How Easy is Matching 2D Line Models Using Local Search?

J. Ross Beveridge and Edward M. Riseman, *Senior Member, IEEE*

Abstract—Local search is a well established and highly effective method for solving complex combinatorial optimization problems. Here, local search is adapted to solve difficult geometric matching problems. Matching is posed as the problem of finding the optimal many-to-many correspondence mapping between a line segment model and image line segments. Image data is assumed to be fragmented, noisy, and cluttered. The algorithms presented have been used for robot navigation, photo interpretation, and scene understanding. This paper explores how local search performs as model complexity increases, image clutter increases, and additional model instances are added to the image data. Expected run-times to find optimal matches with 95 percent confidence are determined for 48 distinct problems involving six models. Nonlinear regression is used to estimate run-time growth as a function of problem size. Both polynomial and exponential growth models are fit to the run-time data. For problems with random clutter, the polynomial model fits better and growth is comparable to that for tree search. For problems involving symmetric models and multiple model instances, where tree search is exponential, the polynomial growth model is superior to the exponential growth model for one search algorithm and comparable for another.

Index Terms— Object recognition, optimal model matching, line segment models, run-time performance characterization, random-starts local search.

1 Introduction

Local search [1], [2], [3] is known within the combinatorial optimization literature as an effective means of solving difficult combinatorial optimization problems. Conceptually, local search is refreshingly simple. A tractable neighborhood is defined within an intractable combinatorial search space, and search repeatedly moves to neighboring states until a state is found which is better than all its neighbors. In random starts local search, local search is initiated from states drawn randomly from the space. Multiple trials of local search increase the probability of finding a near optimal solution. Our contribution has been to adapt these ideas to geometric matching problems [4], [5], [6] associated with object recognition.

Our local search algorithms have been used for semi-autonomous photo-interpretation [7], robot navigation [8], [9], [10], and scene understanding [11]. A matching system based upon the ideas presented here is now included in the *KBVision* system produced by AAI in Amherst, Mass. When estimates for object position and orientation are available, a 3D version finds optimal matches in domains with significant 3D perspective [12].

Over the course of our work, we have continued to try to understand how local search scales up to larger and more difficult problems.[1] To address this question, here we present results from our largest empirical study to date. A short version of this study appears in [14] and the results significantly extend those presented in [6]. We use nonlinear regression to compare two alternatives

- run-time growth is exponential $O(e^n)$,
- run-time growth is polynomial $O(n^\beta)$,

where n is the problem size. Over a test suite of 48 problems varying in size from $n = 12$ to $n = 1{,}296$, the polynomial hypothesis is a significantly better fit to our observed run-times. The exponent β is roughly two, and hence, run-times appear to grow as a linear function of n^2.

Problem size n is typically the product of the number of model features m and data features d, i.e. $n = md$. Our matching algorithms consider many-to-many mappings between features, and thus, there are 2^n possible matches. It is perhaps surprising that run-time growth appears polynomial, while search space growth is exponential. These findings suggest that local search compares favorably with two of the better known and well understood alternative matching algorithms: Grimson's tree search [15], [16] and Cass' [17] pose equivalence analysis.

Grimson has shown that for tree search, excluding problems with symmetric models and multiple instances, average case complexity is $O(m^2d^2)$. Since $n = md$, this average case complexity is comparable to that which we observe for local search. However, Grimson also shows that if models are symmetric, or more than one model instance is present, then tree search becomes exponential: $O(d^m)$ or $O(m^d)$ depending on formulation. Our test suite of problems includes both symmetric models and multiple model instances, and, thus, we are observing n^2 average case growth on problems where tree search is exponential.

Pose equivalence analysis cleverly combines search in pose and correspondence space. For 2D problems involving

1. Johnson and Papadimitriou [13] are thanked for inspiring the title to this paper with their own title, "How Easy is Local Search?"

- *J.R. Beveridge is with the Computer Science Department, Colorado State University, Fort Collins Colorado. E-mail: ross@cs.colostate.edu.*
- *E.M. Riseman is with the Computer Science Department at the University of Massachusetts, Amherst Massachusetts.*

Manuscript received 10 May 1996; revised 4 Apr. 1997. Recommended for acceptance by P. Flynn.
For information on obtaining reprints of this article, please send e-mail to: transpami@computer.org, and reference IEEECS Log Number 104918.

rotation, translation, and scale, pose equivalence analysis has an analytic worst-case complexity bound of $O(k^4 n^4)$. Here, k is the number of sides on a convex polygon within which corresponding features must appear. The four derives from the four degrees of freedom in a 2D similarity transform. While the existence of this bound is significant, the dependence upon n^4 precludes large problems in the worst case, and average case performance has not been reported.

We begin this paper with an overview of local search matching including examples on real data. Section 3 and Section 4 present our matching objective function and two local search algorithms. Section 5 develops a random sampling methodology and describes how we characterize the difficulty of individual matching problems. Section 6 presents our nonlinear regression analysis of run-times on our test suite of 48 matching problems. Section 7 summarizes run-times for the real data examples presented earlier. We finish with a discussion of the relationship between random starts local search and algorithms that use feature subsets for pose indexing.

2 Local Search Matching Overview

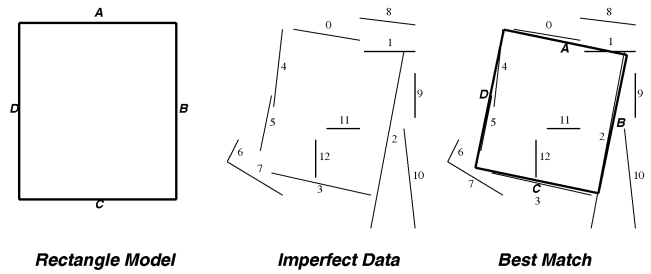
Let us look at an illustration of the key concepts of local search matching and review some results on real data.

2.1 A Thumbnail Sketch of Local Search Matching

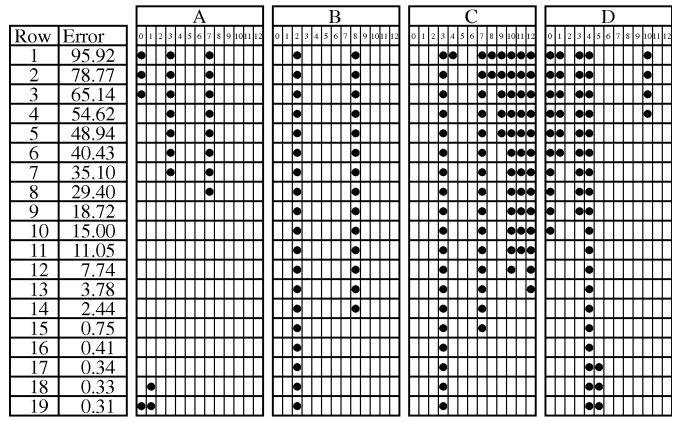
Fig. 1 introduces many of the essential elements of local search matching. Fig. 1a shows an object model, imperfect data, and the associated best match. Both model and data are represented simply as sets of 2D straight line segments. While the segments in this model form a closed contour, closed contours are not required. Models are always fit to corresponding data when evaluating match quality: Note that the model has been rotated, translated, and scaled to best match the data. Fitting is an essential part of our approach and is described in detail in Section 3.3.1.

Fig. 1b illustrates one run, or trial, of local search. Each row indicates a particular match. Those pairs of segments included in the match are indicated with a black circle. The initial match, shown in row one, is selected at random. This introduction of randomness is important and is explained below. Each successive row indicates a new match generated by a single step of the local search process. Search considers the addition or removal of a single pair of corresponding segments, and then makes the change which reduces the match error by the greatest amount. This is an example of a local search neighborhood and neighborhood search strategy. Neighborhoods and search strategies are discussed in Sections 4.1 and 4.2.

The match error, shown for each successive match, defines what constitutes the best match and guides the local search through the combinatoric space of possible matches. Match error takes into account the quality of the fit between the model and data, and how well the corresponding data "explains" or covers the model. Match error details are presented in Section 3.3.



(a)



(b)

Fig. 1. Illustration of local search matching. (a) A model, imperfect data, and an optimal match. (b) Successive rows show local search improving upon an initial random match. A black circle indicates correspondence of a pair.

Local search will not always arrive at the best match. Typically there are many local optima, and the crux of this paper lies in demonstrating that local search remains a viable tool, despite the existence of local optima. Section 5 discusses the use of randomly sampling to mitigate the importance of local optima.

Let us suggest the type of analysis developed in Section 5 by considering actual performance numbers for the problem shown in Fig. 1. On this problem, local search finds the best match in 25 out 100 tries initiated from independently chosen random starting matches. This tells us that the probability of seeing this best match on any single trial is roughly 0.25. Consequently, if 12 independent trials are run, the best match will be seen at least once with probability better than 0.95. Running on a Sun Sparc 10 workstation, our algorithm takes just under one second to run 12 trials: Run-times per trial range from 0.04 to 0.12 seconds.

2.2 Some Sample Results on Actual Image Data

Figs. 2 and 3 illustrate how local search matching might be used to find buildings in aerial photographs. In these examples, models have been hand built. However, while these examples are hand built, local search has been used with real building models on the RADIUS calibrated terrain board imagery [18]. It has also been used with aerial photographs to register orthorectified images [7]. The data line segments for these examples are produced using the Burns algorithm [19].

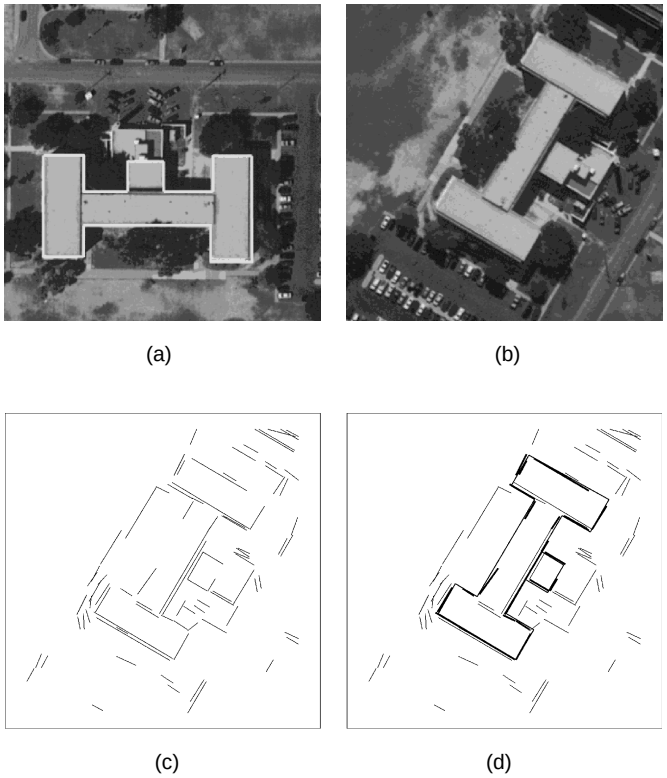


Fig. 2. Matching a building in an aerial photograph. (a) Model in white. (b) Test image. (c) Data segments. (d) Optimal match.

