Hierarchical Matching Beats The Non-Wildcard and Interpretation Tree Model Matching Algorithms

Robert B. Fisher
Dept. of Artificial Intelligence, University of Edinburgh
5 Forrest Hill, Edinburgh EH1 2QL, Scotland, United Kingdom

Abstract

In Fisher[1] we introduced a non-wildcard model matching algorithm that has speed advantages over the standard Interpretation Tree model matching algorithm. This paper describes a *hierarchical* model-matching algorithm that has improved performance over both the standard and non-wildcard algorithms.

1 Introduction

The most well-known control algorithm for high-level model matching in computer vision is the *Interpretation Tree*(IT) expansion algorithm, as used by Grimson and Lozano-Perez[2, 3]. In Fisher[1] we introduced a variation on this algorithm that did not use a wildcard which gave performance advantages of 4-10. Both algorithms search a tree of model-to-data correspondences, such that each node in the tree represents one correspondence and the path of nodes from the current node back to the root of the tree is a set of simultaneous pairings. The non-wildcard algorithm avoids the many matches requiring wildcards and only investigates the single model-to-data pairings once, while still exploring the same match search space. It works by extending a set of matches by only adding pairs of matching real model-to-data pairings, rather than also adding pairings that contain wildcards. These algorithms both have the potential for combinatorial search explosion, hence prompting further research into alternative algorithms.

The main cause of the complexity is the re-exploration of the same search subspaces on back-tracking to consider new initial matches. This paper describes a hierarchical model-matching algorithm that results in improved performance over both the standard and non-wildcard algorithms, over a wide range of problem conditions. In the discussion that follows, the term *generated* refers to matches that are hypothesized prior to consistency testing, and *accepted* refers to matches that pass the consistency tests. The following quantities are used:

- There are M model features in the model.
- On average, $p_v M$ of these are visible in the scene (e.g. less than M by occlusion). In 2D scenes, $p_v \doteq 1$ and, in 3D scenes, $p_v \doteq 0.5$ as about half of the features are on the back side of the object and hence not visible.
- Of the visible model features, only p_r of these are recognizable (because of segmentation failures, etc.) forming $C = p_r p_v M$ correct matchable data

Figure 1: Generated and Accepted Nodes versus Number of Model Features (M) with $S = 20$ $p_r = 0.95$ $p_1 = 0.1$ $p_2 = 0.01$ $p_v = 0.5$ $\tau = 0.5$ (loglog plot)

features. (If the model chosen for this scene is incorrect, $p_r = 0$.) Which C of the M model features are matchable is not known initially.

- There are also S spurious data features (e.g. noise features and unrecognizable visible model features). Altogether there are $D = C + S$ data features.
- The probability that a randomly chosen model feature matches with an incorrect random data feature is p_1 (correct pairings alway match).
- The probability that a random pair of model features is consistent with an incorrect random pair of data features (given that the individual model-to-data pairings are consistent) is p_2.
- An acceptable set of model-to-data pairings must have at least $T = \tau p_v M$ non-wildcard correspondences ($\tau \in [0..1]$). Whenever this many are achieved, then the whole matching process terminates successfully immediately. Any set of matches that can never get T matches (because insufficient potential matches remain) is terminated immediately and the matching process proceeds to considering other matches.

2 The Hierarchical Matching Algorithm

Suppose that the $M = K^L$ model features can be decomposed into K^{L-1} submodels each containing K features. Each of these submodels are grouped into K^{L-2} larger submodels containing K^2 features, and so on hierarchically until there is one top-level model containing all K^L model features. The matching algorithm described below generates hypotheses of these submodel types, and only these submodels can be matched together to create hypotheses of the next larger model type.

