Cute Tricks with Dynamic Programming: Tracking and Parsing

D.A. Forsyth

Recall: Dynamic Programming

- We had to choose a set of discrete variables to minimize a cost function

$$u_1(X_1) + b_{12}(X_1, X_2) + u_2(X_2) + b_{23}(X_2, X_3) + u_3(X_3) + ...u_n(X_n)$$

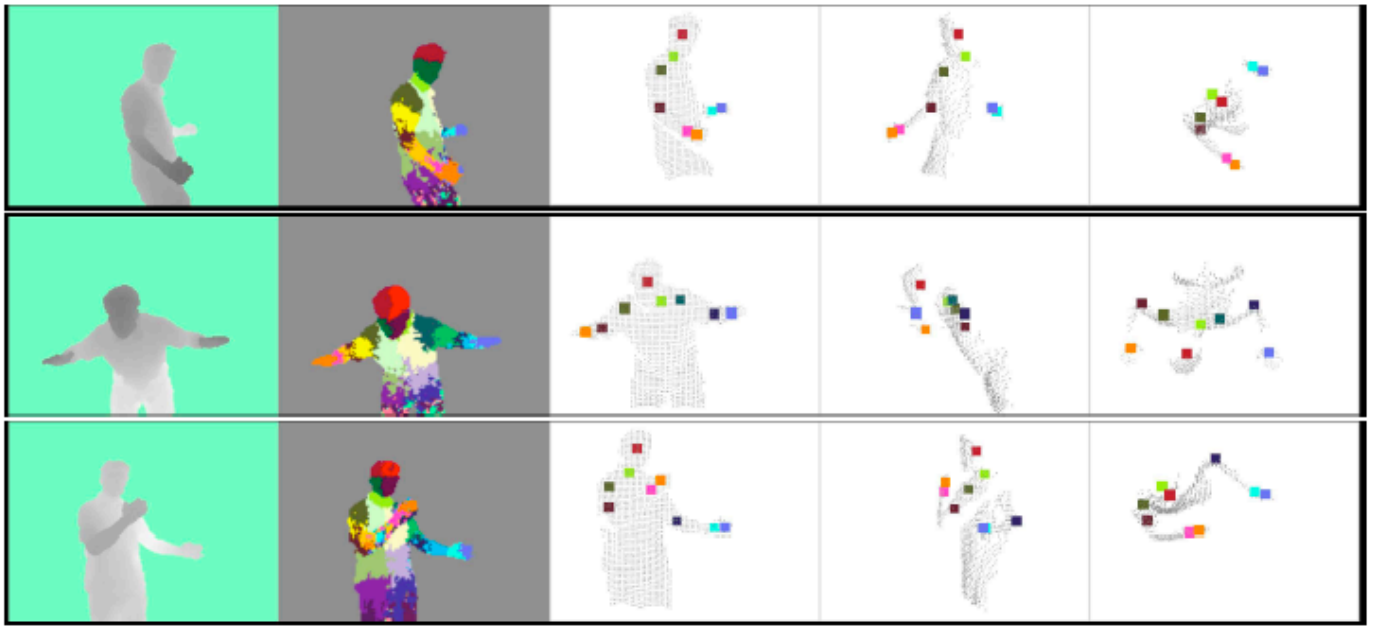
- We did this by building a cost-to-go function
  - recursively
  - at X_(n-1)=cost of best path leaving X_n-1
  - at X_k= best of (cost of leaving X_k+cost to go(X_k+1))

Human parsing

- We are given an image
- We have a model of a person
  - head+torso
  - head+torso+upper+lower arms
  - head+torso+upper+lower arms+legs
  - more, perhaps
  - could be 2D or 3D
- We know a bunch of stuff about this model
  - eg color of each segment
  - eg how segments join up
- We must report configuration of the model

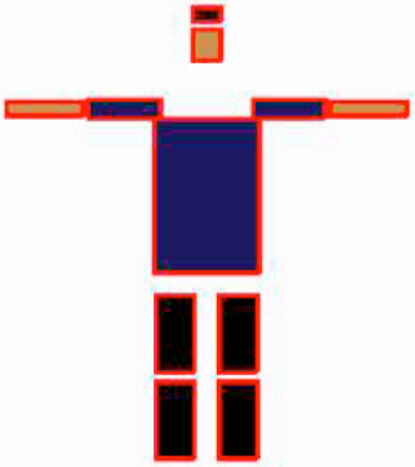
Applications

- Kinect



Simplest case

- Model
  - 2D
  - segments are image rectangles of known size
  - we know the color of the segments
  - we have two cost functions
    - segment to image match
      - eg sum of squared color differences
    - segment to segment compatibility
      - eg ends are close; possibly angles
      - in a tree (more details to follow)
- Matching = dynamic programming



Building a trellis

- Simple cases
  - head+torso
    - one column each
    - entries in the column are image rectangles
    - picture just like the texture picture
  - head+torso+two upper arms
    - one column per segment
    - entries are image rectangles
    - picture only slightly more interesting, because we have a tree

In notation

- We have variables X1..Xn
  - one per body segment
  - each can be mapped to an image rectangle
  - for the moment, assume they form a chain
    - eg head+torso; head+torso+upper+lower leg
    - trees will be easy, as above
- We have a cost function
  - with a special form (for the case of a chain)

$$f_{\text{chain}}(X_1, \ldots, X_n) = \sum_{i=1}^{i=n} u_i(X_i) + \sum_{i=1}^{i=n-1} b_i(X_i, X_i + 1)$$

- We seek

$$\underset{X_1, \ldots, X_n}{\text{argmax}} \; f_{\text{chain}}(X_1, \ldots, X_n)$$

In notation -II: the cost to go function

- We have for the chain function:

$$f_{\text{chain}}(X_1, \ldots, X_n) = \sum_{i=1}^{i=n} u_i(X_i) + \sum_{i=1}^{i=n-1} b_i(X_i, X_i + 1)$$

- Define the cost-to-go function by

$$f^{(n-1)}_{\text{cost-to-go}}(X_{n-1}) = \max_{X_n} b_{n-1}(X_{n-1}, X_n) + u_n(X_n),$$

  - this is the base of a recursion!
- Notice that

$$\underset{X_1, \ldots, X_n}{\text{argmax}} \; f_{\text{chain}}(X_1, \ldots, X_n)$$

  - is the same as (except for Xn, which is missing)

$$\underset{X_1, \ldots, X_{n-1}}{\text{argmax}} \; \left(f_{\text{chain}}(X_1, \ldots, X_{n-1}) + f^{(n-1)}_{\text{cost-to-go}}(X_{n-1}) \right),$$