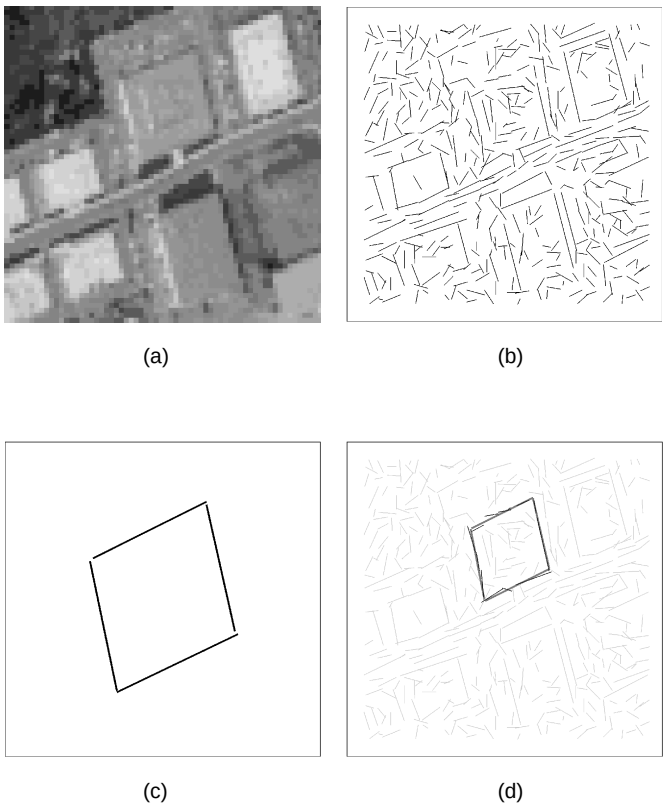


Fig. 3. Match with high clutter and fragmentation. (a) Image. (b) Line segments. (c) Simple building model. (d) Optimal match.

For the match in Fig. 2d, the model is rotated by 120°, illustrating that the original orientation of the model does not matter. In this example, there is little clutter and the building has a distinctive form. We'll see in Section 7 that this is not a difficult problem. The match in Fig. 3 is more difficult. The data is highly fragmented and cluttered, the matching algorithm must find the exact set of 12 data line segments which, when taken together, are the optimal match to the model.[2]

Fig. 4 illustrates a different kind of matching problem. Figs. 4a and 4b show two images of a car coming toward a camera. Fig. 4c shows a labeled set of 27 model line segments extracted from Fig. 4a. Fig. 4d shows a labeled set of data line segments extracted from Fig. 4b. In this example, there is no clutter. However, the model itself is complex and has locally ambiguous structure.

The matrix in Fig. 4e indicates the correspondence mapping found to be the optimal match. A square indicates a pair which potentially match, and a filled-in square indicates a pair belonging to the optimal match. Some pairings between model and data segments are ruled out based upon an initial estimate of how much the model has moved from Fig. 4a to Fig. 4b. The amount of time required to find this match is discussed in Section 7.

The pairs of closely spaced lines bordering the windshield on the top and sides present a challenge to a matching algorithm. The outer segments could locally match the inner segments as well as the outer segments, and the matching algorithm must overcome this local ambiguity. Fitting the object model as a whole effectively disambiguates this structure.

3 Matching as Combinatorial Optimization

Several key concepts underlie our approach.

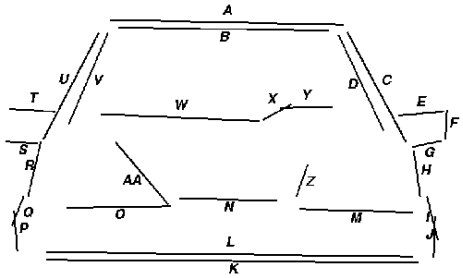
1) Matching is the problem of finding a discrete correspondence mapping, possibly many-to-many, between model and data segments, which minimizes a heuristic measure of match quality (Section 3.1).
2) Efficient global fitting techniques align 2D models to 2D data (Section 3.3.1).
3) Global alignment is the basis for evaluating all matches, and alignment implicitly guides search through the combinatorial space of matches (Sections 3.3.3, 3.3.4, and 4.1).
4) Tractable neighborhoods make exploration of combinatorial match space feasible (Sections 4.1 and 4.2).
5) Random sampling finds, with arbitrarily high probability, globally optimal matches between model and data line segments[3] (Sections 5 and 6).

2. For this model, as for the rectangle shown earlier, there are two equally good matches. These two differ by rotating the model by 180°.

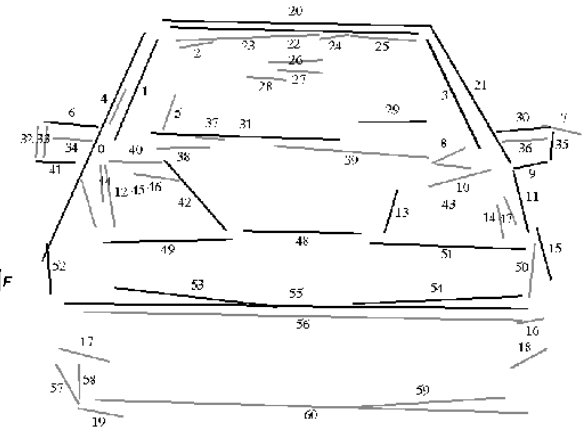
3. Often, but not always, in the domain of matching, visual inspection of results is enough to make a determination that an algorithm has, in fact, found the globally optimal match. In some subtle problems the true global optima may not be known to us, and, in these cases, we are referring to the best our algorithms have ever found.

(a)

(b)



(c)



(d)

Pairs	Pairs	Pairs	Pairs	Pairs	Pairs
A 20	F 35	K 55	P 52	U 0	Z 13
B 22	G 9	L 53, 54	Q 0	V 2	AA 42
C 21	H 11	M 51	R	W 31	
D 3	I 15	N 48	S 41	X	
E 30	J	O 49	T 6	Y 29	

(e)

Fig. 4. Matching an oncoming car. (a) Image one. (b) Image two. (c) Model extracted from image one. (d) Data from Image two with matching segments in black and others in gray. (e) Optimal correspondence mapping between segments.

3.1 Discrete Space of Many-to-Many Correspondences

Match error is defined over a discrete space of possible model-to-image segment mappings. Let M be the set of model segments, D the set of data segments, and S the cross product of M and D, the correspondence space C is the power set of S: $C = 2^S$. In other words, C contains all possible many-to-many mappings between model and data segments.

Fig. 5 shows an example of a match for a stylized tree. Relating the above definitions to this example,

$M = \{A, B, C, D, E, F, G, H, I, J, K, L\}$
$D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$
$S = \{(A, 0), (A, 1), \ldots, (A, 12), (B, 0), \ldots, (L, 12)\}$
$C = 2^S$

The sets M, D, S, and C contain 12, 13, 156, and 2^{156} elements, respectively.

3.2 Match Error

A match error E_{match} is defined such that

$$E_{\text{match}} : C \rightarrow \Re, \quad E_{\text{match}}(c) \geq 0 \quad \forall\, c \in C$$

The goal of matching is to find the optimal match c^* where

$$E_{\text{match}}(c^*) \leq E_{\text{match}}(c) \quad \forall\, c \in C$$

Our match error, E_{match}, measures three things:

1) the degree to which the model *fits* the corresponding data segments,

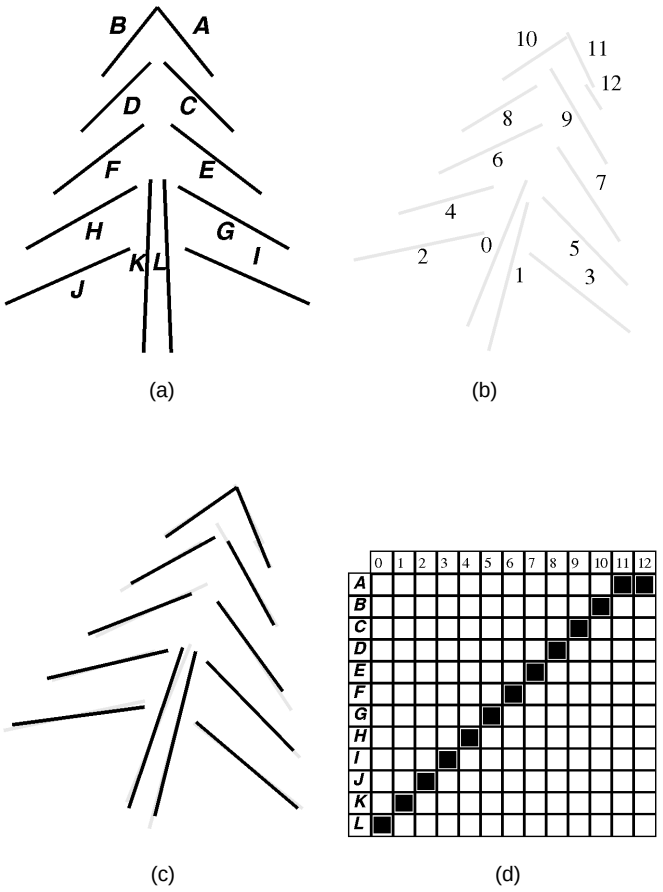


Fig. 5. An example match from the test suite to be presented in Section 6. (a) Model line segments. (b) Data line segments. (c) Model shown matched to data. (d) Correspondence matrix with matching pairs blacked out. The match error $E_{\text{match}} = 0.052$ for this best match. It is the sum of fit error $E_{\text{fit}} = 0.022$ and the omission error $E_{\text{om}} = 0.030$.

2) the degree to which portions of the model are *omitted* from the match, and
3) the degree to which the best-fit transformation is acceptable.

