Dynamic Programming
For Detection
Fast Detection

- For example finding faces at video rates
Dynamic Programming (DP)

- General algorithmic technique
  - Not specific algorithm
  - Analogous to “divide and conquer” – bottom up

- Methods that cache solutions to sub-problems rather than re-computing them
  - E.g., Fibonacci, substring matching

- Applies to problems that can be decomposed into sequence of stages
  - Each stage expressed in terms of results of fixed number of previous stages
Simple DP Example: Box Sum

- Sum n-vector over sliding k-window
  - $W_k[x] = f[x] + \ldots + f[x+k]$  
  - Note: often k odd, sum between $x \pm (k-1)/2$

- Explicit summation $O(k*n)$ additions
- Recurrence yields $O(n+k)$ time method
  - $W_k[x] = W_k[x-1] + f[x+k] - f[x-1]$  
  - Each element of sum differs from previous by just two values
Box Sums in d Dimensions

- One pass along each dimension
  - Sum intermediate result from previous pass
  - 2D case: horizontal then vertical (or vice versa)
    - m by n image, $O(mn+wh)$ time vs. $O(mnwh)$
    - E.g., 10 by 10 summation window, 100x faster
1d Integral Images

- Fast summations over different sized regions (non spatially uniform)
- Cumulative sum
  - \( S[x] = f[0] + \ldots + f[x] \)
- DP recurrence O(n) time
  - \( S[x] = S[x-1] + f[x] \)
- Sum over window of \( f[x] \) independent of size \( k \)
  - \( W_k[x] = S[x+k-1] - S[k-1] \)
n-d Integral Images

- Analogous for higher dimensions, 2D:
  - \( S[x,y] = f[0,0] + \ldots + f[0,y] + \ldots \)
  - \( f[x,0] + \ldots + f[x,y] \)

- Separate recurrence per dimension
  - \( C[x,y] = C[x,y-1] + f[x,y] \) (column sum)
  - \( S[x,y] = S[x-1,y] + C[x,y] \) (total sum)
  - Or alternatively row sum then total sum
Fast Region Sums With II

- Sum over a rectangle, constant time
  - $S[b_r] + S[t_l-(1,1)] - S[b_l-(1,0)] - S[t_r-(0,1)]$

- Sum over arbitrary region, linear time
  - Running time proportional to length of boundary not area
Fast Detection With II

- Features formed from combinations of sums over rectangles
  - For example positive and negative regions
  - Running time independent of rectangle size
- Viola and Jones use for face detection at approximately video rates
Fast Detection With II

- Also useful for arbitrary shaped regions
  - Decompose into rectangles
    - With no holes in worst case this is number of scan lines (not too bad with holes either)
    - Proportional to boundary length rather than area
  - Construct chain-code representation of boundary and sum values
    - Positive for downward links and negative for upward (reverse for holes)
  - Note relation to work of Jermyn and Ishikawa on boundary integrals
Distance Transforms

- Map of distance to nearest features
  - Computed from map of feature locations
    • E.g., edge detector output

- Powerful and widely applicable
  - Can think of as “smoothing in feature space”
  - Related to morphological dilation operation
  - Often preferable to explicitly searching for correspondences of features

- Efficiently computable using DP
  - Time linear in number of pixels, fast in practice
Distance Transform Definition

- Set of points, $P$, some distance $\| \bullet \|$
  \[ D_P(x) = \min_{y \in P} \| x - y \| \]
  - For each location $x$ distance to nearest $y$ in $P$
  - Think of as cones rooted at each point of $P$
- Commonly computed on a grid $\Gamma$ using
  \[ D_P(x) = \min_{y \in \Gamma} ( \| x - y \| + 1_{P}(y) ) \]
  - Where $1_{P}(y) = 0$ when $y \in P$, $\infty$ otherwise
DP for $L_1$ Distance Transform

- 1D case
  - Two passes:
    - Find closest point on left
    - Find closest on right if closer than one on left
  - Incremental:
    - Moving left-to-right, closest point on left either previous closest point or current point
    - Analogous moving right-to-left for closest point on right
  - Can keep track of closest point as well as distance to it
    - Will illustrate distance; point follows easily
L₁ Distance Transform Algorithm

