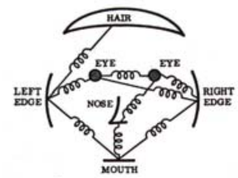
Efficient Matching of Pictorial Structures

01/30/2007

Pushkala Iyer

Pictorial Structures

- "Collection of parts arranged in a deformable configuration"
- Local appearance
  - Part models
  - Parts ≠ feature detection
- Global geometry
  - Not necessarily fully connected graph
- Joint optimization
  - Combine appearance and geometry without hard constraints
    - "Stretch and fit"
    - Qualitative



Sparse representation

+ Computationally tractable (10^5 pixels → 10^1 -- 10^2 parts)
+ Generative representation of class
+ Avoid modeling global variability

- Throw away most image information
- Parts need to be distinctive to separate from other classes

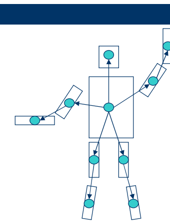
History of related work

- Fischler and Elschlager original 1973 paper
- Burl, Weber and Perona ECCV 1998
  - Probabilistic formulation
  - Full joint Gaussian spatial model
  - Computational challenges led to feature-based
- Felzenszwalb and Huttenlocher CVPR 2000
  - Explicit revisiting of FE73 for trees
  - Probabilistic MAP estimates
  - Efficient algorithms using distance transforms

The Matching Problem

- Find the best placement of parts in an image
  - How well does each part match the image ?
  - How well do all they all fit together ?
- Minimize a certain energy function

Matching Problem



The Solution Approach

- Pictorial Structure model [EF73]
- Restrictions on relationships
  - Tree structure
  - Natural skeletal structure of many animate objects
  - Dynamic programming
- Pairwise relationships
  - Broad range of objects
  - Generalized Distance Transforms
- Globally best match of generic objects
- FH2000 vs other approaches
  - Perona et al – central coordinate system, limited to one articulation point.
  - No hard decisions.
  - Valid configurations are not treated as being equally good.

Recognition Framework Model

- Graph Model G = (V, E)
  - Parts are the vertices $V = \{v_1, v_2, \ldots v_n\}$
  - If v_i, v_j are connected, then $(v_i, v_j) \in E$.
- Instance of a part in an image specified by location l.
  - Position, Rotation, Scale for 2D parts.
- Match cost function $m_i(I, l)$ measures how well the part matches the image I when placed at location l.
- Deformation cost function $d_{ij}(l_i, l_j)$ for every edge (v_i, v_j) measures how well the locations l_i of v_i and l_j of v_j agree with the object model.

Model Framework

- A configuration $L = (l_1, l_2, \ldots, l_n)$ specifies a location for each of the parts v_i in V w.r.t the image.
- Best configuration is the configuration that minimizes the total cost: match cost of individual parts + pair wise cost of the connected pairs of parts.
- $L^* = \arg\min_L \left(\sum_{(v_i,v_j) \in E} d_{ij}(l_i,l_j) + \sum_{v_i \in V} m_i(I,l_i)\right)$

Problem reduction

- Minimization of $L^* = \arg\min_L \left(\sum_{(v_i,v_j) \in E} d_{ij}(l_i,l_j) + \sum_{v_i \in V} m_i(I,l_i)\right)$ is $O(m^n)$
  - Where m is the number of discrete values for each l_i and n is the number of vertices in the graph.
  - Markov Random Fields, Dynamic Contour Models (snakes).
- Restricted graphs reduce time complexity
  - For first order snakes (chain) reduces to $O(m^2n)$ from $O(m^n)$
  - Dynamic programming - is a method of solving problems exhibiting the properties of overlapping subproblems and optimal substructure in a way better than naïve methods.(Wikipedia)
  - Memoization and bottom-up approach.
  - Tree structured graphs enable similar reduction to be achieved.