In its general form, it may be written as:

$$E_{\text{match}}(c) = E_{\text{fit}}(c) + E_{\text{om}}(c) + E_{\mathcal{F}}(c) = \left(\frac{1}{\sigma^2}\right) E_{\text{I}}(c) + E_{\text{om}}(c) + E_{\mathcal{F}}(c) \quad (1)$$

where $E_{\text{I}}(c)$ is a normalized residual after least-squares fitting of the model to the data and the weighting coefficient σ controls the relative importance of fit versus omission.

Our experience [4], [5], [6], [12] suggests $E_{\text{match}}(c)$ induces reasonable rankings over matches for a given model. It is not intended for comparing matches to *different* models. Developing measures to compare complex versus simple models is itself a subtle problem [20].

In part inspired by Wells [21], we have experimented with a data omission term penalizing matches that leave data unmatched. On problems of the type presented here, data omission hurts rather than helps. Consider what happens when two instances of a model appear in one image. Making the quality of a match to one instance dependent upon the size of the other instance leads to unpredictable and undesirable behavior. However, in the context of matching laser range data we have found a data omission term to be helpful [20].

To evaluate E_{match}, first a global alignment of model to data must be determined based upon the correspondence c. Next, the model must be transformed to this configuration and omission measured. Finally, the best fitting transformation $\mathcal{F}^*$ is compared to some acceptable range and a penalty is added if the transformation lies outside these bounds.

3.3 Match Evaluation Using Global Alignment

Of several alternative ways to fit a line segment model to line segment data, perhaps the most obvious but flawed approach is to minimize point-to-point distance between corresponding segment endpoints or midpoints. Point-to-point fitting has problems when line segments are fragmented or overextended. Both Lowe and Ayache [22],[23] appropriately suggested it is better to minimize a perpendicular point-to-line distance measure.

Ayache [23] developed a closed-form solution for the rotation, translation, and scale which minimizes squared perpendicular distance from endpoints of model line segments to infinitely extended data lines. The weakness of this approach is that often line segments extracted from imagery fragment [24]. Extending a fragmented segment amplifies orientation errors and in turns skews the overall resulting fit. We have developed closed-form solutions with the role of data and model reversed so that the inherently more stable model segments are infinitely extended [4], [24].

In another improvement to least-squares fitting of line models, the perpendicular distance is integrated along the data line segment rather than being allowed to concentrate at the ends. Consequently, the optimal least-squares fit is completely invariant with respect to breakpoints in data line segments. More precisely, if a single data segment is broken into two adjacent segments at any position along the segment, the resulting fit will not change.

As an aside, the case of rigid 2D fitting deserves comment. For many matching techniques, such as tree search [16] and Hough transforms [25], forcing scale to remain constant makes problems easier to solve. In a somewhat counter-intuitive discovery, we have found that least-squares fitting of line models for the rigid case is harder than for the variable scale case. Variable scale requires the solution of a quadratic equation, while fixing scale leads to a quartic equation [6].

3.3.1 Minimizing Perpendicular Distance

Models are fit to data so as to minimize the sum of the integrated squared perpendicular distance (ISPD) between h corresponding pairs of segments in c. The perpendicular distance from the endpoint of a data line segment to the corresponding infinitely extended model line segment may be parameterized by a similarity transformation $\mathcal{F}$.

The transformation $\mathcal{F}$ rotates, translates and scales the model relative to the data and may be written as:

$$\mathcal{F} = \left(\vec{\Phi}_{DM},\, \vec{T}_{DM},\, s_{MD}\right) = \left(\begin{vmatrix}\cos\phi & \\ \sin\phi & \end{vmatrix},\, \begin{vmatrix} t_x \\ t_y \end{vmatrix},\, s_{MD}\right) \tag{2}$$

The terms $\vec{\Phi}_{DM}$ and $\vec{T}_{DM}$ rotate and translate data segments data relative to the model, while s_{MD} scales the model relative to the data segments. This asymmetric treatment of rotation and translation versus scaling simplifies the solution for the best-fit transformation. The subscripts indicate the direction of the transformation: *DM* for data reference frame to model and *MD* for model to data.

Rotation may be written with the vector $\vec{\Phi}_{DM}$ provided we introduce a matrix to represent 2D points. Thus, the *j*th endpoint of a data line segment D_i is written as:

$$D_{ij} = \begin{vmatrix} x_{ij} & -y_{ij} \\ y_{ij} & x_{ij} \end{vmatrix} \tag{3}$$

The term s_{MD} scales the distance ρ from the origin to the nearest point on the infinitely extended model line. Hence, for a model segment M_i,

$$\rho_i = \hat{N}_i \cdot M_{ik} \tag{4}$$

where $\hat{N}_i$ is the unit normal vector to the line M_i, and M_{ik} is any point lying on M_i.

The perpendicular distance v_{ij} from the endpoint j of a data line segment D_i to the corresponding infinitely extended model line segment M_i may be written as a function of $\mathcal{F}$:

$$v_{ij}(\mathcal{F}) = \left(\hat{N}_i \cdot \left(D_{ij}\vec{\Phi}_{DM} + \vec{T}_{DM}\right) - s_{MD}\rho_i\right) \tag{5}$$

The integrated squared perpendicular distance may be defined in terms of v_{ij}: Integrated (E_I) and endpoint concentrated (E_P) measures are shown to highlight the difference between the two:

$$E_P = \left(\frac{1}{L_D}\right)\sum_{i=1}^{h}\frac{\ell_i}{2}\left(v_{i1}^2 + v_{i1}^2\right) \tag{6}$$

$$E_I = \left(\frac{1}{L_D}\right)\sum_{i=1}^{h}\frac{\ell_i}{3}\left(v_{i1}^2 + v_{i1}v_{i2} + v_{i1}^2\right) \tag{7}$$

The term ℓ_i is the length of the data segment D_i and L_D is the cumulative length over all matched data line segments. Normalizing by L_D yields a weighted average of squared perpendicular distance over all pairs of matching model and data line segments.

Multiplying out the terms in (7) and collapsing the sums yields the following:

$$E_I = \left(\frac{1}{L_D}\right)\left(\vec{\Phi}^{\mathrm{T}}A\vec{\Phi} + 2\vec{T}^{\mathrm{T}}B\vec{\Phi} + \vec{T}^{\mathrm{T}}C\vec{T} - 2\vec{U}^{\mathrm{T}}\vec{\Phi}s + 2\vec{V}^{\mathrm{T}}\vec{T}s + ks^2\right) \tag{8}$$

$$\begin{aligned} A &= \sum_{i=1}^{h}\left(\ell_i / 3\right)\left(D_{i1}^{\mathrm{T}}\hat{N}_i\hat{N}_i^{\mathrm{T}}D_{i1} + D_{i1}^{\mathrm{T}}\hat{N}_i\hat{N}_i^{\mathrm{T}}D_{i2} + D_{i2}^{\mathrm{T}}\hat{N}_i\hat{N}_i^{\mathrm{T}}D_{i2}\right) \\ B &= \sum_{i=1}^{h}\sum_{j=1}^{2}\left(\ell_i / 2\right)\hat{N}_i\hat{N}_i^{\mathrm{T}}D_{ij} \\ C &= \sum_{i=1}^{h}\ell_i\hat{N}_i\hat{N}_i^{\mathrm{T}} \\ \vec{U} &= \sum_{i=1}^{h}\sum_{j=1}^{2}\left(\ell_i / 2\right)\rho_i D_{ij}^{\mathrm{T}}\hat{N}_i \\ \vec{V} &= \sum_{i=1}^{h}\ell_i\rho_i\hat{N}_i \\ k &= \sum_{i=1}^{h}\ell_i\rho_i^2 \end{aligned} \tag{9}$$

For simplicity the subscripts have been dropped from the transformation terms: $s = s_{MD}$, $\vec{\Phi} = \vec{\Phi}_{DM}$, and $\vec{T} = \vec{T}_{DM}$.

The best-fit alignment of the model to the data

$$\mathcal{F}^* = \left(\vec{\Phi}^*,\, \vec{T}^*,\, s^*\right) \tag{10}$$

minimizes (8) subject to the constraint

$$\vec{\Phi}^T\vec{\Phi} = 1 \tag{11}$$

To solve for $\mathcal{F}^*$, set to zero the derivative of (8) with respect to s and solve for s^* as a function of $\vec{\Phi}$ and $\vec{T}$.

$$s^* = \left(\vec{U}^{\mathrm{T}}\vec{\Phi} + \vec{V}^{\mathrm{T}}\vec{T}\right) / k \tag{12}$$

Substitute s^* for s in (8) and drop the normalization term (L_D) to yield:

$$E_I' = \vec{\Phi}^{\mathrm{T}}D\vec{\Phi} + 2\vec{T}^{\mathrm{T}}E\vec{\Phi} + \vec{T}^{\mathrm{T}}F\vec{T} \tag{13}$$

where

$$D = A - \vec{U}\vec{U}^{\mathrm{T}} / k \qquad E = B - \vec{V}\vec{U}^{\mathrm{T}} / k \qquad F = C - \vec{V}\vec{V}^{\mathrm{T}} / k$$

Set the partial derivative of (13) with respect to $\vec{T}$ to zero and solve for $\vec{T}^*$ as a function of $\vec{\Phi}$:

$$\vec{T}^* = -F^{-1}E\,\Phi \tag{14}$$

Finally, substitute $\vec{T}^*$ into (13) to get

$$E_I'' = \vec{\Phi}^{\mathrm{T}}G\vec{\Phi}, \text{ where } G = D = E^{\mathrm{T}}F^{-1}E \tag{15}$$

By Rayleigh's Principle ([26 p. 429]) the vector $\vec{\Phi}^*$ which minimizes 15 is the unit eigenvector associated with the lesser eigenvalue of the matrix *G*.

For some configurations, such as corners, the best-fit transformation $\vec{\Phi}^*$, $\vec{T}^*$, s^* is underconstrained. Adding a regularizing term, squared Euclidean distance between midpoints of corresponding line segments, resolves these cases. We use 10^{-3} as the weight for this regularization term.