- Two pass O(n) algorithm for 1D L₁ norm (for simplicity just distance)

1. **Initialize**: For all j
   \[ D[j] \leftarrow 1_p[j] \]

2. **Forward**: For j from 1 up to n-1
   \[ D[j] \leftarrow \min(D[j], D[j-1] + 1) \]

3. **Backward**: For j from n-2 down to 0
   \[ D[j] \leftarrow \min(D[j], D[j+1] + 1) \]

![Diagram of L₁ Distance Transform Algorithm]
**L₁ Distance Transform**

- 2D case analogous to 1D
  - Initialization
  - Forward and backward pass
    - Fwd pass finds closest above and to left
    - Bwd pass finds closest below and to right

- Note nothing depends on \(0, \infty\) form of initialization
  - Can “distance transform” arbitrary array

![Distance Transform Diagram](image-url)
L₂ Distance Transform

- Approximations using fixed size masks
  - Analogous to L₁ case
  - Simple to understand but not best methods

- Exact linear time method for L₂²
  - Can compute sqrt (but usually not needed)
  - Fast in practice, easy to implement
  - Harder to understand than L₁ algorithm
  - Uses important general algorithmic technique of amortized analysis

- 1D case – lower envelope of quadratics
1D $L_2^2$ Distance Transform

- Single left-to-right pass
  - Adding k-th quadratic to lower envelope (LE) of first k-1 quadratics
  - Quadratics differ only in location of their base

- Concerned about intersection of k-th quadratic and LE of first k-1
  - Consider only rightmost quadratic visible in LE
  - Keep track of locations of bases of *visible quadratics* (VQ), ordered left-to-right
  - Keep track of *visible intersections* of adjacent quadratics (VI), ordered left-to-right
Adding k-th Quadratic to LE

- Case 1: intersection of k and rightmost VQ (RVQ) outside range, k not visible on LE
- Case 2: intersection of k and RVQ to right of rightmost VI (RVI), k added to right
- Case 3: intersection of k and RVQ to left of RVI, k covers at least RVQ, remove RVQ and try adding again
Running Time of 1D Algorithm

- Traditional analysis would consider time for each case, multiplied by n iterations
  - Cases 1 and 2 $O(1)$, but case 3 ??
- **Amortized analysis**: charge work done by algorithm to “events” that can be bounded
  - Three event types
    - K-th quadratic initially excluded
    - K-th quadratic added
    - K-th quadratic removed
  - Each event happens at most once per quadratic (note once removed, never again)
  - Algorithm does constant work per event
2D Algorithm

- Horizontal pass of 1D algorithm
  - Computes minimum $x^2$ distance
- Vertical pass of 1D algorithm on result of horizontal pass
  - Computes minimum $x^2 + y^2$ distance
  - Note algorithm applies to any input (quadratics can be at any location)
- Actual code straightforward and fast
  - Each pass maintains arrays of indexes of visible parabolas and the intersections
  - Fills in distance values at each pixel after determining which parabolas visible
Horizontal Pass of 2D $L_2^2$ DT

for (y = 0; y < height; y++) {
    k = 0; /* Number of boundaries between parabolas */
    z[0] = 0; /* Indexes of locations of boundaries */
    z[1] = width; /* No current boundaries (first at end of array) */
    v[0] = 0; /* Indexes of locations of visible parabola bases */
    for (x = 1; x < width; x++) {
        do {
            /* intersection of this parabola with rightmost visible parabola */
            s = ((imRef(im, x, y) + x*x) - (imRef(im, v[k], y) + v[k]*v[k])) / 
                (2 * (x - v[k]));
            sp = ceil(s);
            /* case one: intersection off end, this parabola not visible */
            if (sp >= width)
                break;
            /* case two: intersection is rightmost, add it to end*/
            if (sp > z[k]) {
                z[k+1] = sp; z[k+2] = width; v[k+1] = x; k++;
                break; }
            /* case three: intersection is not rightmost, hides rightmost 
             parabola and perhaps others, remove rightmost and try again */
            if (k == 0) {
                v[0] = x; break;
            } else {
                z[k] = width; k--; }
        } while (1);
    }
}
DT Values From Intersections