In notation -III: the cost-to-go function-II

which means that we can eliminate the nth variable from the optimization by replacing the term $b_{n-1}(X_{n-1}, X_n)+u_n(X_n)$ with a function of X_{n-1}. This function is obtained by maximizing this term with respect to X_n. Equivalently, assume we must choose a value for X_{n-1}. The cost-to-go function tells us the value of $b_{n-1}(X_{n-1}, X_n) + u_n(X_n)$ obtained by making the best choice of X_n conditioned on our choice of X_{n-1}. Because any other choice would not lead to a maximum, if we know the cost-to-go function at X_{n-1}, we can now compute the best choice of X_{n-1} conditioned on our choice of X_{n-2}. This yields that

$$\max_{X_{n-1},X_n} [b_{n-2}(X_{n-2}, X_{n-1}) + u_{n-1}(X_n - 1) + b_{n-1}(X_{n-1}, X_n) + u_n(X_n)]$$

is equal to

$$\max_{X_{n-1}} \left[b_{n-2}(X_{n-2}, X_{n-1}) + u_{n-1}(X_n - 1) + \left(\max_{X_n} b_{n-1}(X_{n-1}, X_n) + u_n(X_n) \right) \right].$$

But all this can go on recursively, yielding

$$f^{(k)}_{\text{cost-to-go}}(X_k) = \max_{X_{k+1}} b_k(X_k, X_{k+1}) + u_k(X_k) + f^{(k+1)}_{\text{cost-to-go}}(X_{k+1}).$$

In notation - IV

$$\max_{X_1,\ldots,X_n} f_{\text{chain}}(X_1,\ldots,X_n)$$

is equal to

$$\max_{X_1,\ldots,X_{n-1}} \left(f_{\text{chain}}(X_1,\ldots,X_{n-1}) + f^{(n-1)}_{\text{cost-to-go}}(X_{n-1}) \right)$$

which is equal to

$$\max_{X_1,\ldots,X_{n-2}} \left(f_{\text{chain}}(X_1,\ldots,X_{n-2}) + f^{(n-2)}_{\text{cost-to-go}}(X_{n-2}) \right),$$

So

$$\max_{X_1,\ldots,X_n} f_{\text{chain}}(X_1,\ldots,X_n) = \max_{X_1} \left(f_{\text{chain}}(X_1) + f^{1}_{\text{cost-to-go}}(X_1) \right)$$

Dynamic programming for a chain

which yields an extremely powerful maximization strategy. We start at X_n, and construct $f^{(n-1)}_{\text{cost-to-go}}(X_{n-1})$. We can represent this function as a table, giving the value of the cost-to-go function for each possible value of X_{n-1}. We build a second table giving the optimum X_n for each possible value of X_{n-1}. From this, we can build $f^{(n-2)}_{\text{cost-to-go}}(X_{n-2})$, again as a table, and also the best X_{n-1} as a function of X_{n-2}, again as a table, and so on. Now we arrive at X_1. We obtain the solution for X_1 by choosing the X_1 that yields the best value of $\left(f_{\text{chain}}(X_1) + f^2_{\text{cost-to-go}}(X_2)\right)$. But from this solution, we can obtain the solution for X_2 by looking in the table that gives the best X_2 as a function of X_1; and so on. It should be clear that this process yields a solution in polynomial time; in the exercises, you will show that, if each X_i can take one of k values, then the time is $O(nK^2)$.

Heavily simplified tracking model

- We have a moving object, and want to trace its path
- Two sources of information
  - Measurement
    - actual, but unknown, position in k'th image is x_k (2D, continuous)
    - measured position in k'th image is m_k (2D, continuous)
      - very close to actual position
        - m_k=x_k+(tiny position error)
    - far more complex models are possible
  - Dynamics
    - it isn't moving much
      - i.e. x_k+1=x_k+(tiny position change)
    - far more complex models are possible

In equations

Motion constraint

$$x_{k+1} = x_k + \zeta_k$$

tiny position change
scale is σ_p

Measurement constraint

$$m_k = x_k + \eta_k$$

tiny position change
scale is σ_m

Finding the path

- To find the path, we must choose x_1, ..., x_n to minimize

$$\frac{(m_1 - x_1)^T (m_1 - x_1)}{\sigma_m^2} +$$
$$\frac{(x_2 - x_1)^T (x_2 - x_1)}{\sigma_p^2} +$$
$$\cdots$$
$$\frac{(x_n - x_{n-1})^T (x_n - x_{n-1})}{\sigma_p^2} +$$
$$\frac{(m_n - x_n)^T (m_n - x_n)}{\sigma_m^2}$$

Building a cost-to-go function

- Terms involving x_n:

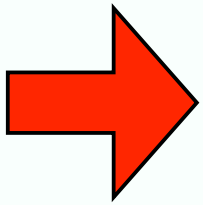
$$\frac{(x_n - x_{n-1})^T (x_n - x_{n-1})}{\sigma_p^2} + \frac{(m_n - x_n)^T (m_n - x_n)}{\sigma_m^2}$$

- Maximize with respect to x_n
  - set gradient wrt x_n to zero
  - rearrange
  - get

$$x_n = \left(\frac{x_{n-1}}{\sigma_p^2} + \frac{m_n}{\sigma_m^2} \right) \left(\frac{\sigma_m^2 \sigma_p^2}{\sigma_m^2 + \sigma_p^2} \right)$$

We can substitute back

$$\frac{(m_1-x_1)^T(m_1-x_1)}{\sigma_m^2}+\frac{(x_2-x_1)^T(x_2-x_1)}{\sigma_p^2}+\cdots+\frac{(x_n-x_{n-1})^T(x_n-x_{n-1})}{\sigma_p^2}+\frac{(m_n-x_n)^T(m_n-x_n)}{\sigma_m^2}$$



$$\frac{(m_1-x_1)^T(m_1-x_1)}{\sigma_m^2}+\frac{(x_2-x_1)^T(x_2-x_1)}{\sigma_p^2}+\cdots+\frac{(x_{n-2}-x_{n-1})^T(x_{n-2}-x_{n-1})}{\sigma_p^2}+\frac{(x_{n-1}-m_{n-1})^T(x_{n-1}-m_{n-1})}{\sigma_m^2}+c(x_{n-1})$$

And do this again and again...