3.3.2 Fit Error is Normalized Residual after Global Fitting

The fit error is the residual of E_I from (7) weighted by *sigma*:

$$E_{\mathrm{fit}}(c) = \left(\frac{1}{\sigma^2}\right)E_I\left(\vec{\Phi}^*,\, \vec{T}^*,\, s^*\right) \tag{16}$$

It is possible now to give a clearer interpretation to the

relative weighting term σ used in (1). Since E_I is average squared perpendicular distance, the fit error $E_{\text{fit}}(c)$ reaches 1.0 when the average perpendicular distance reaches σ. Practically speaking, since the omission error is normalized to the range [0, 1], σ is the largest amount of integrated perpendicular distance allowed in a match. This is because if $E_{\text{fit}}(c)$ exceeds 1.0, the match error will favor removal of those segments causing the poor fit. As a parameter of the matching system, σ is described as the "maximum-displacement," and unless otherwise stated, is set to 4.0 pixels in all the experiments which follow.

The symbol σ is commonly used for the standard deviation of a noise process, and our use of the term is meant to be suggestive. While the connection is not rigorous, one may think informally of σ as the standard deviation of a noise process which skews data line segments relative to their "true" position. For an excellent treatment on noise process models and straight line extraction, see [27].

3.3.3 Omission Error is a Nonlinear Function of Coverage

The omission error for a model segment M is defined as a nonlinear function of the percent p of the model line unaccounted for by data. Let M_1 and M_2 be the endpoints of M and define a parametric expression for points lying on M.

$$P(t) = \vec{T}t + M_1,\ 0 \le t \le \ell,\ \vec{T} = \frac{M_2 - M_1}{\left|M_2 - M_1\right|} \tag{17}$$

where ℓ is the length of M in pixels and $\vec{T}$ is a unit tangent vector pointing from M_1 to M_2. The t value for the perpendicular projection of any arbitrary point D may be written:

$$t = \left(D - M_1\right) \cdot \vec{T} \tag{18}$$

A data segment D_i with endpoints D_{i1} and D_{i2} transformed into the best-fit configuration covers M over the interval $[t_{i1}, t_{i2}]$ where

$$t_{i1} = \left(D_{i1} - M_1\right) \cdot \vec{T}$$
$$t_{i2} = \left(D_{i2} - M_1\right) \cdot \vec{T}$$

The percent omitted p is the portion of M (range $[0, \ell]$) not included in the projection range $[t_{i1}, t_{i2}]$ of any corresponding data segment.

While p itself could be used as an error, there are reasons to favor a nonlinear function of p. First, even under the best of circumstances, a small amount of omission is to be expected (e.g., the ends of lines are often difficult to extract). Thus, small values of p should incur a relatively small penalty. Second, as increasingly large portions fail to be found, the penalty should begin to grow substantially. The following nonlinear function of p captures this relationship:

$$E_{\text{om}}(p) = \begin{cases} \dfrac{e^{\gamma p} - 1}{e^{\gamma} - 1} & \text{if } \gamma \neq 0 \\ p & \text{otherwise} \end{cases} \tag{19}$$

Since p lies in the range [0, 1], the omission error also lies in this range. The degree of nonlinearity is controlled by γ. However, the exact manner in which changing γ changes the form of $E_{\text{om}}(p)$ is less than obvious.

It is helpful to introduce an auxiliary parameter a, and then define γ in terms of a:

$$\gamma = 2 \ln\left(\frac{2}{a} - 1\right) \tag{20}$$

The parameter a *attenuates* the omission error for small amounts of omission. In the special case $a = 1.0$, E_{om} is a linear function of p and as a decreases E_{om} drops below the linear case. A value of $a = 0.5$ drops E_{om} by 50 percent at the midpoint of the curve, i.e. where $p = 0.5$. In all the experiments presented in this paper, $a = 0.75$.

Omission error for a match is a weighted sum of omission for each model segment:

$$E_{\text{om}} = \sum_{m \in M} \left(\frac{\ell(m)}{L_M}\right) E_{\text{om}}\left(p_m\right) \tag{21}$$

where ℓ is the length of each model line segment and L_M is the cumulative length of all model line segments. Weighting by relative length makes the omission error more directly comparable to the normalized fit error E_{fit}.

3.4 Discouraging Excess Scale Change

We discourage correspondences that imply near pathological transformations, such as shrinking the model to a point. In order to make such matches less desirable the following $E_{\mathcal{F}}(c)$ is defined:

$$E_{\mathcal{F}}(c) = \begin{cases} \left(1 / s^*\right) - r & \text{if } s^* < 1 / r \\ 0.0 & \text{if } 1 / r \le s^* \le r \\ s^* - r & \text{if } s^* > r \end{cases} \tag{22}$$

The scale change s^* indicates how much the model changes size and the parameter r defines an allowable range of size changes. If s^* becomes larger than r, then the error starts out at 0 and grows linearly with s^*. If s^* becomes smaller than the reciprocal of r, then error grows linearly in the reciprocal of s^*. The error approaches infinity as s^* approaches infinity or 0. For all the experiments presented here $r = 2$: Models may range from half to twice their original size without penalty.

4 Two Local Search Matching Algorithms

A Hamming-distance-one neighborhood and the associated steepest-descent algorithm is described in the following section. A neighborhood structure specifically tailored to geometric matching problems is defined in Section 4.2.

4.1 Steepest-Descent Local Search

To understand how local search is carried out, it helps to first understand our bit-string encoding of the search space C and its relationship to the local search neighborhood. This encoding assigns a unique bit to every pair of model and image line segments in the set $S = M \times D$. Therefore, if $|S| = n$, then the search space C maps to the space of all possible bit strings of length n. A one in the ith position of the string indicates the ith pair $s_i \in S$ is part of the match.

The local search neighborhood is now defined to contain all strings within Hamming-distance-one of the current match.

Local search is itself just a loop in which the n neighbors of some current match are evaluated and if a better neighbor is found it becomes the current match. More specifically, a steepest-descent local search algorithm in the Hamming-distance-one neighborhood toggles each bit in the bit string encoding of the current match and evaluates the match error $E_{\text{match}}(c)$. The algorithm records the toggle which yields the greatest drop in $E_{\text{match}}(c)$ and uses this to create a new better state. Search terminates when none of the neighboring matches are an improvement upon the current match. This algorithm has already been illustrated in Fig. 1b. This neighborhood can add or remove a single pair of model-data segments in one move: It cannot swap one data segment for another.

A Hamming-distance-two neighborhood permits swapping of segments. However, the size of the Hamming-distance-two neighborhood is n^2. Early experiments were made with this neighborhood [24], but the n^2 growth in neighborhood size makes this an unattractive alternative for even medium-sized problems. Results using the Hamming-distance-one neighborhood show run-times to solve complete problems appear to grow as a function of n^2. If the neighborhood alone grows as n^2, then the resulting local search algorithm cannot help but do worse.

For efficiency, local search exploits the fact that the n neighbors are slight perturbations of the current match. In principle, for every neighbor tested, the model must be completely fit to the corresponding data and the associated omission over the entire model computed. However, the change in fit error can be more efficiently computed incrementally relative to the current match [6].

The incremental computation of E_{fit} requires about 20 floating point additions and multiplications and the finding of one square root. A useful heuristic is, therefore, to see if the change in fit error appears to preclude improvement. Most neighbors being tested suggest adding a pair of model-data segments. The rule of thumb is if that if the fit error grows more than can be possibly made up for by an associated drop in omission for that model segment, then do not bother to compute the complete change in omission error. Applying this heuristic reduces required computation by nearly an order of magnitude [28]. It is a heuristic because it neglects subtle interaction effects in which a small match change might drop omission error for many model line segments. Such cases exist but are rare.

4.2 Subset-Convergent Local Search

Subset-convergent local search tests whether subsets of a locally optimal match in the Hamming-distance-one neighborhood are "consistent" with the overall match. For a truly good match, Hamming-distance-one local search initiated from subsets of the matching pairs should converge back to the same match. Alternatively, if the match is poor, then subsets of the match are probably incompatible, and search initiated from a subset may well lead to an overall better match. Our experiments, including those presented below, have shown this intuition to be correct.

Subset-convergent local search begins by running the steepest-descent algorithm until a Hamming-distance-one local optima is encountered. This match is recorded and search is initiated from new matches containing only subsets of the model-data pairs present in the local optima. Subsets are defined relative to the model segments. If $M' \subset M$ is a designated subset, then only pairs s containing model segments in M' are retained in the new match. These matches almost always score worse than the local optima since removing data increases omission error. However, steepest-descent initiated from these subset matches often leads to better matches. If search from all the subsets fails to yield a better match, then subset-convergent local search terminates.

There are many ways to define the subsets and subset selection is perhaps the least studied aspect of our algorithm. One guiding principle has been that the total number of subsets remain small and not grow as a function of model size. The heuristic we have chosen begins with a list of all $\frac{m(m-1)}{2}$ pairs of model segments and passes this list through three filters:

- *Remove nearly parallel pairs of segments*: Remove pairs differing in orientation by less then five degrees.
- *Retain pairs with proximal endpoints*: Sort the remaining pairs in ascending order according to the minimum Euclidean distance between endpoints. Retain the first m pairs in this list.
- *Retain the four longest disjoint pairs*: Sort the m pairs in descending order according to the sum of the lengths of the two segments. Select the first four disjoint pairs in this list. If there are fewer than eight model segments then do not require the pairs to be disjoint.

Provided there are at least four pairs of nonparallel model segments to begin with, this algorithm will always select four pairs of model segments to serve as subsets for the subset-convergent local search algorithm.

5 Local Optima and Random Sampling

Both of the local search algorithms described above are deterministic: Starting from any match c in the space C, local search will move predictably to an associated local optima. As one might expect, for all but trivial problems, there are a tremendous number of local optima in the space C. Often this fact causes people to prematurely dismiss local search as a useful technique.

Tovey [29] makes several very insightful observations about local search and local optima. The first is that a deterministic local search algorithm imposes a "forest structure" upon the search space. To be more specific, the space may be viewed as a "forest" of trees, with the root of each individual tree a locally optimal match. From nodes which are not locally optimal, there is path leading "down" the tree to the root. These paths represent the successive matches found by the local search algorithm. There is one tree, the globally optimal tree, whose root is the global optima.[4]

To visualize what this forest looks like, imagine placing each node in the search space at a "height" off the ground corresponding to the match error: Nodes with larger error are higher. Only one branch leads down from each node,

4. For simplicity, ties for "best" are ignored.

and this represents the move from one state to another taken by steepest-descent local search. At the bottom of each tree is a root node representing the local optima found by local search initiated from any branch in the tree.

The shape of this forest, the number of trees and their relative size, are the combined product of the local search neighborhood definition, the criterion function and the specific problem instance.[5] Tovey [29] proves that for several classes of NP-complete problems the expected number trees in the forest grows exponentially. He further expresses a belief that all NP-complete problems have exponentially many local optima, but stops short of offering a proof (a proof would amount to a proof that P ≠ NP).