We describe here a binary submodel hierarchy matching algorithm (i.e. $K = 2$). As an example, consider the simple model hierarchy consisting of four model features A - D organized into two larger submodels AB and CD, which are combined

Figure 2: Generated and Accepted Nodes by Spurious Features (S) with $M = 40$ $p_r = 0.95$ $p_1 = 0.1$ $p_2 = 0.01$ $p_v = 0.5$ $\tau = 0.5$ (loglog plot)

into a larger model $ABCD$. We use a top-down matching strategy (e.g. look for instances of $ABCD$, which recursively looks for instances of AB and CD, etc.). Then, at level λ from the bottom of the model hierarchy, we need only compare the hypotheses that were successfully generated at level $\lambda - 1$.

The algorithm records hypotheses accepted at any given point in the model hierarchy. Thus, if the algorithm needs to backtrack from level $ABCD$ to find a new hypothesis for AB, it is not necessary to later re-explore previously explored portions of the matching space (e.g.) for CD. The algorithm recalls and retries previously verified matches, and then, if there were no successes, generates additional hypotheses starting from where the matching last stopped for this model. This provides a substantial savings over the standard IT algorithm.

At the lowest levels of the hierarchy, many consistent hypotheses are composed mainly of wildcards. Hence, the hierarchical algorithm uses a "largest-subhypothesis-first" search algorithm (i.e. having the most matched non-wildcard data features). The algorithm determines the largest possible hypothesis size, then attempts to generate hypotheses of that size before considering smaller hypotheses. It matches together subhypotheses that may be of different sizes, but tries the largest ones first. The algorithm for pairing hypotheses at this level uses a sensible ordering: (1) all hypotheses that have N matched data features are generated before any hypothesis having $N - 1$ and (2) amongst all hypotheses with the same size, the algorithm generates the hypotheses in the order that keeps the subhypothesis sizes as similar as possible (e.g. it chooses the pair of sizes (3+2) over the pair of sizes (4+1))

At higher levels, when a new subhypothesis is needed, the algorithm may: (1) return a previously generated subhypothesis of the desired size, (2) generate a new subhypothesis that is the same size as that returned at the last call (if there was a previous call), (3) generate the largest possible new subhypothesis of a smaller size or (4) fail when no more subhypotheses are possible.

Figure 3: Generated and Accepted Nodes by Binary Match Probability (p_2) with $M = 40$ $S = 20$ $p_r = 0.95$ $p_1 = 0.1$ $p_v = 0.5$ $\tau = 0.5$ (loglog plot)

When the recursive request for hypotheses reaches the lowest level, (e.g. model feature A in the above example), the algorithm matches the original data features to the original primitive model submodels. New pairings are tested on demand from above, so not all given model-data feature pairings need always be tested before a successful match is found. Consistency is tested using the standard unary feature matching tests. After all possible matches with data features have been attempted, then a match using the wildcard is generated. This promotes filled hypotheses over the proliferation of empty hypotheses.

When testing subhypothesis consistency at level 1 (i.e. pairings involving two model and data features), the standard algorithm's binary feature matching tests are used. At level 2 and higher, consistency is based on the same binary compatibility tests, only tested using primitive model-data feature pairings that come from different subhypotheses.

This algorithm works because, by induction on the previous level, the subhypotheses generated for the two submodels at the next lower level are also recursively generated in a largest-first order, and are then combined in the order that produces the largest hypotheses first. The order of features in the hierarchy is not important to the success of the algorithm, and the algorithm does not require the model to have any natural binary decomposition. However, efficiency is improved if highly likely matches are in the leftmost nodes, thus preventing the algorithm from having to back-track through false starts. Appendix A gives pseudocode for this algorithm and Appendix B gives an example of a matching.

3 The Experiments

To demonstrate the effectiveness of the hierarchical search algorithm, as compared to the non-wildcard and standard algorithms, we use the following simulated experimental problem, based on an example described in Grimson[4]. (A real problem

Figure 4: Generated and Accepted Nodes by Unary Match Probability (p_1) with $M = 40$ $S = 20$ $p_r = 0.95$ $p_2 = 0.01$ $p_v = 0.5$ $\tau = 0.5$ (loglog plot)

follows.) Grimson showed that the model and simulation gave a reasonable characterization of real matching problems. The use of the simulated problems then allows us to compare the algorithm performance on data sets of varying sizes.