- To get some c(x_1)
  - minimize this
  - substitute x_1 in expression for x_2
    - x_2 in expression for x_3
    - ...
    - x_n-1 in expression for x_n
- This is a dynamic program, too

- Can be extended to:
  - more complex state models
  - more complex measurement models
  - more complex dynamic models
- Result: The Kalman Filter

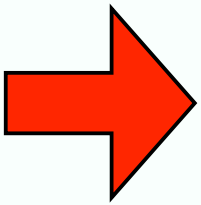
Reminder

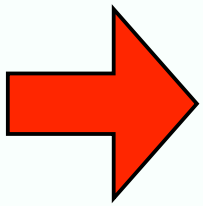
9.4.1 Terminology and Facts for Graphs

We review terminology here very briefly, as it's quite easy to forget.

- A *graph* is a set of vertices V and edges E that connect various pairs of vertices. A graph can be written $G = \{V, E\}$. Each edge can be represented by a pair of vertices—that is, $E \subset V \times V$. Graphs are often drawn as a set of points with curves connecting the points.
- The *degree* of a vertex is the number of edges incident on that vertex.

- A *directed graph* is one in which edges (a, b) and (b, a) are distinct; such a graph is drawn with arrowheads indicating which direction is intended.
- An *undirected graph* is one in which no distinction is drawn between edges (a, b) and (b, a).
- A *weighted graph* is one in which a weight is associated with each edge.
- Two edges are *consecutive* if they have a vertex in common.
- A *path* is a sequence of consecutive edges.
- A *circuit* is a path which ends at the vertex at which it begins.
- A *self-loop* is an edge that has the same vertex at each end; self-loops don't occur in our applications.
- Two vertices are said to be *connected* when there is a sequence of edges starting at the one and ending at the other; if the graph is directed, then the arrows in this sequence must point the right way.
- A *connected graph* is one where every pair of vertices is connected.
- A *tree* is a connected graph with no circuits.





- Given a connected graph $G = \{V, E\}$, a *spanning tree* is a tree with vertices V and edges a subset of E. By our definition, trees are connected, so a spanning tree is connected.
- Every graph consists of a disjoint set of *connected components*—that is, $G = \{V_1 \cup V_2 \ldots V_n, E_1 \cup E_2 \ldots E_n\}$, where $\{V_i, E_i\}$ are all connected graphs and there is no edge in E that connects an element of V_i with one of V_j for $i \neq j$.
- A *forest* is a graph whose connected components are trees.

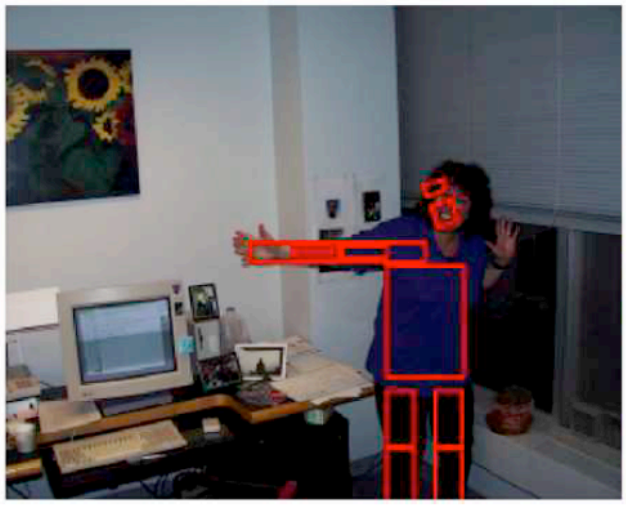
DP works for forests

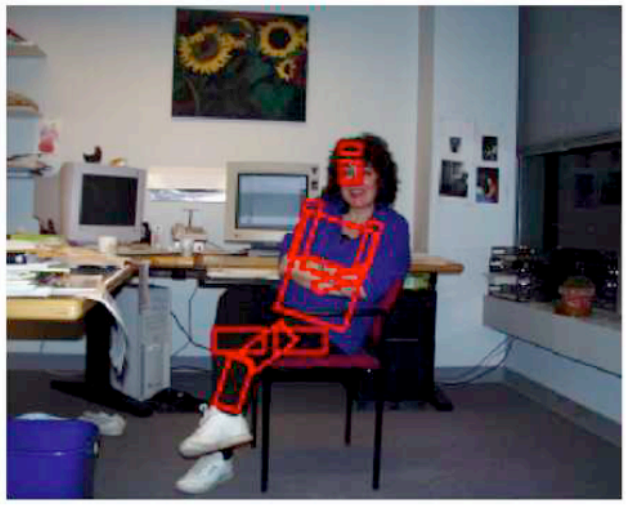
This strategy will work for a model with the structure of a forest. The proof is an easy induction. If the forest has no edges (i.e., consists entirely of nodes), then it is obvious that a simple strategy applies (choose the best value for each X_i independently). This is clearly polynomial. Now assume that the algorithm yields a result in polynomial time for a forest with e edges, and show that it works for a forest with $e + 1$ edges. There are two cases. The new edge could link two existing trees, in which case we could re-order the trees so the nodes that are linked are roots, construct a cost-to-go function for each root, and then choose the best pair of states for these roots from the cost-to-go functions. Otherwise, one tree had a new edge added, joining the tree to an isolated node. In this case, we reorder the tree so that this new node is the root and build a cost-to-go function from the leaves to the root. The fact that the algorithm works is a combinatorial insight, but many kinds of model have a tree structure. Models of this form are particularly important in cases of tracking and of parsing.

Some issues

- How do you deal with two legs?
  - or two arms?
  - or, rather, how do you ensure the match has two?

"Efficient Matching of Pictorial Structures," by P. Felzenszwalb and D.P. Huttenlocher, Proc. IEEE CVPR 2000, c 2000, IEEE.