From a practical standpoint, Tovey's most important observation is that while the number of local optima may explode, the relative size of the trees containing local optima versus the size of the tree containing the global optima may remain such that random starts local search will continue to perform well.

Let us formalize some of these notions. First, let O be the fraction of the search space C containing the global optima. Next, let us assume local search is initiated from a state c_i uniformly sampled from C. Under these conditions, the probability P_s of successfully arriving at the global optima on any single independent random trial is:

$$P_s = O \tag{23}$$

If P_s for a given problem is known, then the probability of failing to see the global optima over t independent trials is simply

$$Q_f = (P_f)^t, \quad \text{where} \quad P_f = 1 - P_s \tag{24}$$

From (24) it is possible to compute the number of trials t_s required to find the optimal match with probability $Q_s = 1 - Q_f$:

$$t_s = \lceil \log_{Pf} Q_f \rceil \tag{25}$$

In all the work presented in this paper, Q_s is set to 0.95.

5.1 The Work Required to Solve a Specific Problem

Given a test problem with a known solution, it is usually possible to determine how many trials are required to find that solution with probability Q_s. We run k trials of local search and record the number of times the global optima is seen, o. In classifying the result as true for an optimal match and false otherwise, we are equating our multiple trials with a binomial process with unknown probability of returning true P_t. A true value from the binomial process means success at finding an optimal match, and consequently $P_s = P_t$. This means that the maximum likelihood estimate for P_s is the ratio:

$$\hat{P}_s = \frac{o}{k} \tag{26}$$

It is also possible to predict the degree of uncertainty in the estimate $\hat{P}_s$ and these bounds for different combinations of trials and $\hat{P}_s$ appear in [6].

5. This interplay between the neighborhood definition and the evaluation function makes formal analysis of local search difficult [29].

A good rule of thumb is that the best match must be seen some reasonable number of times: perhaps more than 10 times. While all the problems presented in this paper may be studied in this fashion, clearly there are limits. At some point it becomes prohibitive to run sufficient trials to reliably estimate $\hat{P}_s$. Typically we run 100 trials for easy problems and 1,000 for hard problems.

To determine an expected run-time r_s to solve a problem, take the estimated probability of success P_s and the average run-time r for a single trial of local search, compute the required number of trials t_s to solve the problem with 95 percent confidence using (25), and multiply time-per-trial by the number of trials:

$$r_s = t_s r \tag{27}$$

This measure of problem difficulty will be used in Section 6.2 to test two alternative hypothesis about how run-time grows as a function of problem size.

5.1.1 Biased Random Sampling

The choice of initial random starting matches need not be uniform, and it is common [3], [29] to bias random selection of starting states in order to improve the likelihood that the state is in the tree leading to the global optima. While introducing bias destroys the strict interpretation of P_s as the fraction of the space spanned by the globally optimal tree, P_s may still be reliably estimated using random sampling, and (25), (26), and (27) still hold.

Random initial correspondences are generated with, on average, λ data segments matched to each model segment. This is done by defining a binding probability P_{M_i} for each model segment M_i:

$$P_{M_i} = \min(0.5, \lambda / K_i) \tag{28}$$

where K_i is the number of model-data pairs included in M_i. When building the initial match, a pair $s \in S$ is included in the initial correspondence c_i with probability P_{Mi}. For the example in Fig. 1, $\lambda = 4$. Elsewhere, $\lambda = 2$ unless otherwise noted.

5.1.2 How-Many-Trials?

To illustrate these ideas with a concrete example, consider the global optima shown in Fig. 5. For the subset-convergent algorithm, this match was found in 761 out of 1,000 random trials, and the average run-time per trial was $r = 2.2$ seconds. Consequently,

$$\hat{P}_s = 0.76 \quad t_s = 3 \text{ trials} \quad r_s = 6.6 \text{ seconds} \tag{29}$$

For the steepest-descent algorithm on this same problem, the global optima was found in 70 out of 1,000 trials and the average run-time per trial was $r = 0.8$ seconds. Thus,

$$\hat{P}_s = 0.07 \quad t_s = 42 \text{ trials} \quad r_s = 33.6 \text{ seconds} \tag{30}$$

5.2 Example Optima—The Good, the Bad, and the Ugly

Fig. 6 shows a sampling of local optima for the Tree example presented in Fig. 5. Figs. 6a and 6b demonstrates that

local search is finding partial symmetries in the model. These local optima tell us something about the structure of our models: For instance, revealing the self-similar structure of the tree branching structure. However, most local optima are uninteresting. Figs. 6c and 6d show two such matches. The match error ranks these local optima as worse than those arising out of the symmetry in the tree branching structure.

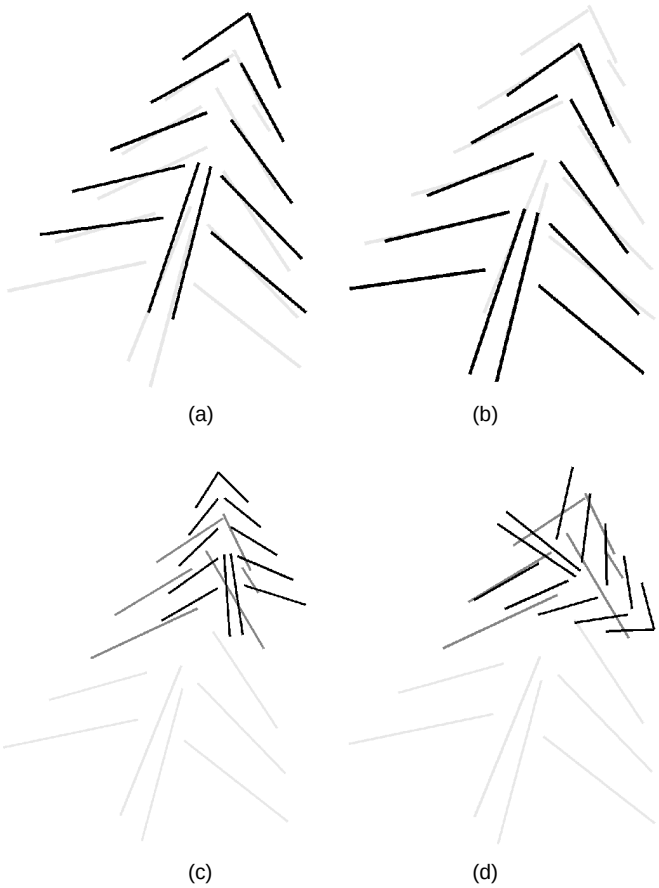


Fig. 6. Local optima: (a) Model shifted up with $E_{match} = 0.395$. (b) Model shifted down with $E_{match} = 0.340$. (c) $E_{match} = 0.523$. (d) $E_{match} = 0.575$.

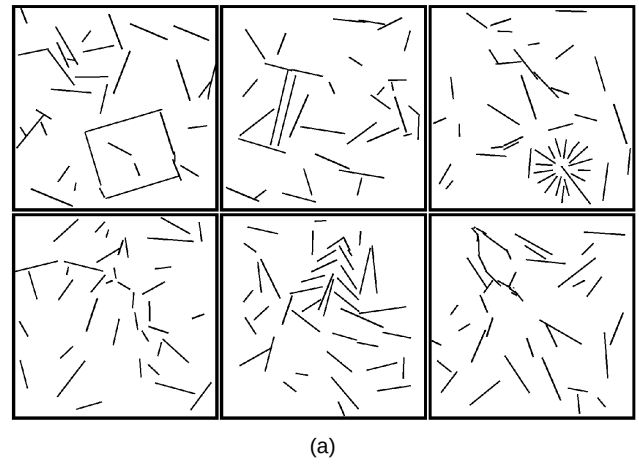
6 Characterizing Performance

A test suite of 48 distinct matching problems is used in this study. They are derived from the six "stick figure" models shown in Fig. 7. We and others [30] have used this test suite in the past to benchmark local search matching algorithms [6], [14] and to compare local search with genetic algorithms [28]. The test suite as well as our optimal matches are available through our web site:

`http://www.cs.colostate.edu/~vision`

Each model is defined by a set of 2D straight line segments. In matching, these models may be rotated and translated to lie anywhere in the image. In addition, model size is allowed to vary. The models have been selected to be simple enough to permit study yet varied enough to test for possible weaknesses in a matching algorithm.

For example, the dandelion exhibits a 16-fold near symmetry. Symmetries in models complicate matching for



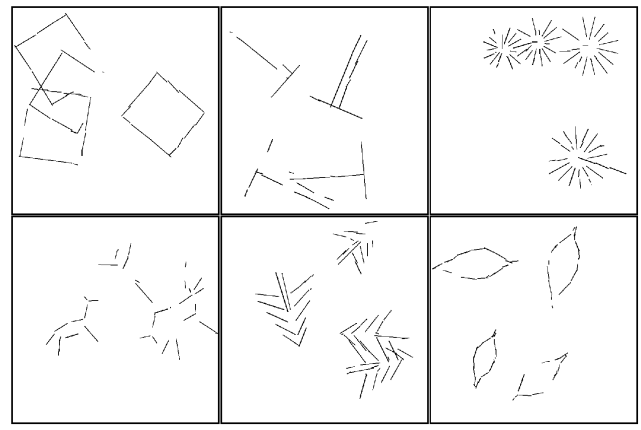


Fig. 8. Test data. (a) Random clutter. (b) Multiple instances.

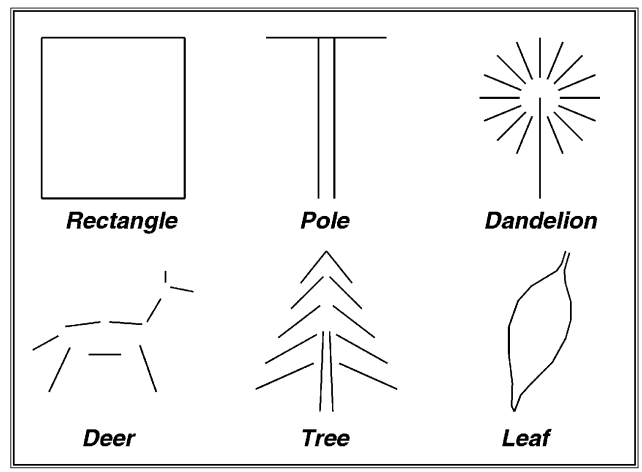


Fig. 7. Six stick figure models used in tests.

many well established techniques [16]. The leaf presents an example where model and data line segments approximate a curved contour. In this case, a many-to-many mapping between model and image segments is needed to account for breakpoints falling at different positions along the curve.

