than others, order the set *F* from lowest to highest match error;

$$F = \{f_1, f_2, \ldots f_{2n}\} \tag{9}$$

$$E(f_k) < E(f_l) \quad \text{iff} \quad k < l \tag{10}$$

The question arises, 'How deep into the ranked set *F* should we search?'. Past work has typically assumed that it is not necessary to go very deep into the set *F*. Our experiments take a conservative approach and use the best *n* triples.

5 Comparing Approaches

We present comparative performance data on two test problems suites produced by a Monte Carlo problem generator. The advantage of this data is that it allows for controlled testing under well specified conditions. We also consider performance on the real world problem shown in Figure 1.

The first set of tests uses the six models shown in Figure 2a. To create datasets D, model segments are randomly scaled and placed in the data images. In addition, the model line segments are fragmented, skewed and potentially omitted. Finally, random clutter and structured clutter are added to the data. Structured clutter consists of more highly degraded copies of the same object model. Examples are shown in Figures 2b and 2c. This dataset and results for it are available through our website (`http://www.cs.colostate.edu/~vision/`).

A second set of tests use data images with 50 randomly placed segments. Models are created by randomly drawing 5, 10,15 or 20 of these segments. This dataset tests the reliability of the proximity heuristic as the ratio of model to clutter lines varies.

5.1 Six Corrupted Models

The models shown in Figure 2a exhibit characteristics known to make matching difficult. For example, the Pole is an interesting case for the key feature algorithm because of its simplicity: there exists only one possible model triple for the key feature algorithm to exploit. The Dandelion exhibits a 16 fold near symmetry; model symmetry complicates matching for many well established techniques [Grimson, 1990]. The Leaf presents an example where model and data line segments approximate a curved contour. For this model, a many-to-many mapping between model and image features is needed to account for breakpoints at different positions along the curve.

A Monte Carlo simulator produces corrupted image data. The simulator rotates, translates and scales the model so placement and size are unknown. Model segments are also fragmented and skewed. In 24 of the problems, 0,10,20 and 30 additional clutter segments are randomly placed about the image. A sampling of this data is shown in Figure 2b. In the other 24 problems, 0, 1, 2 and 3 additional corrupted model instances are added. A sampling of this data is shown in Figure 2c.

The steepest-descent local search algorithm has been run for 1,000 trials on all 48 problems. The key feature algorithm has been run using the best n triples. The resulting run times and quality of results are indicated in Figure 3a. Problem instances are grouped along the *x* axis by model type, with run

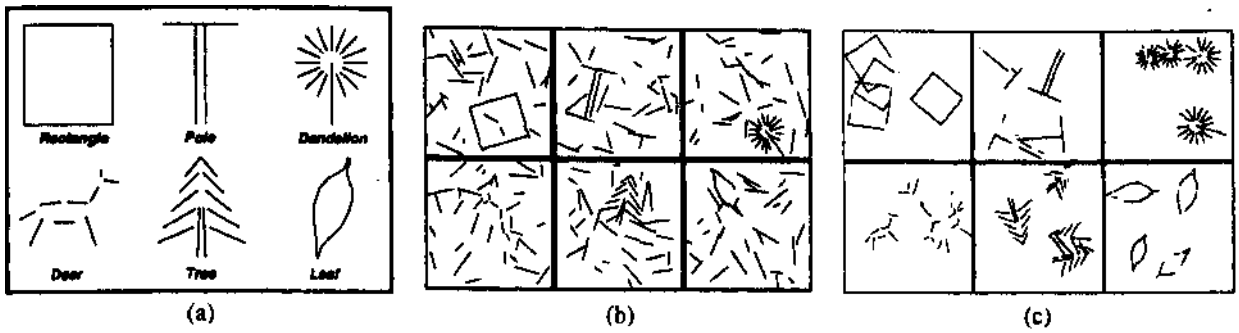

Figure 2: Models and data for controlled tests, a) Six stick figure models, b) Random clutter test data, c) Multiple instances test data.