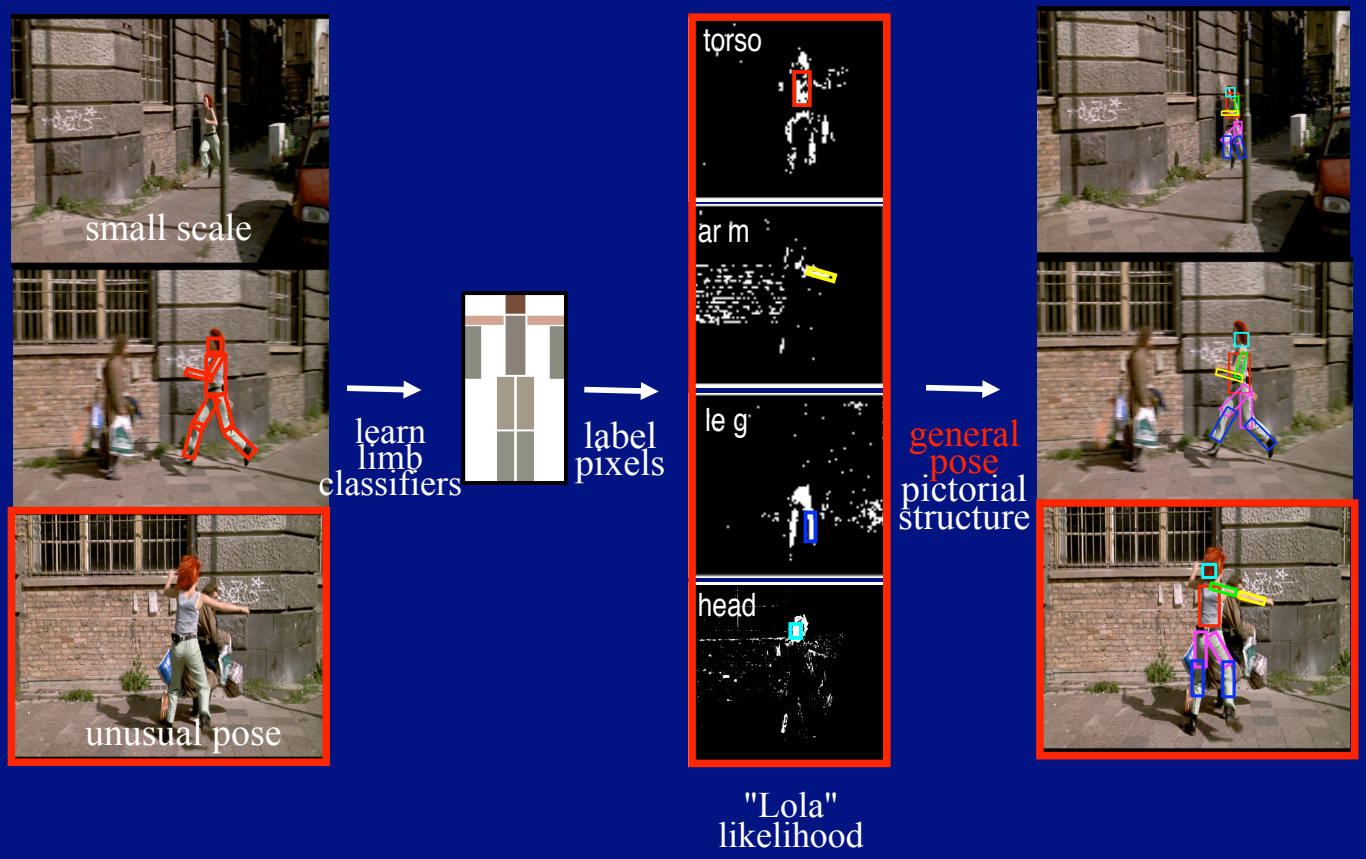




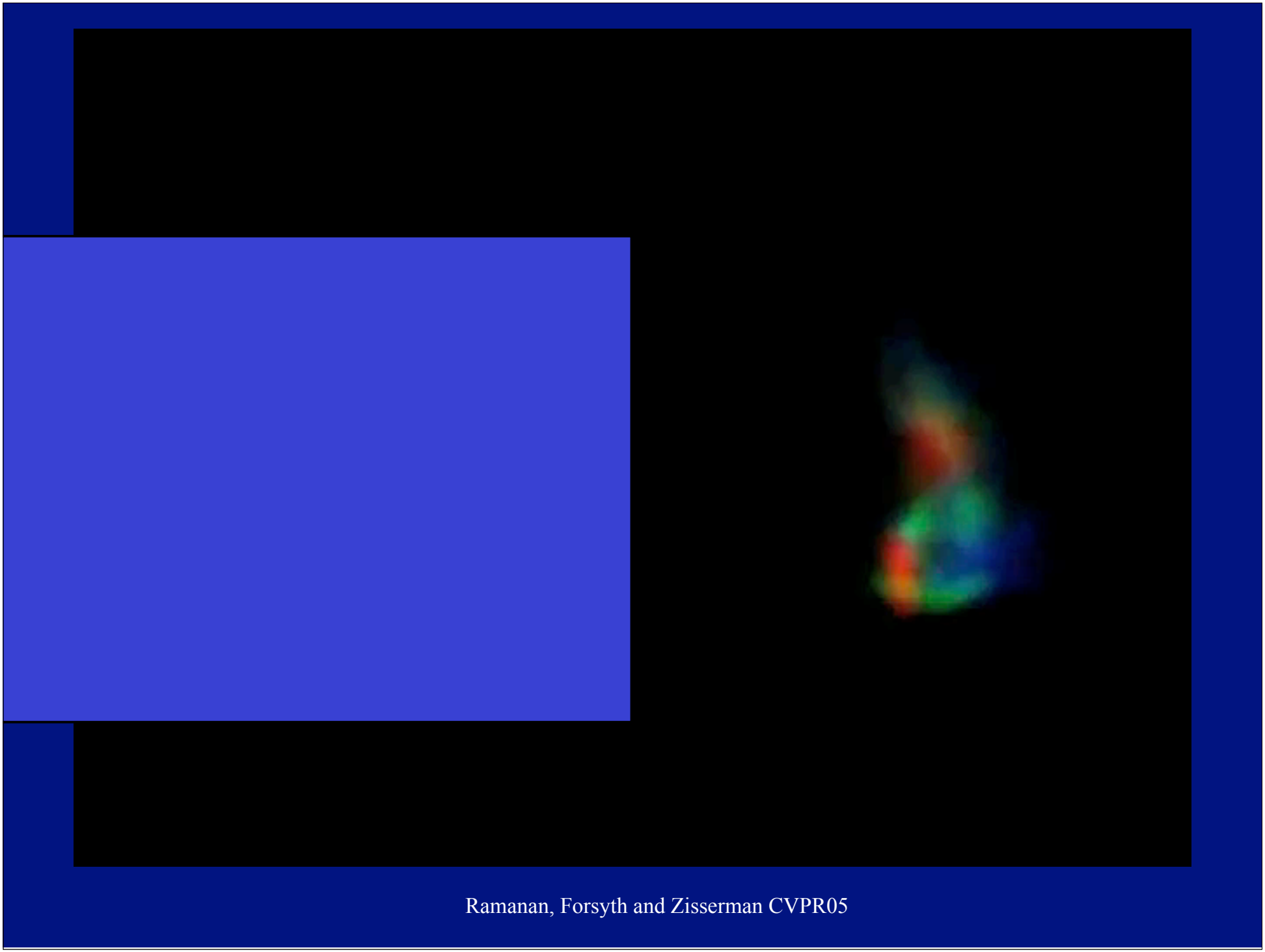
What if you don’t know the color?