TABLE 1
EXPECTED RUN-TIMES SORTED BY PROBLEM SIZE *N*

				r_s (seconds)		Ratio	
M	C.	I.	*n*	SD	SC	SD	SC
Po	0	1	12	0.2	0.1		2.4
Po	0	1	24	0.8	0.4		2.0
Re	0	1	28	0.6	0.5		1.2
Po	10	1	42	2.6	0.7		3.7
Po	0	2	42	1.2	0.8		1.5
Re	0	1	52	1.0	1.2	1.2	
Re	10	1	68	1.1	1.2	1.1	
Po	20	1	72	3.5	1.8		1.9
De	0	1	81	3.9	1.4		2.8
Po	0	3	81	3.7	2.7		1.4
Po	0	4	96	9.0	3.9		2.3
De	0	1	99	7.6	2.2		3.5
Po	30	1	102	9.6	7.5		1.3
Re	20	1	108	9.4	3.6		2.6
Re	0	2	108	3.6	10.4	2.9	
Re	0	3	124	5.6	12.0	2.1	
Re	30	1	148	12.6	9.6		1.3
Tr	0	1	156	33.6	6.6		5.1
Re	0	4	168	10.0	32.5	3.3	
De	10	1	171	17.4	23.0	1.3	
De	0	2	180	21.7	45.0	2.1	
Tr	0	1	216	66.3	3.6		18.4
De	20	1	261	43.0	74.0	1.7	
De	0	3	261	59.0	6.2		9.5
Da	0	1	272	239.4	119.7		2.0
Tr	10	1	276	76.5	55.1		1.4
Le	0	1	306	23.2	36.0	1.6	
De	0	4	342	159.0	335.0	2.1	
Le	0	1	342	23.8	41.5	1.7	
De	30	1	351	96.0	128.8	1.3	
Tr	20	1	396	114.4	14.0		8.2
Da	0	1	416	233.1	333.0	1.4	
Da	10	1	432	732.8	113.3		6.5
Tr	0	2	432	310.5	17.2		18.1
Le	10	1	486	147.5	60.5		2.4
Tr	30	1	516	297.0	18.8		15.8
Tr	0	3	552	461.9	24.0		19.2
Da	20	1	592	915.4	102.9		8.9
Le	0	2	648	301.6	511.2	1.7	
Le	20	1	666	217.6	268.8	1.2	
Da	0	2	736	1,897.0	265.2		7.2
Da	30	1	752	1,153.2	121.8		9.5
Tr	0	4	780	1,554.8	40.4		38.5
Le	30	1	846	494.1	615.4	1.2	
Le	0	3	882	2,571.4	1,340.0		1.9
Da	0	3	1,130	8,676.8	6,515.5		1.3
Le	0	4	1,293	823.6	2,499.8	3.0	
Da	0	4	1,293	10,322.4	10,923.3	1.1	

LEGEND	
M	Model
C	Number of Clutter Lines
I	Number of Model Instances
n	Number of Model-Data Pairs
SD	Steepest-descent
SC	Subset-convergent
r_s	Time to 95 percent Prob. Optimal
Ratio	Amount Faster Than Other

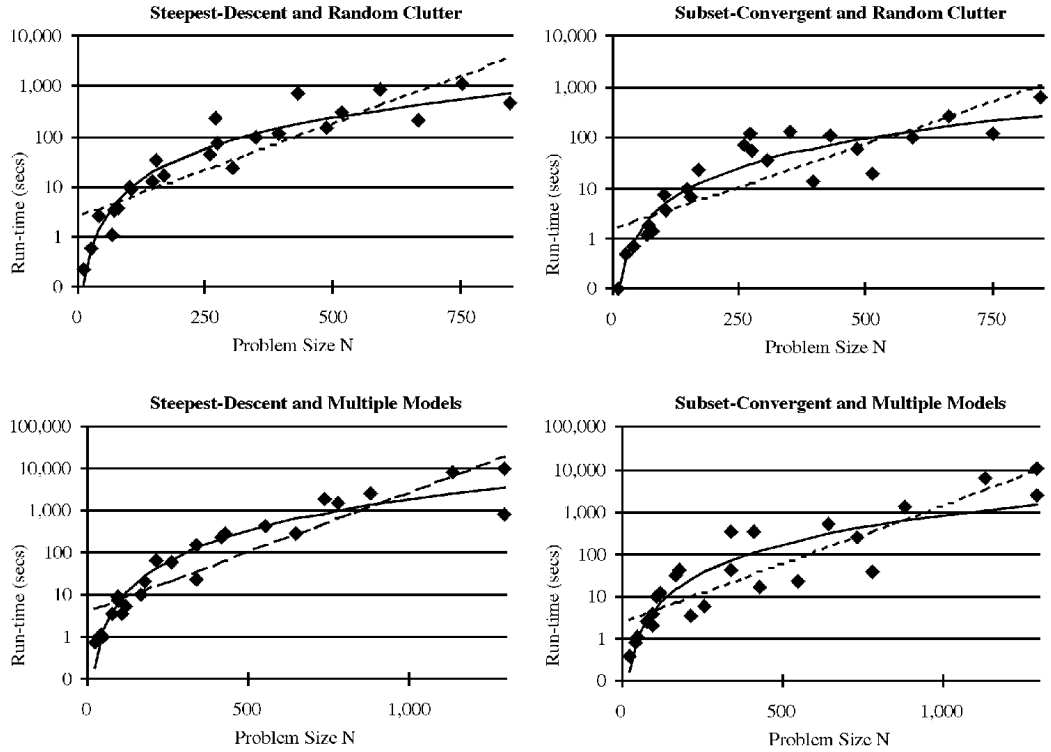


Fig. 9. Estimated run-times r_S as a function of n. Times are broken out by algorithm and random clutter versus multiple model instances. Both polynomial and exponential regression lines are shown.

A Monte Carlo simulator produces corrupted image data. The simulator rotates, translates, and scales the model so placement and size is unknown. Model segments are also fragmented and skewed. In 24 of the problems, zero, 10, 20, and 30 additional clutter segments are randomly placed about the image. A sampling of this data is shown in Fig. 8a. In the other 24 problems, one, two, three, and four instances of the model are added to the image. A sampling of this data is shown in Fig. 8b.

6.1 Steepest-Descent Versus Subset-Convergent Local Search

Having run each algorithm 1,000 times on each of the 48 problems on a Sparc 10, we have reliable estimates of r and P_s. From these, we compute t_s and r_s. Values for r_s for each problem are given in Table 1. Size $n = md$ is indicated for each problem. Due to fragmentation, d may be larger than m, even when no clutter is present.

Table 1 indicates SC does better on 30 out of the 48 problems. Moreover, while SD is never more than 3.3 times faster than subset-convergent on any problem, SC is as much as 38.5 times faster than SD. Overall, to solve the entire test suite, SC would require 6.9 hours compared to 8.9 hours using SD.

An interesting difference does emerge relative to which models do better using which algorithms. For both the pole and tree, the SC algorithm is better on eight out of eight problems. For the dandelion, SC is still doing well, performing better on six out of eight problems. For both the rectangle and deer, SC is better on only three out of eight problems. Finally, for the leaf problem, SC does better on only two out of eight problems. It appears the difference between these two algorithms may depend upon model structure, but it is not immediately apparent why these differences arise.

6.2 Run-Time Versus Problem Size

The estimated run-times r_s are charted on log plots shown in Fig. 9. The multiple model instance problems have been broken out from the random clutter problems. By separating these two cases, the growth trend can be examined for each independently. Within these two problem classes, problems deriving from the six different models have not been distinguished.

Also shown in Fig. 9 are two nonlinear regression curves. These are derived using standard nonlinear regression techniques, as described in [31]. Two alternative statistical models are proposed for how run-time varies as a function of problem size n:

$$\text{Polynomial } r_s = \alpha n^{\beta} \quad \text{Exponential } r_s = \alpha e^{\beta n} \tag{31}$$

The exponential regression comes out as a straight line on these log plots and is easily distinguished from the polynomial regression curve. Qualitatively, it appears the polynomial model is a better fit to the data. Moreover, the normalized coefficients of determination, or R^2 values shown in Table 2 support this interpretation. R^2 measures the proportion of the variation explained by the regression curve. For both algorithms applied to random clutter problems, the R^2 values are substantially higher for the polynomial model. For the SD algorithm on the multiple instance problems, R^2 for the polynomial model is also higher.

The one ambiguous case is the SC algorithm applied to multiple instance problems. Here, each regression model is

TABLE 2
SUMMARY OF RUN-TIME REGRESSION STATISTICS

	Random Clutter			Multiple Instances		
	Poly.		Exp.	Poly.		Exp.
Alg.	R^2	β	R^2	R^2	β	R^2
SD	0.91	2.06	0.76	0.96	2.44	0.82
SC	0.89	1.92	0.69	0.80	2.25	0.79

TABLE 3
R2 VALUES FOR LN(PS) VS. PROBLEM ATTRIBUTES

	m	d	n	ξ	d & n	n & ξ	d & ξ
Random	0.14	0.46	0.30	0.48	0.46	0.56	**0.77**
Multiple	0.36	0.66	0.61	0.35	0.67	0.66	**0.71**

equally bad. The comparatively low R^2 values are indicative of the high problem-to-problem variance in run-time relative to problem size. For SC, some problems are running much faster, and some are not. The side-by-side plots for SD and SC algorithms have the same vertical scaling, so one can see that the most significant difference is the emergence of some much lower run-times on the SC plot.

The exponents β for the polynomial model are also shown in Table 2. For the random clutter cases, these empirical estimates are surprisingly close to the n^2 average case bounds derived for tree search by Grimson [15], [16]. However, note that our random clutter data includes the Dandelion model, which because of its near symmetry would cause tree search great difficulty. For the multiple-instance problems, the growth rate is higher, tending up toward 2.5 rather than 2.0.

6.3 Relating P_s to Problem Attributes

By running many trials on each problem, we have estimated P_s and hence t_s and r_s for *each* problem. While this analysis says much about how local search behaves, it does not address a key problem: How many trials should be run on a novel problem instance. While in general we must leave a detailed study of this issue to future work, we have looked into a number of different models of how t_s might depend upon measurable problem attributes.