/* get value of input image at each parabola base */
for (x = 0; x <= k; x++) {
    vref[x] = imRef(im, v[x], y);
}
k = 0;
/* iterate over pixels, calculating value for closest parabola */
for (x = 0; x < width; x++) {
    if (x == z[k+1])
        k++;
    imRef(im, x, y) = vref[k] + (v[k]-x)*(v[k]-x);
}

- No reason to approximate L2 distance!
- Code available at www.cs.cornell.edu/~dph/matchalgs/
DT and Morphological Dilation

- Dilation operation replaces each point of P with some fixed point set Q
  - $P \oplus Q = U_p U_q \, p+q$
- Dilation by a “disc” $C^d$ of radius d replaces each point with a disc
  - A point is in the dilation of P by $C^d$ exactly when the distance transform value is no more than d (for appropriate disc and distance fcn.)
  - $x \in P \oplus C^d \iff D_P(x) \leq d$
Generalizations of DT

- Combination distance functions
  - Robust “truncated quadratic” distance
    - Quadratic for small distances, linear for larger
    - Simply minimum of (weighted) quadratic and linear distance transforms

- DT of arbitrary functions: \( \min_y \| x - y \| + f(y) \)
  - Exact same algorithms apply
  - Combination of cost function \( f(y) \) at each location and distance function
    - Useful for certain energy minimization problems
Distance Transforms in Matching
Distance Transforms in Matching

- **Chamfer measure** – asymmetric
  - Sum of distance transform values
    - “Probe” DT at locations specified by model and sum resulting values

- **Hausdorff distance** (and generalizations)
  - Max-min distance which can be computed efficiently using distance transform
  - Generalization to quantile of distance transform values more useful in practice

- **Iterated closest point (ICP) like methods**
  - Traditionally search for matches, DT faster
Hausdorff Distance

- Classical definition
  - Directed distance (not symmetric)
    \[ h(A,B) = \max_{a \in A} \min_{b \in B} \| a - b \| \]
  - Distance (symmetry)
    \[ H(A,B) = \max(h(A,B), h(B,A)) \]

- Minimization term is simply a distance transform of B
  - \[ h(A,B) = \max_{a \in A} D_B(a) \]
  - Maximize over selected values of DT

- Classical distance not robust, single “bad match” dominates value
Hausdorff Matching

- Partial (or fractional) Hausdorff distance to address robustness to outliers
  - Rank rather than maximum
    - \( h_k(A, B) = \text{rank}_{a \in A} \min_{b \in B} \| a - b \| = \text{rank}_{a \in A} D_B(a) \)
  - K-th largest value of \( D_B \) at locations given by \( A \)
  - Often specify as fraction \( f \) rather than rank
    - \( 0.5 \), median of distances; \( 0.75 \), 75\(^{\text{th}} \) percentile

\[
\begin{array}{cccccccccccc}
1,1,2,2,3,3,3,3,4,4,5,12,14,15 \\
.25 & .5 & .75 & 1.0
\end{array}
\]
Hausdorff Matching

- Best match
  - Minimum fractional Hausdorff distance over given space of transformations

- Good matches
  - Above some fraction (rank) and/or below some distance

- Each point in (quantized) transformation space defines a distance
  - Search over transformation space
    - Efficient branch-and-bound “pruning” to skip transformations that cannot be good
Fast Hausdorff Search

- Branch and bound hierarchical search of transformation space
- Consider 2D transformation space of translation in x and y
  - (Fractional) Hausdorff distance cannot change faster than linearly with translation
    - Similar constraints for other transformations
  - Quad-tree decomposition, compute distance for transform at center of each cell
    - If larger than cell half-width, rule out cell
    - Otherwise subdivide cell and consider children
Branch and Bound Illustration

- Guaranteed (or admissible) search heuristic
  - Bound on how good answer could be in unexplored region
    - Cannot miss an answer
  - In worst case won’t rule anything out
- In practice rule out vast majority of transformations
  - Can use even simpler tests than computing distance at cell center
DT Based Matching Measures

- Fractional Hausdorff distance
  - Kth largest value selected from DT

- Chamfer
  - Sum of values selected from DT
    - Suffers from same robustness problems as classical Hausdorff distance
    - Max intuitively worse but sum also bad
  - Robust variants
    - Trimmed: sum the K smallest distances (same as Hausdorff but sum rather than largest of K)
    - Truncated: truncate individual distances before summing
Comparing DT Based Measures