Problem Reduction

- $O(m^2n)$ algorithm is not practical – large number of possible locations for each part.
- Restriction on pairwise cost function dij yields a minimization algorithm that is O(mn).
- $d_{ij}(l_i,l_j) = \| T_{ij}(l_i) - T_{ji}(l_j) \|$
  - d_{ij} measures the degree of deformation.
  - Is restricted to be a Norm.
  - A **norm** is a function which assigns a positive *length* or *size* to all vectors in a vector space, other than the zero vector. (wikipedia)
  - T_{ij} and T_{ji} should be invertible, together capture the ideal relative configurations of parts v_i and v_j.
  - $T_{ij}(l_i) = T_{ji}(l_j)$ => l_i and l_j are ideal locations for v_i and v_j
  - $T_{ji}(l_j)$ should be discretized in a grid.

Efficient Minimization

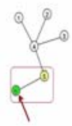
- Dynamic programming to find the configuration $L^* = (l_1^*, \ldots, l_n^*)$ that minimizes the cost.
- Computation involves n-1 functions, each of which specifying the best location of one part w.r.t the possible locations of another part.

Efficient Minimization

- The best location of a leaf node v_j (6) can be computed as a function of the location of just its parent v_i (5).
- Only contribution of l_j to the energy is $d_{ij}(l_i,l_j) + m_j(I,l_j)$ – the contribution of the edge (5,6) and the position of 6.
- Best location of v_j given location l_i of v_i is

  $B_j(l_i) = \min_{lj} (d_{ij}(l_i,l_j) + m_j(I, l_j))$

- Replacing min by argmin, we get the best location of v_j as a function of the location l_i of its parent v_i.

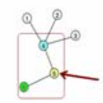


Efficient Minimization

- For non leaf vertices $v_j \neq v_r$, if $B_c(l_j)$ is known for every child v_c ε of C_j, then the best location of v_j given its parent v_i is

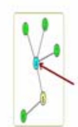
  $B_j(l_i) =$
  $\min_{lj} (d_{ij}(l_i, l_j) + m_j(I,l_j) + \sum_{vc\, \epsilon\, Cj} B_c(l_j)$

- Replacing min with arg-min yeilds the best location of v_j as a function of l_i.



Efficient Minimization

- For the root node v_r, if $B_c(l_r)$ is known for every child v_c ε C_r, then the best location of the root is
  $L_r^* =$
  $\arg\min_{lr} (m_r(I, l_r) + \sum_{vc\, \epsilon\, Cr} B_c(l_j))$

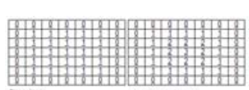


Algorithm

- Recursive equations specify an algorithm.
- For every leaf node, compute its best location as a function of the location of its parent.
- For every non leaf node X, compute its best location as a function of the location of X's parent, also taking into consideration the cost of placing X's children (previous step).
- Repeat until the best location of the root is calculated.
- Now traverse the tree starting at the root, to find the optimum configuration.
- O(nM) – n (# nodes) M (time to compute $B_j(l_i)$ and $B'_j(l_i)$)

Distance Transforms

- A **distance transform**, also known as **distance map** or **distance field**, is a representation of a digital image. (Wikipedia)
- The map supplies each pixel of the image with the distance to the nearest *obstacle pixel*. A most common type obstacle pixel is a *boundary pixel* in a binary image.
- An example of a chessboard distance transform on a binary image.



Generalized Distance Transforms

- $D_B(Z) = \min_{w\, \epsilon\, B} \| z- w \|$ where B is a subset of G
- $D_B(Z) = \min_{w\, \epsilon\, G} (\| z- w \| + 1_B (w))$ where $1_B (w)$ is an indicator function for membership in B.

- Algorithm (G.Borgefors) computes this in O(mD) time for a D dimensional grid.
  - Two pass, local neighborhoods of 7x7 pixels used.
- Meijster, Roerdink & Hesselink:
  - Generic distance transform algorithm in linear time.
  - 2 Phases, first columnwise, second rowwise, each phase 2 scans.
  - Per row computation independent of per column computation, can be parallelized.