Based on the problem model given in Section 2, each model-match experiment of the three algorithms will consist of:

1. Initially determining a random selection of C of the D data features to be the solution.
2. For each generated model-to-data pairing, a correspondence that is not part of the solution and does not use a wildcard is accepted if the new correspondence is individually satisfied with probability p_1 and the new correspondence is pairwise satisfied with each previously filled non-wildcard feature with probability p_2. Correspondences that are part of the solution or use the wildcard are always accepted.

For the experiments described in this paper, we used:

PARAMETER	NOMINAL	RANGE
M	40	5 to 100 by 5
S	20	0 to 100 by 5
p_1	0.1	0.05 to 0.75 by 0.05
p_2	0.01	0.001, 0.002, 0.004, 0.008, 0.01, 0.02 to 0.20 by 0.02, 0.25
τ	0.5	0.2 to 0.9 by 0.1
p_v	0.5	no variation
p_r	0.95	no variation

In each experiment described in this section, one parameter was varied over the range given above and all others were set to the nominal value. All experiments

Figure 5: Generated and Accepted Nodes by Acceptance Threshold (τ) with $M = 40$ $S = 20$ $p_r = 0.95$ $p_1 = 0.1$ $p_2 = 0.01$ $p_v = 0.5$ (loglog plot)

were run 200 times and the value reported is the mean value. The graphs in Figures 1–5 given show how the number of nodes generated and accepted varied with the parameters for the hierarchical, non-wildcard and standard IT algorithms. The results for the non-wildcard algorithm are an improvement on those given in [1], due to improvements in the algorithm. As we look over the results, which explore a substantial portion of the parameter spaces likely to be encountered in visual matching problems, we can see that the hierarchical algorithm is better than the non-wildcard and standard algorithm with respect to the number of nodes searched except when τ becomes large, but the non-wildcard algorithm is better with regards to the number of nodes accepted except for when p_2 becomes large.

When there is no instance of the object in the scene, it is unlikely that the early success conditions would occur, and thus almost all of the search space would have to be explored. In this case, a much greater amount of work is required. In the case of the hierarchical algorithm, about 17 times more work is required to reject a match in a scene, but this is still 4 times better than the standard algorithm (but requires about twice the work of the non-wildcard algorithm).

The algorithms were compared on edges extracted from real test scene similar to those used by Grimson. Because the algorithms are sensitive to data feature order, the algorithms were run 100 times with the model and data features permuted randomly. The effective probabilities in this scene were $p_1 = .235$ and $p_2 = 0.017$ and the number of features were $M = 13$ and $D = 129$. Seven of 13 model edges match true data edges in the test scene using the given tolerances. The average time taken for the matching algorithms on a Sparcstation 1+ was 1.47 seconds for the hierarchical algorithm and 5.88 sec. for the standard algorithm (and 0.96 sec. for the non-wildcard algorithm). The mean number of nodes generated and accepted was 55025 and 1721 for the hierarchical algorithm, 64412 and 845 for the non-wildcard algorithm and 544171 and 39711 for the standard algorithm. On another test scene containing 10 instances of one of these parts, the average times required for a match was hierarchical 21.4 sec., non-wildcard 20.4 sec. and standard 419 sec. The effective probabilities in this scene were $p_1 = .288$ and $p_2 = 0.011$ and the number of features were $M = 28$ and $D = 191$. The extra memory costs of recording the successful submatches in the hierarchical algorithm was about 1M bytes.