- Easy answer for video:
  - look for lateral walking view with a classifier
  - read off appearance of arms, legs, etc
  - feed into model; now detect in each frame

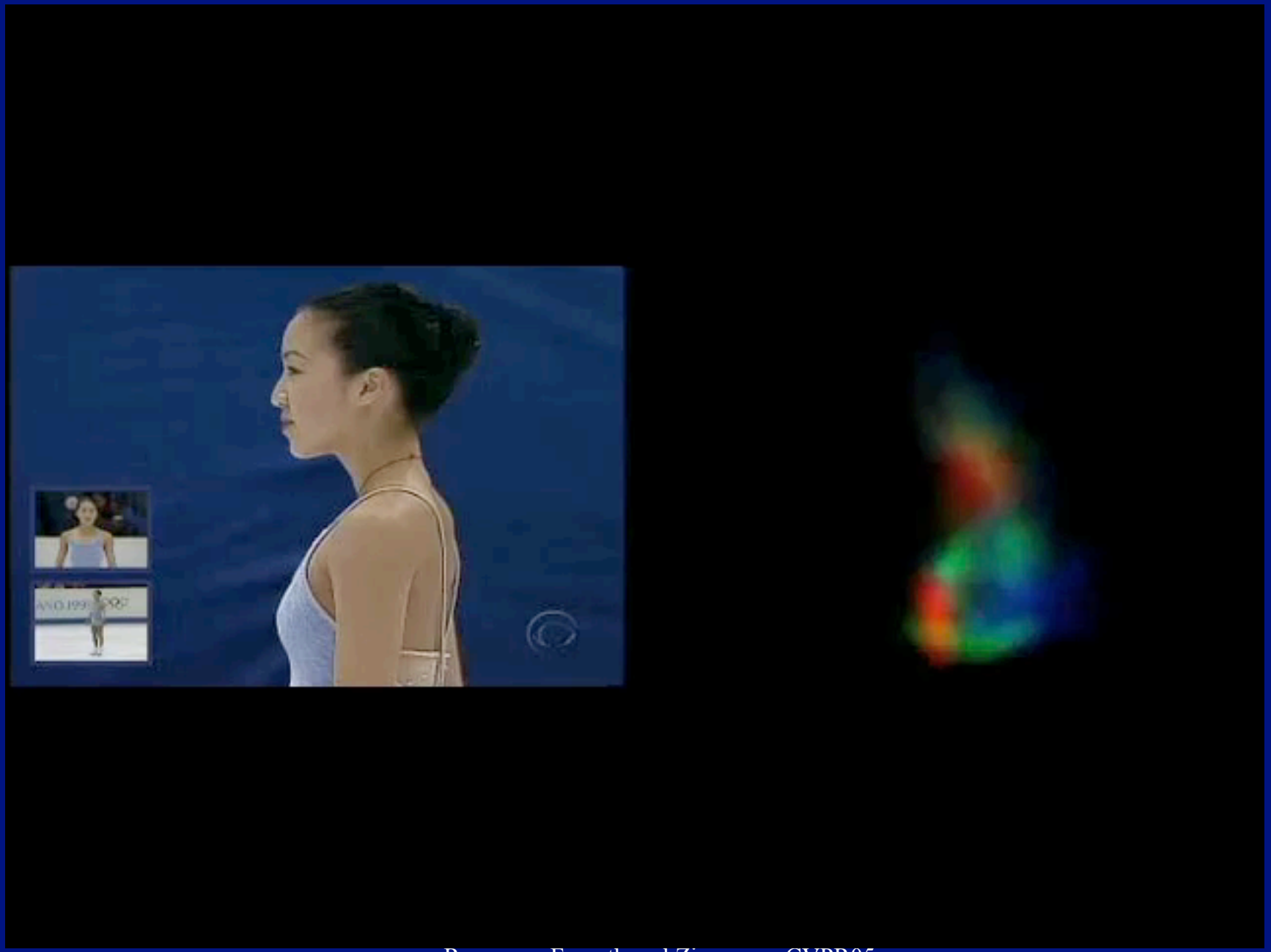
Build and detect models



Ramanan, Forsyth and Zisserman CVPR05