The strongest relationship we find is between $\ln(P_s)$ and the number of data segments d. This relationship holds for the SD algorithm. It also appears helpful to define a problem attribute ξ which notes the number of partial symmetries in the model. For the dandelion, $\xi = 16$, for the tree $\xi = 3$, and for all others $\xi = 1$.

Table 3 gives coefficients of determination (R^2 Values) for different combinations of problem attributes and shows that the combination of d and ξ has the highest R^2 values. Three different characterizations of problem size are tested in Table 3: m, the number of model segments, d the number of data segments, and $n = md$ the number of possibly matching pairs of segments.

Recall that the normalized R^2 expresses the percent of variation in the dependent variable $\ln(P_s)$ explained by the independent variables. The combination of d and ξ explains 77 percent of the variation in the random clutter data and 71 percent of the variation in the multiple instance problems. A similar analysis for the SC algorithm was less productive. There is a higher variance in P_s not explained by any of the attributes considered: The highest R^2 value was 0.16 for the random clutter problems and 0.43 for the multiple instance problems.

For the SD algorithm, the regression coefficients give the following relationship between $\ln(P_s)$ and variables d and ξ

$$\begin{aligned} -\ln(P_s) &= 0.0451d + 0.112\xi + 1.072 \\ -\ln(P_s) &= 0.0456d + 0.070\xi + 1.009 \end{aligned} \quad (32)$$

for the random clutter and multiple instance problems respectively. This nonlinear model of how P_s varies with d and ξ along with (25) lets us compute the number of trials our regression model predicts

$$t_{s,rm} = \lceil \log_{Pf} 0.05 \rceil \quad (33)$$

Fig. 10 plots $t_{s,rm}$ versus the actual number of trials required t_s for the 48 test problems.

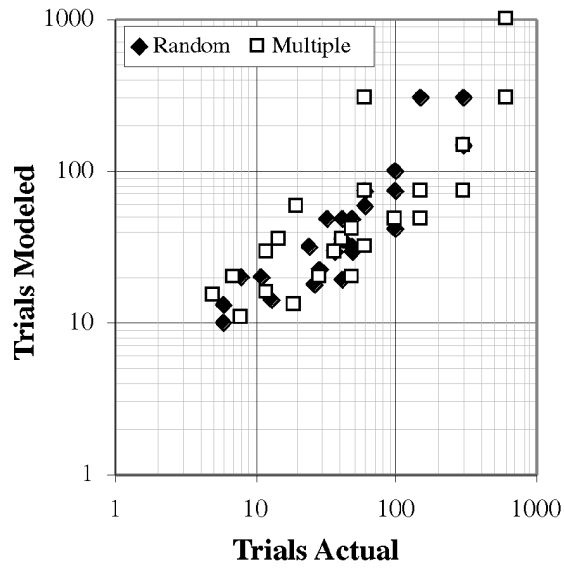


Fig. 10. Trials required actual versus estimates derived from multivariate regression model

Being based upon regression, $t_{s,rm}$ underestimates some cases and overestimates others. A more conservative number of trials may be generated by multiplying $t_{s,rm}$ by a constant. For the 48 problems, $t_{s,rmc} = 3t_{s,rm}$ trials is sufficient to

TABLE 4
RUN-TIMES FOR REAL DATA PROBLEMS AND COMPARISONS

Matching Problem		Statistics				Time	Predicted	
		n	$\hat{P}_s$	t_s	r	r_s	$f_1(n)$	$f_2(n)$
Building, Fig. 2	(No Placement)	1,376	6/100	48	25.1	1,206	680	1,694
Building, Fig. 3	(Placement)	261	28/100	10	4.4	44	28	40
Building, Fig. 3	(No Placement)	1,788	12/1,000	248	178.6	44,293	1,123	3,061
Car, Fig. 4	(Placement)	557	30/100	9	13.5	121	120	219
Car, Fig. 4	(No Placement)	1,701	25/250	28	35.7	1,000	1,021	2,735

Times predicted by regression models.

TABLE 4 LEGEND

n	Number of Model-Data Pairs
$\hat{P}_S$	Times global optimum found out of total trials
t_s	Trials required to find global optimum with 95 percent confidence
r	Average time per trial (seconds)
r_s	Seconds required to solve problem with 95 percent confidence
$f_1(n)$	Seconds predicted by polynomial regression with random clutter
$f_2(n)$	Seconds predicted by polynomial regression with multiple instances

overestimate t_s on all but two problems. To see if using a number of trials that was no longer problem specific changes our observed relationship between n and run-time, we repeated the run-time regression analysis presented in Section 6.2 using $t_{s,rmc}$ in place of t_s.

On the random clutter problems $R^2 = 0.91$ for the polynomial growth model and $R^2 = 0.85$ for the exponential model. For the multiple instance problems, $R^2 = 0.93$ for the polynomial model and $R^2 = 0.82$ for the exponential model. Thus, using $t_{s,rmc}$ slightly lessens the difference between the two models, but the polynomial growth model is still explaining more of the run-time variation relative to n. The exponents on the polynomial model are within 0.01 of those found using t_s.

7 PERFORMANCE ON THE REAL DATA EXAMPLES

Table 4 summarizes how the SC algorithm performs on the matching problems from Figs. 2, 3, and 4. For the examples in Figs. 3 and 4, results are shown both with and without an initial placement estimate for where the model appears in the image.

Table 4 compares run-times needed to solve these specific problems with run-times predicted by the polynomial regression lines shown in Fig. 9. It is perhaps surprising how well these run-times bracket the actual run-times for four out of the five cases. The one exception is the example from Fig. 3 when no placement estimate is available. Exactly why this problem is so hard is not certain. However, a reasonable conjecture is that the run-time scaling as a function of n seen in the test suite should not be expected when n grows solely due to larger numbers of image segments.

8 SOME OBSERVATIONS

The performance of local search as a general method for finding matches appears as good or better than any of the known alternative general methods [16], [17], [32]. However, there are some important caveats. First, while local search probabilistically finds optimal matches, these other techniques deterministically find acceptable (not optimal) matches. Comparison at a coarse level is informative but also somewhat problematic.

Second, while local search does well with clutter, multiple model instances, highly fragmented data, and symmetric models, in its current form on current machines it will not solve problems involving tens of thousands of possibly matching pairs of line segments. In practice, either the complexity of an image must not be excessive, or alternatively, constraints must provide focus of attention within the image.

Finally, we have not yet mentioned anything about methods which use some form of pose indexing to generate match hypotheses and then explore these either in sequence or in parallel. This line of work is useful and important, so let consider briefly how it relates to our local search technique.

8.1 A Comment About Indexing

The strengths and weaknesses of our method come in large part from the lack of any indexing phase. The initial matches are drawn at random from the search space with no attempt to discern or capitalize upon localized structure or domain specific constraints. This lack of reliance upon indexing sets our approach apart from much of the prior work on geometric object recognition and makes our algorithm robust across a wide range of problem types. However, it also places limits on what problems can be solved.

Going back to Roberts [33], there has been a rich tradition of work that says essentially: To find an object, first find a small subset of features that predict the presence of the object. This general approach to recognition can be traced through many works including [22], [34], work on geometric hashing schemes [35], and on through a collection of excellent recent works [36], [37], [38], [39], [40]. Grimson et al. provide a nice general analysis of the problem [41].

The fundamental difficulty in designing indexing algorithms is efficiently finding reliable sets of domain inde-

TABLE 5
TRIALS TO DRAW K GOOD PAIRS VS. TRIALS OF SC LOCAL SEARCH

Problem Description			Trials 95 % Confidence		
			Draw 3	Draw 4	Local Search
Model	g	n	Good Pairs	Good Pairs	Optimal Match
Rectangle (Fig. 1)	6	52	1,948	16,899	8
Tree (Fig. 5)	13	156	5,175	62,118	3
Tree (30 Clutter Lines)	13	516	187,334	7,435,809	2
Building (Fig. 2)	19	1,375	1,135,402	82,167,362	48
Building (Fig. 3)	12	1,788	9,909,727	1,476,549,592	248

pendent indexing features. Hence, indexing is frequently solved using domain specific heuristics. In some application domains, such as 2D part recognition, these heuristics are easily developed. However, they may not generalize across domains—for instance, from polygonal to non-polygonal models.

Random sampling plays such a key role in our approach that it is worth understanding that random sampling alone is not a good way of selecting consistent indexing features. Random sampling to find small subsets of consistent features has been suggested and put to good use under some conditions [42], [43], [44]. However, if for the problems studied in this paper the set of pairs S is partitioned into "good" and "bad" pairs,

$$S = G \cup B \quad \text{and} \quad G \cap B = \varnothing \tag{34}$$

where pairs in G belong to the optimal match, then the probability of drawing k good pairs at random from S is:

$$P = \left(\frac{g}{n}\right)^k \text{ where } g = |G|,\ n = |S| \tag{35}$$

Table 5 lists values of g and n for a sampling of matching problems as well as how many independent random trials it would take to draw subsets of three or four good pairs with 95 percent confidence. The number of trials is determined by inserting the probabilities from (35) into (25). Just drawing *three-tuples* or *four-tuples* at random from S clearly does not scale well over the problems shown. Upwards of millions to billions of trials are needed. For the sake of comparison, the number of trials of *SC local search* are shown in the final column of Table 5.

8.2 Conclusion

Our work adds a new tool to the relatively small set of general matching techniques. Past work has shown our algorithm performs well in several application domains where rough placement constraints derived from other sources are available. This is true for both 2D and 3D [12] recognition problems. The empirical tests presented in this paper suggest local search does well on a wide range of 2D line matching problems even when no initial estimate of model placement is available.

Local search handles many-to-many feature mappings and optimizes global geometric consistency between model and data. by increasing the number random trials, the probability of finding the optimal match may be made arbitrarily high, and through adjusting the number of trials the same algorithm scales between easy and hard problems. Finally, the expected average run-time required to solve a problem appear to grow as n^2 where n is the number of potential pairings of model and image features. This is comparable to, or better than, the run-time of any other known general matching technique.

ACKNOWLEDGMENTS

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

REFERENCES

[1] B.W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *Bell Systems Tech. J.*, vol. 49, pp. 291-307, 1972.

[2] S. Lin and B. Kernighan, "An Effective Heuristic Algorithm for the Traveling Salesman Problem," *Operations Research*, vol. 21, pp. 498-516, 1973.

[3] C.H. Papadimitriou and K. Steiglitz, "Local Search," *Combinatorial Optimization: Algorithms and Complexity*. Englewood Cliffs, N.J.: Prentice-Hall, pp. 454-480, 1982.