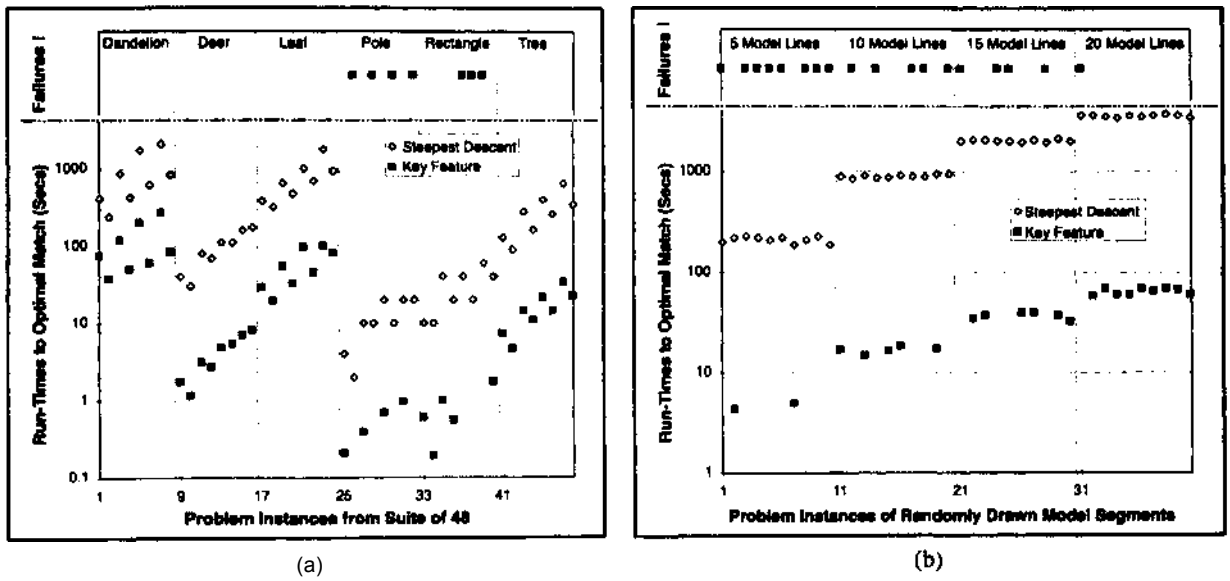

Figure 3: Relative algorithm performance, a) The 48 problems using the six models shown in Figure 2a, b) 40 problems using randomly selected models.

time plotted on the *y* axis. Those problems where key features failed to find the optimal match are indicated in the separated band at the top of the plot.

Two critical types of information are conveyed by this plot. First, over 1,000 trials, the random starts algorithm never failed to find the optimal match at least once. Conversely, the key feature algorithm failed in 7 out of the 48 cases. Second, the key feature algorithm is dramatically more efficient in terms of run time, typically taking less than l/10th of the time required by the random starts local search algorithm.

The key feature algorithm fails on the simpler object models: the pole and rectangle [1]. Two factors may explain these failures. First, simulator fragments model line segments with a fixed probability per unit length. Hence, longer segments fragment more. The rectangle and pole have the longest individual segments, and therefore the highest degree of fragmentation. Too much fragmentation prevents the proximal triples algorithm from finding triples which participate in the best match. The second factor is that there are fewer total features on the simpler models, further reducing the opportunities for the algorithm to find a good key feature.

The statement that the local search algorithm did not fail requires some elaboration. Based upon the 1,000 trials, the probability of success P_s has been estimated for each individual problem. Using these estimates, it is more proper to say that the required number of trials to find the best match with 95% confidence, t_s from equation 6, never exceeds 1,000.

Actually, far fewer than 1,000 trials are required for most of the problems in the test suite. Less than 100 trials are required for 33 out of the 48 problems, and only 3 problems require more than 500 trials. Likewise for the key feature algorithm, it is typically not necessary to consider all *n* triples.

[1]On symmetric models such as the rectangle and building in Figure 1, symmetric matches are treated as equivalent.

The run times shown in Figure 3a are intentionally conservative, making no assumption that there is *a priori* knowledge of how hard a specific problem is to solve.

Looking at the actual work needed per problem, the key feature algorithm finds the best match from the first ranked triple in 26 out of the 48 cases. There are 7 cases where a triple in the top 50 leads directly to the best match. Finally, in the 8 remaining cases, simply running the steepest-descent algorithm where only pairs are added misses the best match. However, in these cases, the final application of random starts local search locks onto the globally best match. This is counted as a success for the key feature algorithm.