- Monte Carlo experiments with known object location and synthetic clutter
  - Matching edge locations
- Varying percent clutter
  - Probability of edge pixel 2.5-15%
- Varying occlusion
  - Single missing interval, 10-25% of boundary
- Search over location, scale, orientation
ROC Curves

- Probability of false alarm vs. detection
  - 10% and 15% occlusion with 5% clutter
  - Chamfer is lowest, Hausdorff (f=0.8) is highest
  - Chamfer truncated distance better than trimmed

![ROC Curves Diagram]
Edge Orientation Information

- Match edge orientation as well as location
  - Edge normals or gradient direction
- Increases detection performance and speeds up matching
  - Better able to discriminate object from clutter
  - Better able to eliminate cells in branch and bound search
- Distance in 3D feature space \([p_x, p_y, \alpha p_o]\)
  - \(\alpha\) weights orientation versus location
  - \(k_{th_{a \in A}} \min_{b \in B} \| a-b \| = k_{th_{a \in A}} D_B(a)\)
ROC’s for Oriented Edge Pixels

- Vast improvement for moderate clutter
  - Images with 5% randomly generated contours
  - Good for 20-25% occlusion rather than 2-5%
Observations on DT Based Matching

- Fast compared to explicitly considering pairs of model and data features
  - Hierarchical search over transformation space
- Important to use robust distance
  - Straight Chamfer very sensitive to outliers
    - Truncated DT can be computed fast
- No reason to use approximate DT
  - Fast exact method for $L_2^2$ or truncated $L_2^2$
- For edge features use orientation too
  - Comparing normals or using multiple edge maps
Template Clustering

- Cluster templates into tree structures to speed matching
  - Rule out multiple templates simultaneously
    - Coarse-to-fine search where coarse granularity can rule out many templates
    - Several variants: Olson, Gavrila, Stenger
- Applies to variety of DT based matching measures
  - Chamfer, Hausdorff and robust Chamfer
- Use hierarchical clustering techniques (e.g., Edelsbrunner) offline on templates
Example Hierarchical Clusters

Larger pairwise differences higher in tree
Hausdorff and Linear Halfspaces
Dilate and Correlate Matching

- Fixed degree of “smoothing” of features
  - Dilate binary feature map with specific radius disc rather than all radii as in DT
- \( h_k(A,B) \leq d \iff |A \cap B^d| \geq k \)
  - At least k points of A contained in \( B^d \)
- For low dimensional transformations such as x-y-translation best way to compute
  - Dilation and binary correlation are very fast
  - For higher dimensional cases hierarchical search using DT is faster
Dot Product Formulation

- Let $A$ and $B^d$ be (binary) vector representations of $A$ and $B$
  - E.g. standard scan line order
- Then fractional Hausdorff distance can be expressed as dot product
  - $h_k(A, B) \leq d \iff A \cdot B^d \geq k$
- Note that if $B$ is perturbation of $A$ by $d$
  then $A \cdot B$ is arbitrary whereas $A \cdot B^d = A \cdot A$
- Hausdorff matching using linear subspaces
  - Eigenspace, PCA, etc.
Learning and Hausdorff Distance

- Learning linear half spaces
  - Dot product formulation defines linear threshold function
    - Positive if $A \cdot B^d \geq k$, negative otherwise
- PAC – probably approximately correct
  - Learning concepts that with high probability have low error
  - Linear programming and perceptrons can both be used to learn half spaces in PAC sense
- Consider small number of values for $d$ (dilation parameter) and pick best
Illustration of Linear Halfspace

- Possible images define n-dimensional binary space
- Linear function separating positive and negative examples
Perceptron Algorithm

- Examples $x_i$ each with label $y_i \in \{+,-\}$
- Set initial prediction vector $v$ to 0
- For $i=1, \ldots, m$
  - If $\text{sign}(v \cdot x_i) \neq \text{sign}(y_i)$
    then $v = v + y_i x_i$
- Run repeatedly until no misclassifications on $m$ training examples
  - Or less than some threshold number but then haven’t found linear separator
- Generally need many more negative than positive examples for effective training
Learned Half-Space Templates

Positive examples (500)

Negative examples (350,000)

All Model Coefs.  Pos. Model Coefs.