4 Discussion and Conclusions

Based on the simulated matchings, the hierarchical algorithm searches about 10 times fewer nodes than the standard algorithm and about one-half the nodes of the non-wildcard algorithm (as the number of features matched grows). However, it is also a much more complicated algorithm and thus the computational costs per search step are higher and it also executes more (expensive) binary tests (53237 *vs* 43682 for the non-wildcard algorithm in the first real-data test) before hypothesis rejection. In the two real examples cited in the previous section, the hierarchical algorithm achieved a performance gain of about 4 to 20 over the standard algorithm, but was comparable in speed to the non-wildcard algorithm. However, based on the simulated results, it is clear that the performance of the algorithms may vary greatly on any one data set. It is also the case that the choice between the hierarchical and non-wildcard algorithms depends on the problem parameters. In particular, when S, p_1 or p_2 are large or τ is small, there are advantages to the hierarchical algorithm.

On the other hand, there are also some general guidelines when the hierarchical algorithm is not as effective as the non-wildcard algorithm. One general principle is "avoid proliferating reasonable hypotheses at the lowest levels of the matching". If we are analyzing a scene with many instances of the same object, or a scene where there are many nearly possible matches, or symmetry, then the hierarchical algorithm will generate many valid matches at the lowest levels, and they will all be explored until the first successful match is found. In a sense, the algorithm is simultaneously finding all matches, of which only one is needed. In addition, the non-wildcard algorithm generally accepts about one-half the number of nodes as the hierarchical algorithm. So, if the costs of processing an accepted node are high, the non-wildcard algorithm is to be favored. However, our experience with the real image data suggests that it is the generation costs, and in particular the costs of the binary feature tests that determine the running speed of the algorithms. Hence, the hierarchical algorithm has clear advantages in this respect.

Acknowledgements

This research was funded by SERC (IED grant GR/F/38310). Other facilities provided by University of Edinburgh. This paper benefited greatly from discussions with D. Borges, A. Fitzgibbon, J. Hallam, H. Hughes, M. Orr, K. Simsarian, M. Waite, M. Trucco and M. Uschold.

References

[1] Fisher, R. B., *Non-Wildcard Matching Beats The Interpretation Tree*, Proc. 1992 British Machine Vision Association Conf., pp 560-569, Leeds, 1992.

[2] Grimson, W. E. L., Lozano-Perez, T., *Model-Based Recognition and Localization from Sparse Range or Tactile Data*, International Journal of Robotics Research, Vol. 3, pp 3-35, 1984.

[3] Grimson, W. E. L., Object Recognition By Computer: The Role of Geometric Constraints, MIT Press, 1990.

[4] Grimson, W. E. L., *The Combinatorics of Heuristic Search Termination for Object Recognition in Cluttered Environments*, Lecture Notes in Computer Science, ECCV-90, Springer-Verlag, pp 552-556, 1990.

A Hierarchical Search Algorithm

```
generatenextbest(treetop)
{
  if all done, return fail
  if treetop is a base level subtree
  { if another untried data feature (generate wildcard last)
    {   increment generated count
        if not wildcard and unary test fails
            then skip this subhypothesis
        record this consistent subhypothesis
        increment accepted count
        return success}
    else return fail}

  // special case of only the left subtree of tree used
  if only left subtree used
  { recurse on left subtree
    if early success or fail then return code
    record subhypothesis for future regeneration
    return success}

  // normal recursive case
  do {
top:
    if first attempt at finding this hypothesis
    {   recurse on left subtree
        if early success or fail then return code
        record subhypothesis for future regeneration
        recurse on right subtree
        if early success or fail then return code
```

```
        record subhypothesis for future regeneration
        reset regeneration pointers
        go to testsection}

    // normal hypothesis generation
    is there a previously generated hyp for slot 1?
    yes, is it the required size?
            yes: go to testsection
            no: smaller, go to getnewslot0
    no: recursively regenerate new slot 1
            early success: return early success
            normal success: record subhypothesis and go to top
            fail: proceed to getnewslot0

getnewslot0:
    is there a previously generated hyp for slot 0?
    yes, is it the required size?
        yes: reset slot 1 list for the current needed size
            and go to testsection
        no: go to getnextvalidsize
    no: recursively regenerate new slot 0
          early success: return early success
          normal success:
             record subhypothesis
             reset slot 1 list for the current needed size
             if no slot 0 of needed size go to getnextvalidsize
             else go to testsection
          fail: proceed to getnextvalidsize

getnextvalidsize: // get new candidate subhypothesis sizes
    do {
        get next smallest hypothesis size
        if no more, then return fail
        if size of both slots negative, then return fail
        if either slot negative, then continue
        if slot 0 size does not exist, then continue
        if slot 1 size does not exist, then continue
        reset slots 0 and 1 for use with this size
    } until a valid new size is found

testsection: // test this hypothesis
    increment generated count
    if neither subhypotheses are completely wildcards
    {   do binary tests between all non-wildcard features
        of the first subhypothesis and all non-wildcard
        features of the second subhypothesis
        if failure, continue}
```

```
    // a consistent match
    record subhypothesis for future regeneration
    increment accepted count
    if enough features matched then return early success
    if cannot match enough features in remaining unfilled slots
        then reject subhypothesis
    return new match
  } forever }
```