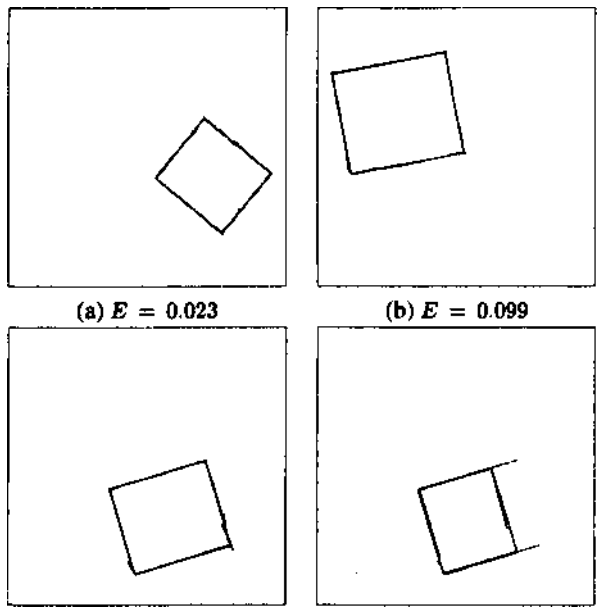

Figure 4: Examples of the key feature algorithm missing the best match, a) best match, b) best found by key feature, c) best match, d) best found by key feature.

It is interesting to observe how the key feature algorithm is lead astray. Figure 4a shows the optimal match found by random starts local search for the three rectangles problem. Figure 4b shows the best match found by the key feature algorithm. This is the 2nd best match found by the local search algorithm. Figure 4c shows the optimal match found by random starts local search for the case of 20 random clutter lines. Figure 4d shows the best match found by the key feature algorithm: again, it is the 2nd best match found by local search. This case is interesting because the coincidently placed long clutter line creates a 'false' rectangle which traps the key feature algorithm.

5.2 Randomly Placed Lines

The probability that the key feature algorithm will succeed depends upon the likelihood that a triple *f in F* belongs to the best match c*. This likelihood, in turn, depends upon the probability that the two segments closest to a data segment coincide with the two closest segments in the model. As the results above suggest, this is a good but not perfect heuristic.

One way to think about this likelihood is to consider how the ratio of clutter to model features varies as more or fewer segments are drawn from a fixed size set of data segments. To illustrate, a set of 40 test problems has been generated, each with 50 randomly placed non-intersecting line segments. Ten distinct sets of 50 random segments are used, and models are created by randomly selecting 5,10,15 or 20 from each set.

The results for these 40 problems are shown in Figure 3b. They support the hypothesis that a higher clutter ratio causes the key feature algorithm to become unreliable. The key feature algorithm fails on 8 out of 10 of the 5-segment models, while only failing on 1 out of 10 for the 20-segment models.

Because the local search algorithm does not depend upon key features, we do not expect it to have difficulty with these problems. Figure 3b shows that this is in fact the case, with local search solving all 40 problems reliably in less than 1,000 trials. Moreover, it may surprise some readers to know that the average t_a for 95% confidence decreases as these models get larger. For the problem instances with models of 5 segments, $\bar{t}_a = 122.3$, while for the models of 20 segments $\bar{t}_a = 74.2$. A *t-test* on this data indicates that the drop is statistically significant: $t = 5.03$ and $p \leq 0.001$. While far fewer than 1,000 trials are actually needed, for the sake of consistency, the times reported in Figure 3b are all based upon 1,000 trials.

5.3 A Real World Example

It should now be apparent that the key feature algorithm has profound computational advantages over random starts local search. When it succeeds, it requires one to two orders of magnitude less computation. Real world examples can be found where it works well, including many of the fairly clean 2D line matching problems common in the literature.

Let us now consider a very difficult problem, the building shown in Figure 1. There are 4 model line segments and 443 data segments, generating 1,772 possible pairs of segments. More important in terms of problem difficulty, there are multiple instances of buildings interacting with other buildings and roads, creating a combinatorial explosion of possible partial matches. Also, the globally best match[2] is highly fragmented and must be pieced together by a search algorithm before it appears more attractive than many of the other more obvious partial rectangles.

To fully test random starts local search on this problem, 10,000 trials have been run. The best match is found in only 12 of these trials: the estimated probability of success $P_s =$

[2]Strictly speaking, we do not know the match shown in Figure 1d to be the global optimum. However, through extensive study of this problem and our ability to eye-ball the results, we are relatively certain it is best.

0.0012 is tiny. Required trials t_s = 2,494 and average time to run a trial is 24 seconds. Consequently, local search requires nearly 18 hours to reliably solve this problem.