Example Model (dilation d=3, picked automatically)
Detection Results

- Train on 80% test on 20% of data
  - No trials yielded any false positives
  - Average 3% missed detections, worst case 5%
Spatial Continuity

- Hausdorff and Chamfer matching do not measure degree of connectivity
  - E.g., edge chains versus isolated points
- Spatially coherent matching approach
  - Separate features into three subsets
    - Matchable
      - Near image features
    - Boundary
      - Matchable but near un-matchable
    - Un-matchable
      - Far from image features
Flexible Templates
Flexible Template Matching

- Pictorial structures
  - Parts connected by springs and appearance models for each part
  - Used for human bodies, faces
  - Fischler & Elschlager, 1973 – considerable recent work
Formal Definition of Model

- Set of parts $V = \{v_1, \ldots, v_n\}$
- Configuration $L = (l_1, \ldots, l_n)$
  - Specifying locations of the parts
- Appearance parameters $A = (a_1, \ldots, a_n)$
  - Model for each part
- Edge $e_{ij}, (v_i, v_j) \in E$ for connected parts
  - Explicit dependency between part locations $l_i, l_j$
- Connection parameters $C = \{c_{ij} \mid e_{ij} \in E\}$
  - Spring parameters for each pair of connected parts
Flexible Template Algorithms

- Difficulty depends on structure of graph
  - Which parts are connected (E) and how (C)
- General case exponential time
  - Consider special case in which parts translate with respect to common origin
    - E.g., useful for faces

  - Distinguished central part \( v_1 \)
  - Spring \( c_{i1} \) connecting \( v_i \) to \( v_1 \)
  - Quadratic cost for spring
Efficient Algorithm for Central Part

- Location $L=(l_1, ..., l_n)$ specifies where each part positioned in image
- Best location $\min_L (\sum_i m_i(l_i) + d_i(l_i,l_1))$
  - Part cost $m_i(l_i)$
    - Measures degree of mismatch of appearance $a_i$ when part $v_i$ placed at location $l_i$
  - Deformation cost $d_i(l_i,l_1)$
    - Spring cost $c_{i1}$ of part $v_i$ measured with respect to central part $v_1$
    - E.g., quadratic or truncated quadratic function
    - Note deformation cost zero for part $v_1$ (wrt self)
Express as Kind of DT

- \( \min_L (\Sigma_i (m_i(l_i) + d_i(l_i,l_1))) \)
- \( \min_L (\Sigma_i m_i(l_i) + \|l_i - T_i(l_1)\|^2) \)
  - Quadratic distance between location of part \( v_i \) and ideal location given location of central part
- \( \min_{l_1} (m_1(l_1)) + \Sigma_{i>1} \min_{l_i} (m_i(l_i) + \|l_i - T_i(l_1)\|^2) \)
  - i-th term of sum minimizes only over \( l_i \)
- \( \min_{l_1} (m_1(l_1) + \Sigma_{i>1} D_{mi}(T_i(l_1))) \)
  - Each term of sum is distance transform of the match cost function \( m_i \)
    - \( D_f(x) = \min_y (f(y) + \|y-x\|^2) \), using same algorithms as before
Application to Face Detection

- Five parts: eyes, tip of nose, sides of mouth
- Each part a local image patch ($m_i$)
  - Represented as response to oriented filters
  - 27 filters at 3 scales and 9 orientations
  - Learn coefficients from labeled examples
- Parts translate with respect to central part, tip of nose ($d_i$)
Flexible Template Face Detection

- Runs at several frames per second
  - Compute oriented filters at 27 orientations and scales for part cost $m_i$
  - Distance transform $m_i$ for each part other than central one (nose tip)
  - Find maximum of sum for detected location
More General Flexible Templates

- Efficient computation using distance transforms for any tree-structured model
  - Not limited to central reference part
- Two differences from reference part case
  - Relate positions of parts to one another using tree-structured recursion
    - Solve with Viterbi or forward-backward algorithm
  - Parameterization of distance transform more complex – transformation $T_{ij}$ for each connected pair of parts
General Form of Problem

