Interpretation Trees

CS510

Lecture #24

April 23, 2001

Variations on Object Recognition

Problem 1: Is this part of the image an instance of X?
Given a model and given and image region.

Problem 2a: What is this part of the image?
Model search is needed but image region is given.

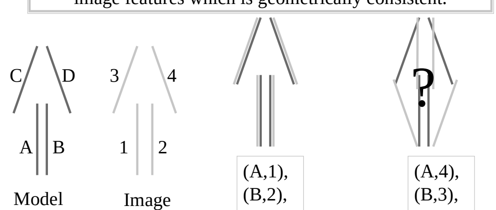
Problem 2b: Are there any instances of X in the image?
Given model, image search is needed.

Problem 4: What objects are we looking at?
Model search and image region search are needed.

Related question, is the model expressed in 2D image space or 3D scene space.

Interpretation Tree Overview

Use tree search to find a mapping of model features to image features which is geometrically consistent.



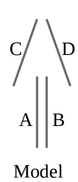
(A,1), (B,2), (C,3), (D,4)

(A,4), (B,3), (C,2), (D,1)

Interpretation Trees

- The model is a set of features with local properties:
  - 2D line segments (length, orientation, contrast…)
  - 2D image regions (avg. intensity, texture, …)
  - 3D line segments (length, orientation, depth…)
- The image data is expressed in the same format
- Binary relations:
  - Constraints between features
    - Between lines: parallel, endpoint near, etc.
    - Between regions: above, inside, etc.

Example of a Model



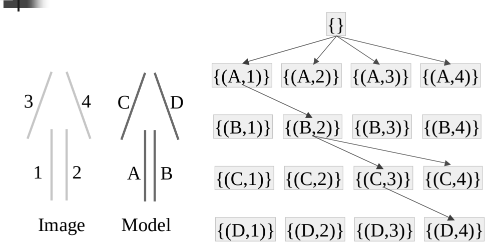
- Four Features
  - Lines A, B, C & D
- Unary Constraints
  - Minimum length for A, B, C & D
- Binary Constraints
  - Parallel: (A,B)
  - Above: (C,A), (D, B)
  - Left-of: (A,B), (C, D)

Question: how precise is this model?

Locally Consistent Interpretation Trees

- Interpretation trees map image features onto model features so as to preserve constraints.
  - To assign image feature 1 to model feature A, 1 and A must be of the same type and their unary features must "match"
  - If 1 is assigned to A, then 2 cannot be assigned to B unless:
    - 2 & B are the same type & match (as above)
    - 1&2 satisfy the binary constraints for A&B

Example of a Tree



Red lines denote assignments that do not allow further assignments

Interpretation Trees (cont.)

- Not every image feature will belong to the model
  - Background "clutter"
  - Feature extraction errors
- Not every model feature will appear in the image
  - (self) Occlusion
  - Feature extraction errors
- The best match is the one with the most correspondences

Generic Interpretation Tree Algorithm (Part 1)

Note: this algorithm is not an exact match to the one in your book (but its close)

- Let Model be the list of model features $\{m_1,\ldots,m_n\}$
- Let Data be the list of image features $\{d_1,\ldots,d_m\}$
- Let Interp be an (initially empty) list of model/data pairs
- Let UnaryP(m_i,d_j) return true iff d_j meets m_i's unary constraints
- Let BinaryP(m_i,d_j,Interp) return true iff the pair (m_i, d_j) is consistent in terms of binary constraints with every pair already in Interp
- List operators (emptyp, destructive pop, non-destructive append)

Generic Interpretation Tree Algorithm (Part 2)

```
InterpTree(Model, Data, Interp) {
  if (emptyp(Model)) return Interp;
  m := pop(Model);
  maxlist = Interp;
  for d in Data do {
    if (UnaryP(m,d) and BinaryP(m,d,Interp)) {
      newlist = InterpTree(Model, Data, append((m,d),Interp));
      if (size(newlist) > size(maxlist)) maxlist := newlist;
  }}
  newlist = InterpTree(Model, Data, Interp);
  if (size(newlist) > size(maxlist)) maxlist := newlist;
  return newlist; }
```


Questions

- Should the recursive call in blue be:
  InterpTree(Model, remove(d, Data), append((m,d),Interp));
- What would the difference be?
- What is the role of the last recursive call (in green)?
- What would be the effect of removing it?
- Is this algorithm guaranteed to terminate?
- How efficient is it?

Observations

- Note the number of combinations tried (worst case)

$$s = 1 + m + m^2 + \ldots + m^n,\ \text{complexity}\,O(m^n)$$

- This can be inverted if model is larger than data (this is rare). The complexity is then: *O(n^m)*
- Pruning based upon geometric constraints is critical!

More Observations

- Eric Grimson has proven polynomial complexity in the average case if:
  - Consider only rotation and translation.
  - The model is guaranteed to be present.
  - No partial symmetries.
- Otherwise, complexity is exponential.

Branch & Bound

- One way to limit the search is "branch & bound"
  - Create a new argument to InterpTree that is the size of the largest interpretation found so far (along any path)
  - If the size of the current interpretation plus the size of the remaining (unmatched) model is less than the current bound, don't recurse.
- Guaranteed never to introduce an error
- On average, prunes search tree (some)
- In the worst case, no faster than previous algorithm

B&B Algorithm

```
int bestsize = 0; // Note that this is global
InterpTree(Model, Data, Interp) {
  if (emptyp(Model)) return Interp;
  if (size(append(Model, Interp)) < bestsize) return NIL;
  m := pop(Model);
  maxlist = Interp;
  for d in Data do {
    if (UnaryP(m,d) and BinaryP(m,d,Interp)) {
      extended_match = append((m,d), Interp);
      bestsize = Max(bestsize, size(extended_match));
      newlist = InterpTree(Model, Data, extended_match));
      if (size(newlist) > size(maxlist)) maxlist := newlist;
    }}
  newlist = InterpTree(Model, Data, Interp);
  if (size(newlist) > size(maxlist)) maxlist := newlist;
  return newlist; }
```


Ullman's Algorithm

CACM '77

- Create an n×m matrix C
  - $C_{ij} = 1 \Rightarrow Data_i$ is consistent with $Model_j$
  - Initialize C using UnaryP(i,j)
- Propagate binary constraints:
  - For every binary relation rel(A,B),
    - $Data_i$ is only consistent with M_a iff it is consistent with some $Data_j$ that is consistent with M_b
    - Otherwise, set entry in C to zero
- Keep Propagating binary constraints until no change in C

Ullman's Algorithm (cont.)

- Preprocess by propagating constraints

  as described on previous slide
- Search as before, except
  - After every model/data binding, repropagate constraints
- Note that branch & bound is consistent with Ullman's algorithm