Ramanan, Forsyth and Zisserman CVPR05

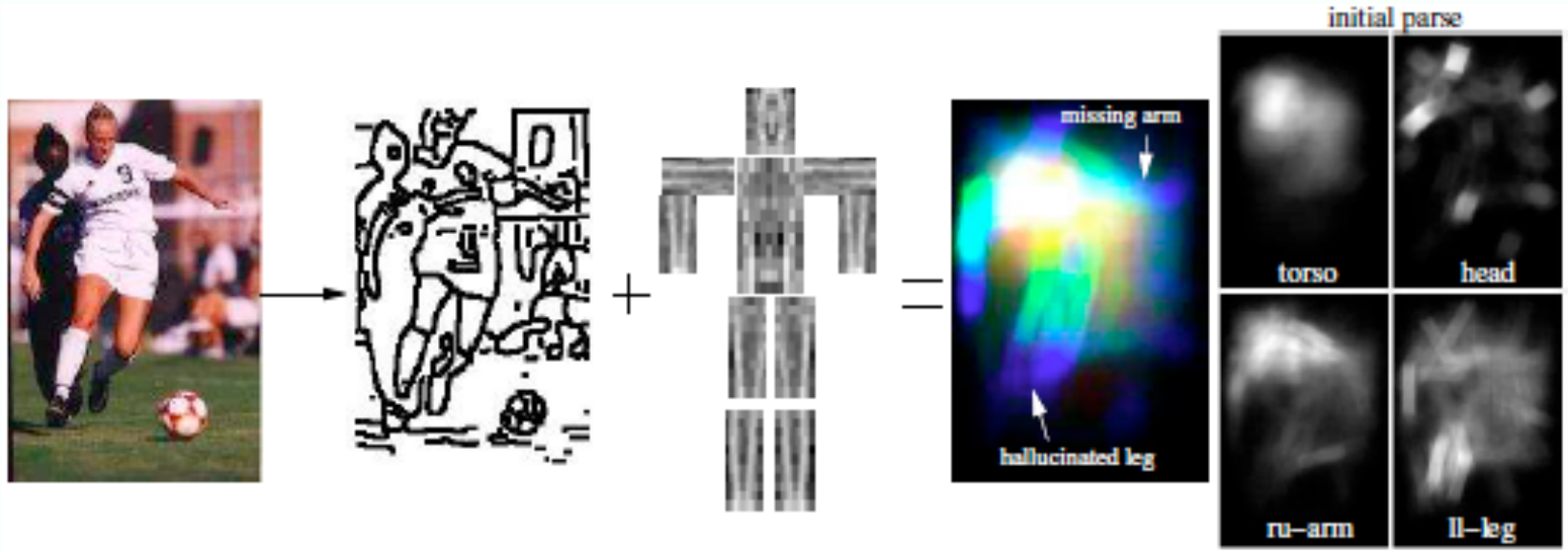


Ramanan, Forsyth and Zisserman CVPR05

What if you don’t know the color? - II

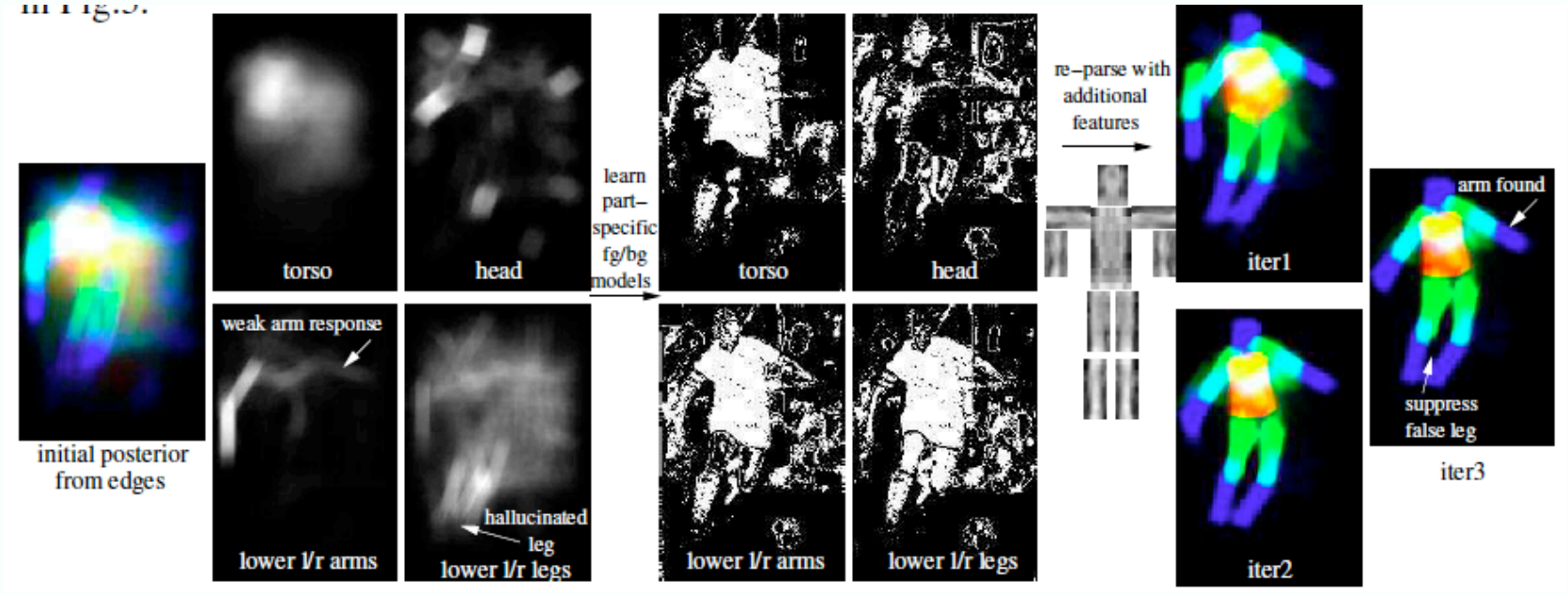
- Hard answer for static images:
  - guess color; parse; reestimate color; reparse; and so on