- Best location can be viewed in terms of probability or cost (negative log prob.)
  - $\max_{L} p(L|I,\Theta) = \arg\max_{L} p(I|L,A)p(L|E,C)$
  - $\min_{L} \sum_{V} m_{j}(l_{j}) + \sum_{E} d_{ij}(l_{i},l_{j})$
    - $m_{j}(l_{j})$ – how well part $v_{j}$ matches image at $l_{j}$
    - $d_{ij}(l_{i},l_{j})$ – how well locations $l_{i},l_{j}$ agree with model (spring connecting parts $v_{i}$ and $v_{j}$)

- Difficulty of maximization/minimization depends on form of graph
  - Exponential time in general, efficient for tree
Minimizing Over Tree Structures

- Use dynamic programming to minimize 
  \[ \sum_v m_j(l_j) + \sum_E d_{ij}(l_i,l_j) \]
- Can express as function for pairs \( B_j(l_i) \)
  - Cost of best location of \( v_j \) given location \( l_i \) of \( v_i \)
- Recursive formulas in terms of children \( C_j \) of \( v_j \)
  - \( B_j(l_i) = \min_{l_j} ( m_j(l_j) + d_{ij}(l_i,l_j) + \sum_{C_j} B_c(l_j) ) \)
  - For leaf node no children, so last term empty
  - For root node no parent, so second term omitted
Efficient Algorithm for Trees

- MAP estimation algorithm
  - Tree structure allows use of Viterbi style dynamic programming
    - $O(ns^2)$ rather than $O(s^n)$ for $s$ locations, $n$ parts
    - Still slow to be useful in practice ($s$ in millions)
  - Couple with distance transform method for finding best pair-wise locations in linear time
    - Resulting $O(ns)$ method

- Similar techniques allow sampling from posterior distribution in $O(ns)$ time
  - Using forward-backward algorithm
O(ns) Algorithm for MAP Estimate

- Express $B_j(l_i)$ in recursive minimization formulas as a DT $D_f(T_{ij}(l_i))$
  - Cost function
    - $f(y) = m_j(T_{ji}^{-1}(y)) + \sum_{c_j} B_c(T_{ji}^{-1}(y))$
  - $T_{ij}, T_{ji}$ map locations to space where difference between $l_i$ and $l_j$ is a squared distance
    - Distance zero at ideal relative locations

- Yields $n$ recursive equations
  - Each can be computed in $O(sD)$ time
    - D is number of dimensions to parameter space but is fixed (in our case D is 2 to 4)
Example: Recognizing People
Variety of Poses
Variety of Poses
Samples From Posterior
Model of Specific Person
Bayesian Formulation of Learning

- Given example images $I^1, \ldots, I^m$ with configurations $L^1, \ldots, L^m$
  - Supervised or labeled learning problem
- Obtain estimates for model $\Theta = (A, E, C)$
- Maximum likelihood (ML) estimate is
  - $\arg\max_\Theta p(I^1, \ldots, I^m, L^1, \ldots, L^m | \Theta)$
  - $\arg\max_\Theta \prod_k p(I^k, L^k | \Theta)$
    - Independent examples
  - $\arg\max_\Theta \prod_k p(I^k | L^k, A) \prod_k p(L^k | E, C)$
    - Independent appearance and dependencies
Efficiently Learning Tree Models

- Estimating appearance $p(I^k|L^k,A)$
  - ML estimation for particular type of part
    - E.g., for constant color patch use Gaussian model, computing mean color and covariance

- Estimating dependencies $p(L^k|E,C)$
  - Estimate C for pairwise locations, $p(l^k_i,l^k_j|c_{ij})$
    - E.g., for translation compute mean offset between parts and variation in offset
  - Best tree using minimum spanning tree (MST) algorithm
    - Pairs with “smallest relative spatial variation”
Example: Generic Person Model

- Each part represented as rectangle
  - Fixed width, varying length
  - Learn average and variation
    • Connections approximate revolute joints
  - Joint location, relative position, orientation, foreshortening
  - Estimate average and variation

- Learned model (used above)
  - All parameters learned
    • Including “joint locations”
  - Shown at ideal configuration