- $D_f(Z) = \min_{w\, \epsilon\, G} (\| z- w \| + f(w))$

- How that helps:

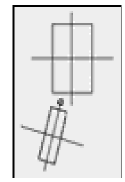
- Given $d_{ij} (l_i,l_j) = \| T_{ij}(l_i) – T_{ji}(l_j) \|$
- $B_j(l_i) = D_f(T_{ij}(l_i))$ where $f(w) = m_j(I, T_{ji}^{-1}(w)) +\sum T_{ji}^{-1}(w))$

Computation

- D is 4 (x, y, rotation, scale)
- $D[x,y,\theta,s]$ is initialized to the values of function f(w)
- $D[x,y,\theta,s] = \min(D[x, y, \theta,s],$
  $D[x-1, y, \theta, s] + k_x,$
  $D[x, y-1, \theta, s] + k_y,$
  $D[x, y, \theta-1, s] + k_\theta,$
  $D[x, y, \theta, s-1] + k_s)$
- $D[x,y,\theta,s] = \min(D[x, y, \theta,s],$
  $D[x+1, y, \theta, s] + k_x,$
  $D[x, y+1, \theta, s] + k_y,$
  $D[x, y, \theta+1, s] + k_\theta,$
  $D[x, y, \theta, s+1] + k_s)$
- Doesn't consider periodic θ.
- Special handling of boundary cases, additional passes.
- $B_j(l_i)$ computable in O(m) time.

Person Model

- Flexible revolute joints.
- Ideal relative orientation given by θ_{ij}.
- Deformation cost measures observed deviation from ideal.
- Given the observed locations $l_i = (\theta_i, s_i, x_i, y_i)$ and $l_j = (\theta_j, s_j, x_j, y_j)$
- $d_{ij}(l_i, l_j) = w_{ij}^{\theta} |(\theta_j - \theta_i) - \theta_{ij}|$
  $+ w_{ij}^{s} |(\log s_j - \log s_i) - \log s_{ij}|$
  $+ w_{ij}^{x} | x'_{ij} - x'_{ji}|$
  $+ w_{ij}^{y} |y'_{ij} - y'_{ji}|$
- Large w_{ij}^{s}, w_{ij}^{x}, w_{ij}^{y} and Small w_{ij}^{θ}



Recognition Results

Car Model

- Flexible prismatic joints
- $d_{ij}(l_i, l_j) =$
  Infinity $|(\theta_j - \theta_i)|$
  $+ w_{ij}^{s} |(\log s_j - \log s_i) - \log s_{ij}|$
  $+ w_{ij}^{x} | x'_{ij} - x'_{ji}|$
  $+ w_{ij}^{y} |y'_{ij} - y'_{ji}|$

Bayesian Formulation

- Best match by MAP estimation
  $L^* = \arg\max_L (\Pr(L|I))$
- Applying Bayes rule,
  $L^* = \arg\max_L (\Pr(I|L) \Pr(L))$
- Prior information – from spring connections
- Likelihood – approx product of match qualities for individual parts.
- To minimize the energy function, take the negative logarithm:
  $L^* = \arg\min_L (\sum_{(vi,vj) \in E} d_{ij} (l_i,l_j) - \sum_{vi \in V} \ln g_i(I,l_i))$

Summary

- No decisions until the end.
  - No feature detection
    - Quality maps or likelihoods
  - No hard geometric constraints
    - Deformation costs or priors
- Efficient algorithms.
  - Dynamic programming critical
  - Not applicable to all problems, need good factorizations of geometry and appearance
- Good for categorical object recognition.
  - Qualitative descriptions of appearance
  - Factoring variability in appearance and geometry
- Deals well with occlusion.
  - In contrast to hard feature detection
- Most applicable to 2D objects defined by relatively small number of parts.
- Unclear how to extend to large number of transformation parameters per part.
  - Explicit representation grows exponentially
- No known way of using to index into model databases.

References

- Efficient Matching of Pictorial Structures – Pedro F Felzenswalb & Daniel P Huttenlocher
- Representation & Matching of Pictorial Structures – Martin A Fischler & Robert A Elschlager
- Discussion of Pictorial Structures - Pedro F Felzenswalb & Daniel P Huttenlocher
- A general algorithm for computing distance transforms in linear time – A Meijster, J B T M Roerdink and W H Hesselink
- Part Based Models – Rob Fergus
- Wikipedia