[4] J.R. Beveridge, R. Weiss, and E.M. Riseman, "Combinatorial Optimization Applied to Variable Scale 2D Model Matching," *Proc. IEEE Int'l Conf. Pattern Recognition 1990*, Atlantic City, N.J., pp. 18-23, June 1990.

[5] J.R. Beveridge, R. Weiss, and E.M. Riseman, "Optimization of 2-Dimensional Model Matching," *Selected Papers on Automatic Object Recognition*, originally appeared in DARPA Image Understanding Workshop, 1989, H. Nasr, ed., SPIE Milestone Series. Bellingham, Wash., SPIE, 1991.

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

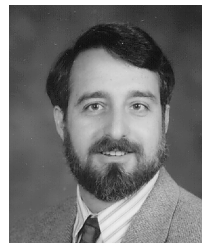
[7] R.T. Collins and J.R. Beveridge, "Matching Perspective Views of Coplanar Structures Using Projective Unwarping and Similarity Matching.," *Proc. 1993 IEEE CS Conf. Computer Vision and Pattern Recognition*, New York, NY, pp. 240-245, June 1993.

[8] C. Fennema, A. Hanson, E. Riseman, J.R. Beveridge, and R. Kumar, "Model-Directed Mobile Robot Navigation," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 20, no. 6, pp. 1,352-1,369, Nov./Dec. 1990.

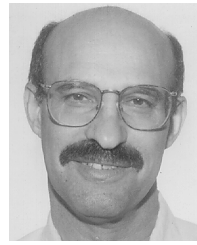
[9] E.M. Riseman, A.R. Hanson, J.R. Beveridge, R. Kumar, and H. Sawhney, "Landmark-Based Navigation and the Acquisition of Environmental Models," *Visual Navigation: From Biological Systems to Unmanned Ground Vehicles*, Y. Aloimonos, ed., pp. 317-374. Lawrence Erlbaum Associates, Inc., 1997.

[10] J.R. Beveridge. C. Graves, and C.E. Lesher, "Local Search as a Tool for Horizon Line Matching," *Proc. Image Understanding Workshop*, Los Altos, Calif., Feb. 1996, pp. 683-686, ARPA. Morgan Kaufmann.

[11] B.A. Draper, *Learning Object Recognition Strategies*, PhD. thesis, Univ. of Massachusetts, Amherst, May 1993.
[12] J.R. Beveridge and E.M. Riseman, "Optimal Geometric Model Matching Under Full 3D Perspective," *Computer Vision and Image Understanding*, vol. 61, no. 3, pp. 351-364, 1995. Short version in *Proc. IEEE Second CAD-Based Vision Workshop*.
[13] D.S. Johnson, C.H. Papadimitriou, and M. Yannakakis, "How Easy Is Local Search?," *J. Computer and System Sciences*, vol. 37, pp. 79-100, 1988.
[14] J.R. Beveridge, E.M. Riseman, and C. Graves, "Demonstrating Polynomial Run-Time Growth for Local Search Matching" *Proc. Int'l Symp. Computer Vision*, Coral Gables, Fla., pp. 533-538, Nov. 1995. Los Alamitos, Calif.: CS Press.
[15] W.E.L. Grimson, "The Combinatorics of Object Recognition in Cluttered Environments Using Constrained Search," *Artificial Intelligence*, vol. 44, no. 1, pp. 121-165, July 1990.
[16] W. Eric and L. Grimson, *Object Recognition by Computer: The Role of Geometric Constraints*. Cambridge, Mass. MIT Press, 1990.
[17] T.A. Cass, "Polynomial-Time Object Recognition in the Presence of Clutter, Occlusion, and Uncertainty," *Proc. Image Understanding Workshop*, San Mateo, Calif., pp. 693-704, Jan. 1992, DARPA. Morgan Kaufmann.
[18] R. Collins, A. Hanson, R. Riseman, and Y. Cheng, "Model Matching and Extension for Automated 3D Site Modeling" *Proc. Image Understanding Workshop*, Los Altos, Calif., Apr. 1993, pp. 197-203, ARPA. Morgan Kaufmann.
[19] J.B. Burns, A.R. Hanson, and E.M. Riseman, "Extracting Straight Lines," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 8, no. 4, pp. 425-456, July 1986.
[20] M.R. Stevens and J.R. Beveridge, "Precise Matching of 3D Target Models to Multisensor Data" *IEEE Trans. Image Processing*, vol. 6, no. 1, pp. 126-142, Jan. 1997.
[21] W.M. Wells III, "Map Model Matching," *Proc. CVPR–91*, pp. 486–492, 1991.
[22] D.G. Lowe, *Perceptual Organization and Visual Recognition*. Kluwer Academic Publishers, 1985.
[23] N. Ayache and O.D. Faugeras, "Hyper: A New Approach for the Recognition and Positioning Of 2D Objects," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 8, no. 1, pp. 44-54, Jan. 1986.
[24] J.R. Beveridge, R. Weiss, and E.M. Riseman, "Optimization of 2-Dimensional Model Matching," *Proc. Image Understanding Workshop*, Los Altos, Calif., pp. 815-830, June 1989, DARPA.: Morgan Kaufmann.
[25] L.S. Davis, "Hierarchical Generalized Hough Transforms and Line-Segment Based Generalized Hough Transforms," *Pattern Recognition*, vol. 15, no. 4, pp. 277-285, 1982.
[26] B. Noble and J.W. Daniel, *Applied Linear Algebra*, 2d. ed. Englewood Cliffs, N.J.: Prentice-Hall, Inc, 1977.
[27] V. Ramesh, *Performance Characterization of Image Understanding Algorithms*, PhD thesis, Univ. of Washington, 1995.
[28] D. Whitley, J.R. Beveridge, C. Graves, and K. Mathias, "Test Driving Three 1995 Genetic Algorithms: New Test Functions and Geometric Matching" *J. Heuristics*, vol. 1, pp. 77-104, 1996.
[29] C.A. Tovey, "Hill Climbing with Multiple Local Optima," *SIAM J. Alg. Disc. Meth.*, vol. 6, no. 3, pp. 384-393, July 1985.
[30] C. Loader, "Local Search Algorithms for 2D Geometric Object Recognition" MS thesis, Univ. of Western Australia, 1995.
[31] Jay L. Devore, "Probability and Statistics for Engineering and the Sciences," *Nonlinear and Multiple Regression*, pp. 459–520. Monterey, Calif.: Brooks/Cole Pub. Co., 1982.
[32] H.S. Baird, *Model-Based Image Matching Using Location*. Cambridge, Mass: MIT Press, 1985.
[33] L.G. Roberts, "Machine Perception of Three–Dimensional Solids," *Optical and Electro-Optical Information Processing*, J.T. Tippett, ed., chapter 9, pp. 159-197. Cambridge, Mass.: MIT Press, 1965.
[34] R.C. Bolles and R.A. Cain, "Recognizing and Locating Partially Visible Objects: The Local-Feature-Focus Method" *Int'l J. of Robotics Research*, vol. 1, no. 3, pp. 57-82, 1982.
[35] Y. Lamdan, J.T. Schwartz, and H.J. Wolfson, "Affine Invariant Model-Based Object Recognition," *IEEE Trans. Robotics and Automation*, vol. 6, no. 5, pp. 578-589, Oct. 1990.
[36] F. Stein and Gérard Medioni, "Recognition of 3D Objects From 2d Groupings," *Proc. Image Understanding Workshop*, San Mateo, Calif., pp. 667-674, Jan. 1992, DARPA. Morgan Kaufmann.
[37] A.R. Pope and D.G. Lowe, "Learning Object Recognition Models From Images," *ICCV*, 1993, pp. 296–301.
[38] A.R. Pope, "Model-Based Object Recognition," Technical Report, Univ. of British Columbia, Jan. 1994.
[39] C.F. Olson, "Time and Space Efficient Pose Clustering," *Proc. CVPR94*, 1994, pp. 251–258.
[40] C.F. Olson, "On the Speed And Accuracy Of Object Recognition When Using Imperfect Grouping," *Proc. SCV95*, 1995, pp. 449–454, http://www.cs.cornell.edu/info/people/clarko/papers.html.
[41] W.E.L. Grimson, D.P. Huttenlocher, and D.W. Jacobs, "A Study of Affine Matching With Bounded Sensor Error" *Proc. IJCV*, vol. 13, no. 1, pp. 7–32, Sept. 1994.
[42] M.A. Fischler and R.C. Bolles, "A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," reprinted in *Readings in Computer Vision*, M.A. Fischler, ed. *Comm. ACM*, vol. 24, no. 6, pp. 381-395, June 1981.
[43] Gerhard Roth, "Extracting Geometric Primitives," *Computer Vision, Graphics, and Image Processing-Image Understanding*, vol. 58, no. 1, pp. 1-22, July 1993.
[44] R. Kumar and A.R. Hanson, "Robust Methods for Estimating Pose And A Sensitivity Analysis," *Proc. CVGIP: Image Understanding*, vol. 11, 1994.



J. Ross Beveridge received his BS degree in applied mechanics and engineering science from the University of California at San Diego in 1980, and his MS and PhD degrees in computer science from the Univ. of Massachusetts in 1987 and 1993, respectively. He has been an assistant professor in the Computer Science Department at Colorado State University since 1993. He is on the editorial board of *Pattern Recognition*. His present interests include object recognition, sensor fusion, image feature extraction, and software development environments for computer vision. He is a member of the ARPA Image Understanding Environment Technical Advisory Committee.



Edward M. Riseman received his BS degree from Clarkson College of Technology in 1964, and his MS and PhD degrees in electrical engineering from Cornell University in 1966 and 1969, respectively. He joined the Computer Science Department as assistant professor in 1969, has been a full professor since 1978, and served as chairman of the department from 1981 to 1985.

Professor Riseman has been director of the Computer Vision Laboratory since its inception in 1975. Recent research projects in the lab include knowledge-based scene interpretation, motion analysis, mobile robot navigation, site model construction for aerial photo interpretation and terrain reconstruction for visualization.

Professor Riseman currently serves on the editorial board for the *International Journal of Computer Vision*, is a senior member of IEEE, and a fellow of the American Association of Artificial Intelligence. He is co-editor of *Computer Vision Systems* (Academic Press, 1978), and is a founder of Amerinex Artificial Intelligence, Inc. (AAI), and Dataview Corporation.