B Example of Hierarchical Matching Algorithm

Suppose we have the example $ABCD$ hierarchical model described in Section 2. Suppose also that the model features can make these individual matches against data features a, b, c and the wildcard *. We assume that the early termination criterion requires at least 3 features matched.

Model Feature	Data Features	True Match
A	a, *	a
B	b, *	b
C	c, *	c
D	a, *	*
AB	ab,a*,*b,**	ab
CD	ca,c*,*a,**	c*
ABCD	abc*	abc*

Then, the matching algorithm goes through the following sequence of actions. Steps 1-16 do the initial exploration of the tree through to rejecting a full, but false, hypothesis. Steps 17-21 continue to explore the right subtree, looking for alternatives with the same size, until a wildcard match is found. This leads to a hypothesis with a size smaller than that currently requested. Since no more right subtree hypotheses have the desired size (because they are generated large-to-small and the most recent generation is smaller than requested), the algorithm generates a new left subtree match in steps 22-24. At step 25, the left subtree also has a smaller match than desired, so the algorithm then reduces the size of the desired goal. Steps 26-30 generate the new reduced size match. Here the algorithm starts with previously generated matches of the desired size. At step 31, the full match occurs, and although it is smaller than the requested size (2+2), it also satisfies the termination threshold (3), so the match terminates successfully.

Step	Model Level	Size Goal	Pairings	Action
1	ABCD	(2,2)	-	get left submodel
2	AB	(1,1)	-	get left submodel
3	A	1	Aa	accept pairing (true)
4	AB	(1,1)	-	get right submodel
5	B	1	Ba	reject bad pairing
6	B	1	Bb	accept pairing (true)
7	AB	(1,1)	ABab	accept pairing (true)
8	ABCD	(2,2)	-	get right submodel
9	CD	(1,1)	-	get left submodel
10	C	1	Ca	reject bad pairing
11	C	1	Cb	reject bad pairing
12	C	1	Cc	accept pairing (true)
13	CD	(1,1)	-	get right submodel
14	D	1	Da	accept pairing (bad)
15	CD	(1,1)	CDca	accept pairing (bad)
16	ABCD	(2,2)	ABCDabca	reject bad pairing
17	ABCD	(2,2)	-	get right submodel
18	CD	(1,1)	-	get right submodel
19	D	1	Db	reject bad pairing
20	D	1	Dc	reject bad pairing
21	D	1	D*	accept wildcard pairing
22	CD	(1,1)	CDc*	reduce pairing size attempt rejected
23	CD	(1,1)	-	get left submodel
24	C	1	C*	accept wildcard pairing
25	CD	(1,1)	-	reduce pairing size accepted
			-	(1,0) or (0,1) possible
26	CD	(1,0)	-	restart and generate left
27	C	1	Cc	retrieve old match
28	CD	(1,0)	-	restart and generate right
29	D	0	D*	retrieve old match
30	CD	(1,0)	CDc*	accept pairing (true)
31	ABCD	(2,2)	ABCDabc*	early termination occurs