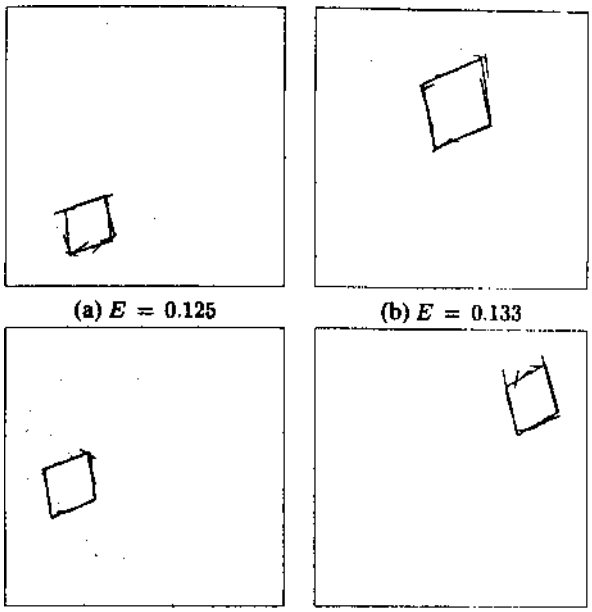

Figure 5: Second through fifth ranked matches for aerial image, a) second best and match found using key features, b) third best, c) fourth best, d) fifth best.

While random starts local search has allowed us to find this best match, 18 hours is far too much time for any practical online system to spend looking for one building. In contrast, the key feature algorithm is very fast, completing in a matter of minutes. However, it fails to find the best match. Instead, it finds the match shown in Figure 5a. This is not totally uninteresting, since according to the local search algorithm, this is the second best match in the image. However, it has a markedly larger match error: 0.125 as opposed to 0.101 for the global optimum. Local search also, in some sense, finds two other buildings: Figures 5c and 5d show the fourth and fifth ranked matches. Figure5b shows that the third ranked match is a variant on the global optimum, sharing many of the same features.

6 Conclusion

We have presented a variant upon key feature matching. It operates within an optimal matching framework and uses the same criterion function to rank both key features and final object model matches. On controlled problem sets, it has been shown that the underlying proximity heuristic yields key features which allow an algorithm to quickly find many but not all optimal matches. In an example of the cost versus generality trade-off typical in Artificial Intelligence, random starts local search is shown to be both more robust and more computationally demanding.

References

[Beveridge *et al,* 1990] J. Ross Beveridge, Rich Weiss, and Edward M. Riseman. Combinatorial Optimization Applied to Variable Scale 2D Model Matching. In *Proceedings of the IEEE International Conference on Pattern Recognition 1990, Atlantic City,* pages 18-23. IEEE, June 1990.

[Beveridge, 1993] J. Ross Beveridge. *Local Search Algorithms for Geometric Object Recognition: Optimal Correspondence and Pose.* PhD thesis, University of Massachusetts at Amherst, May 1993.

[Beveridge *et al.,* 1997] J. Ross Beveridge, Edward M. Riseman and Christopher R. Graves. How Easy is Matching 2D Line Models Using Local Search? *IEEE Trans, on Pattern Analysis and Machine Intelligence,* page (to appear), 1997.

[Bolles and Cain, 1982] R. C Bolles and R. A. Cain. Recognizing and Locating Partially Visible Objects: The Local-Feature-Focus Method. *International Journal of Robotics Research,* I(3):57-82, 1982.

[Burns et al., 1986] J. B. Burns, A. R. Hanson, and E. M. Riseman. Extracting straight lines. *IEEE Trans, on Pattern Analysis and Machine Intelligence,* PAMI-8(4):425 - 456, July 1986.

[Fischler and Bolles, 1987] Martin A. Fischler and Robert C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. In Martin A. Fischler and Oscar Firchein, editors, *Readings in Computer Vision,* pages 726 - 740. Morgan Kaufmann, Los Altos, CA, 1987.

[Grimson, 1990] W. Eric L. Grimson. *Object Recognition by Computer: The Role of Geometric Constraints.* MIT Press, Cambridge, MA, 1990.

[Huttenlocher and Ullman, 1990] Daniel P. Huttenlocherand Shimon Ullman. Recognizing Solid Objects by Alignment with an Image. *International Journal of Computer Vision,* 5(2): 195 - 212, November 1990.

[Lamdan *et al,* 1990] Yehezkel Lamdan, Jacob T. Schwartz, and Haim J. Wolfson. Affine invariant model-based object recognition. *IEEE Transactions on Robotics and Automation,* 6(5):578 - 589, October 1990.

[Lowe, 1985] David G. Lowe. *Perceptual Organization and Visual Recognition.* Kluwer Academic Publishers, 1985.

[Olson, 1995] C.F. Olson. On the speed and accuracy of object recognition when using imperfect grouping. In *SCV95,* pages 449-454,1995.

[Roberts, 1965] L. G. Roberts. Machine perception of three-dimensional solids. In James T. Tippett, editor, *Optical and Electro-Optical Information Processing,* chapter 9, pages 159 - 197. MIT Press, Cambridge, MA, 1965.