Guessing the color



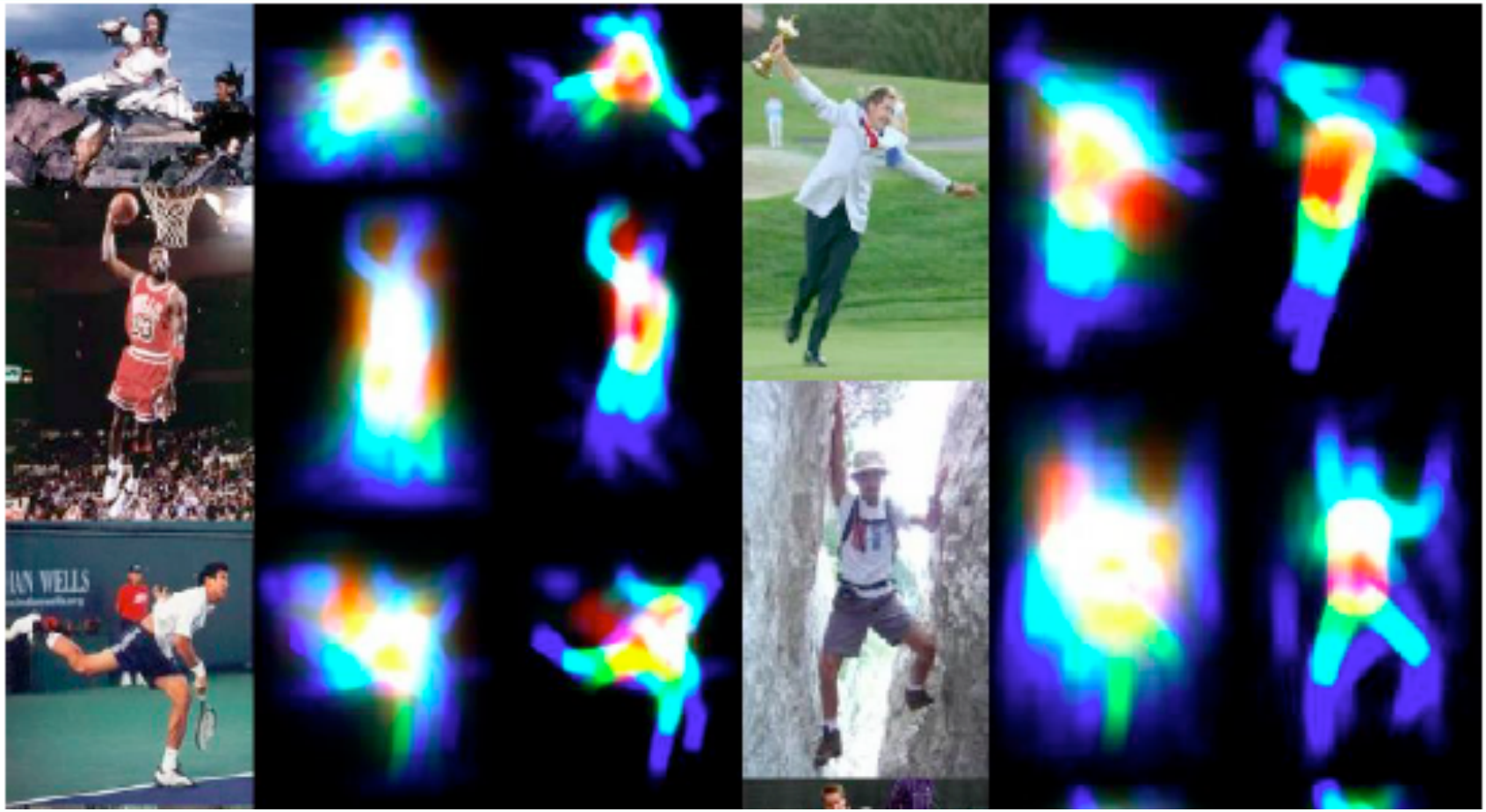
From Ramanan, 03

Iterating



From Ramanan, 03

Iterating



From Ramanan, 03

This is a search

- And we can prune it
  - It’s easy to detect torso’s accurately
  - the arms can get only so far from the torso

Pruned search

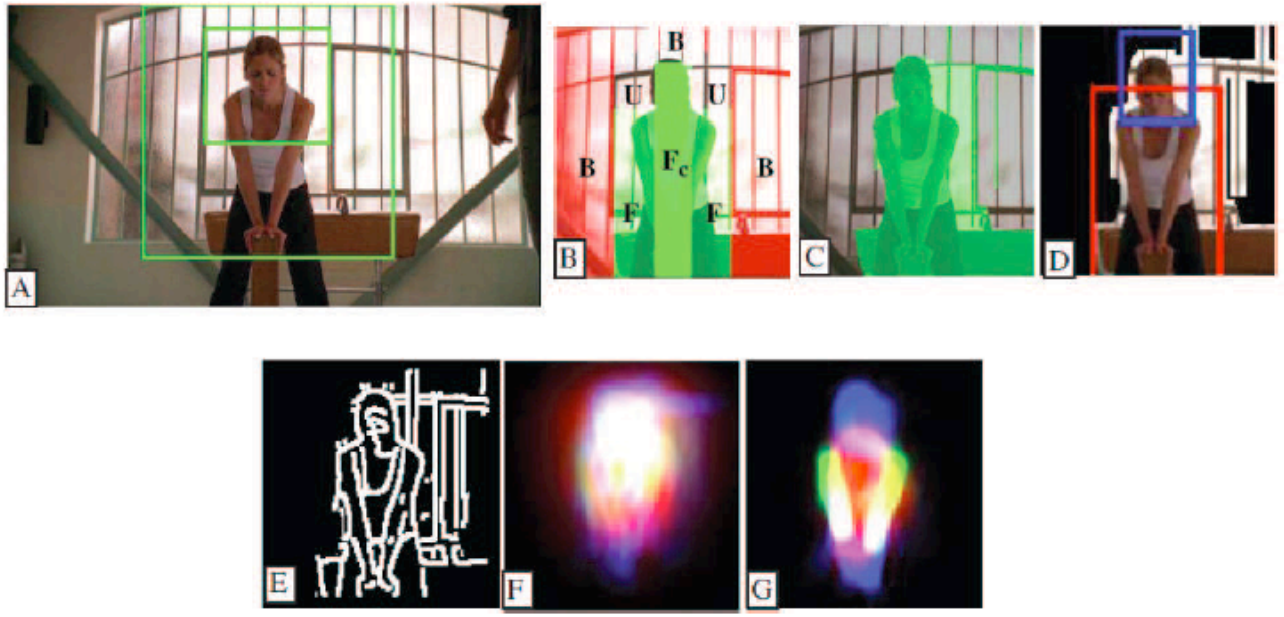


FIGURE 20.5: The human parser of Ramanan (2006) is a search of all spatial layouts in the image to find one that is consistent with the constraints we know on appearance. Ferrari *et al.* (2008) show that reducing the search space improves the results. First, one finds upper bodies, and builds a box around those detections using constraints on the body size (**A**). Outside this box is background, and some pixels inside this box are, too. In **B**, body constraints mean that pixels labeled F_c and F are very likely foreground, U are unknown, and B are very likely background. One then builds color models for foreground and background using this information, then uses an interactive segmenter to segment, requiring that F_c pixels be foreground, to get **C**. The result is a much reduced search domain for the human parser, which starts using an edge map **D**, to get an initial parse **E**, and, after iterating, produces **F**. *This figure was originally published as Figure 2 of "Progressive search space reduction for human pose estimation," by V. Ferrari, M. Marín-Jiménez, and A. Zisserman, Proc. IEEE CVPR 2008,* © *IEEE 2003.*

But dynamic programming fails

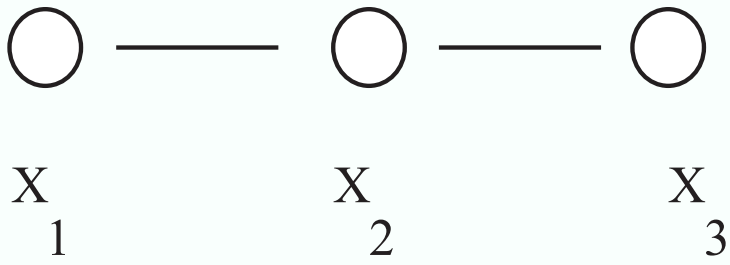
- We've already seen that getting two arms/legs is an issue
  - which we ducked, somewhat
- Our model is wrong
  - arms tend to have the same color
  - legs tend to have the same color
  - this means there should be terms in the cost function that link left/right
  - this breaks dp

Breaking dp - I

- Cost function for a chain of three variables:

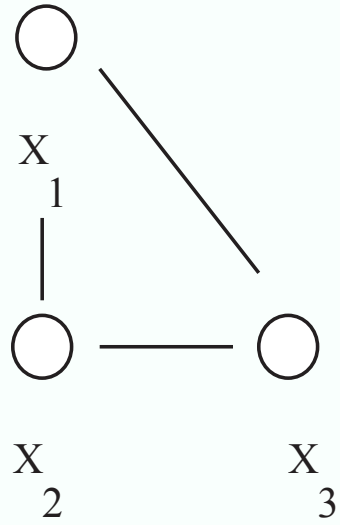
$$u_1(X_1) + b_{12}(X_1, X_2) + u_2(X_2) + b_{23}(X_2, X_3) + u_3(X_3)$$

- Picture:



Breaking dp

- New picture



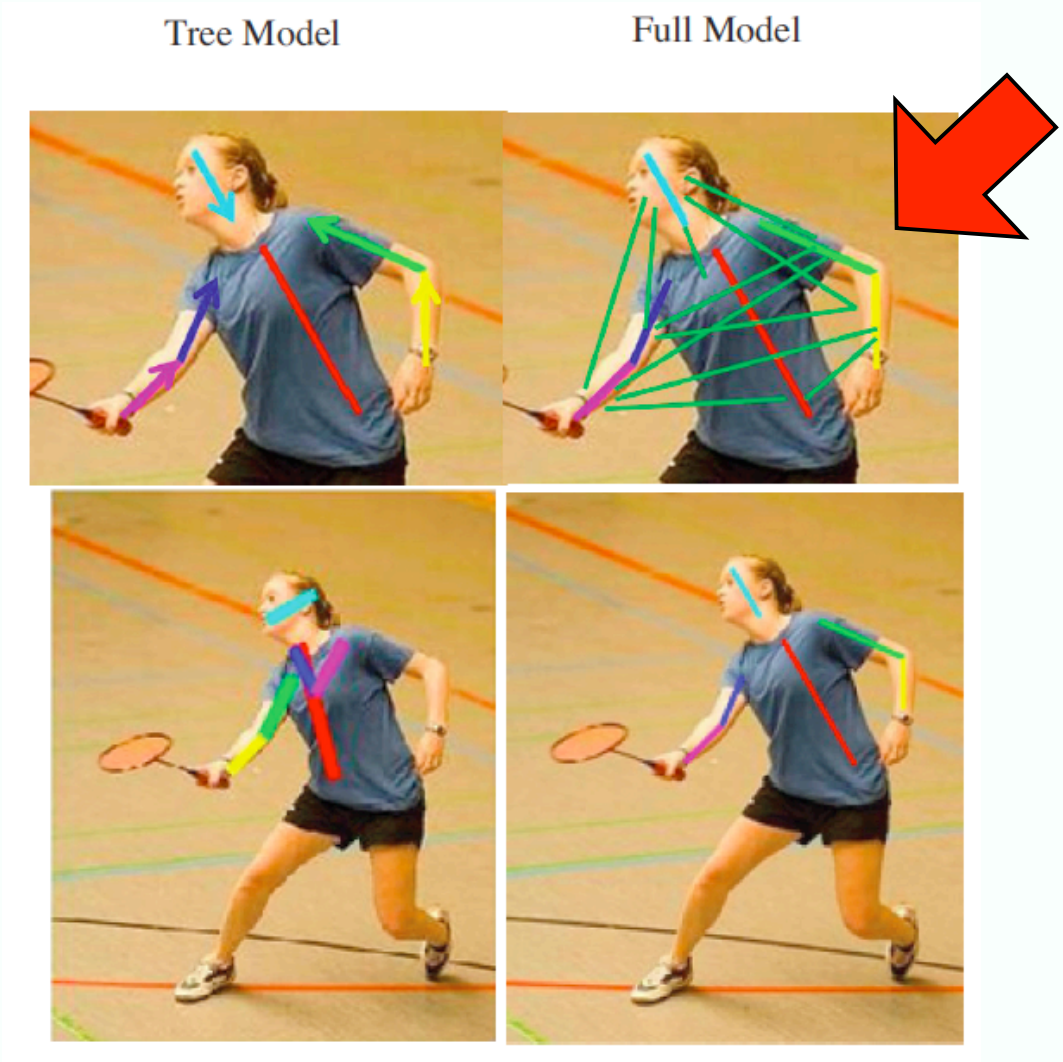
$$u_1(X_1) + b_{12}(X_1, X_2) + u_2(X_2) + b_{23}(X_2, X_3) + u_3(X_3) + b_{13}(X_1, X_3)$$

Breaking dp - III

- Try to build a cost-to-go function:
  - eliminate X3
  - but now our cost-to-go function depends on X1 and X2
  - so no benefit!
- We need to check all triples!

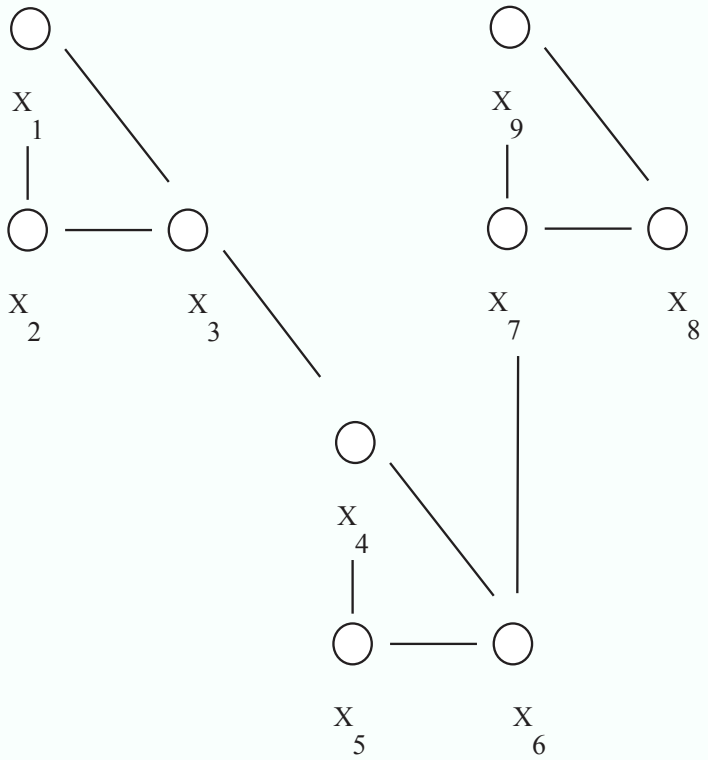
$$u_1(X_1) + b_{12}(X_1, X_2) + u_2(X_2) + b_{23}(X_2, X_3) + u_3(X_3) + b_{13}(X_1, X_3)$$

Breaking dp - IV



Tran and Forsyth 10

All sorts of pictures



Could you do dynamic programming here?

Not in terms of X1...X9

but in terms of G1, G2, G3

G1=(X1, X2, X3)

G2=(X4, X5, X6)

G3=(X7, X8, X9)

Strategy for non-dp cases

- Approximation
  - true answer is intractable
- Strategy:
  - Fix some u's, to make a chain
  - do dp
  - now fix different u's, iterate
- One of many possible strategies
  - question is very rich

Tran+Forsyth, 10

Important points

- Human parsing is useful
- Can do with dp
- Can lead to models where you can't maximize with dp
  - or can't maximize efficiently
- What to do is an "algorithms" question
- Approximate algorithms exist
  - Hard questions
    - "best" approximate algorithm
    - quality of approximation