DYNAMIC PROGRAMMING VERSUS GRAPH CUT ALGORITHMS FOR PHENOTYPING BY IMAGE ANALYSIS

Chris Glasbey

Biomathematics and Statistics Scotland
PLANT PHENOTYPING: HOW BIG ARE THESE LEAVES?

We cannot simply measure area of leaf as it appears in image: this depends on leaf orientation and distance to camera.
ANIMAL PHENOTYPING: HOW FAT IS THIS SHEEP?

Ultrasound image of sheep’s back: **Identify layer of fat**
Compare data ($y$) with each boundary template ($\mu$)
Compare data \((y)\) with each boundary template \((\mu)\), to produce:

\[
f_i(y, \beta) = \sum_{k=-K}^{K} \left( y_{\beta+k,i} - \mu_{k,i} \right)^2 \quad \text{for } i = 1, \ldots, I, \quad \beta \in \mathcal{B} = \{1, 2, \ldots, B\}
\]
Goodness of fit: \( f(y, \beta) \)

Solve optimisation problem to find upper (lower) boundary:

\[ \hat{\beta} = \arg \min_{\beta \in \mathcal{B}^I} \left\{ \sum_{i=1}^{I} f_i(y, \beta_i) + \lambda \sum_{|i-j|=1} (\beta_i - \beta_j)^2 \right\} \]

where boundary at column \( i \) is row \( \beta_i \)
Toy example:

One of approx. 700 possible connected paths

Both dynamic programming (DP) and graph cut (GC) algorithms can find path from left to right with globally minimum cost
PLAN OF TALK

1. Dynamic programming (DP)
2. Application to animal phenotyping
3. Graph cut algorithm (GC)
4. Application to plant phenotyping
5. Summary
Minimum cost paths to column 2
Minimum cost paths to column 3
Minimum cost paths to all cols
Minimum cost in final column
Trace path back to first column
Minimum cost path
2. APPLICATION TO ANIMAL PHENOTYPING (WITH SRUC)

Result: Automatic and hand-drawn boundaries:

(Glasbey and Young, 2002, Applied Statistics)
What if image is 3D?

1D line

\( \beta: 1D \) array of 2D vectors

DP: YES

2D surface

\( \beta: 2D \) array of scalars

DP: NO
Segmentation of 3D images (i.e. a stack of 2D images)

For specific applications we have automatic algorithms: use Iterated DP as a local optimiser to find boundary in each image in turn.

For generic applications we have a semi-automatic algorithm:

- a human operator segments image 1 manually
- computer segments image 2, mimicking image 1 segmentation
- operator corrects any mistakes in image 2 segmentation
- computer segments image 3, mimicking image 2 segmentation
- operator corrects any mistakes in image 3 segmentation
- etc

Here is an illustration of how “mimicking” works:
pig CT image + boundary

next image in stack
+ perpendicular transects
⇒ same transects on next image
compare a transect
compare a transect ⇒ root-mean-square-difference
compare a transect at different shifts
compare a transect at different shifts
compare a transect at different shifts
compare a transect at different shifts
compare a transect at different shifts
compare a transect at different shifts
compare a transect at different shifts
compare a transect at different shifts

- Position
- Pixel value
- Score
- Shift
compare a transect at different shifts
compare a transect at different shifts
compare a transect at different shifts

<table>
<thead>
<tr>
<th>position</th>
<th>pixel value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10</td>
<td>100</td>
</tr>
<tr>
<td>-5</td>
<td>150</td>
</tr>
<tr>
<td>0</td>
<td>200</td>
</tr>
<tr>
<td>5</td>
<td>150</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>shift</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10</td>
<td>0</td>
</tr>
<tr>
<td>-5</td>
<td>50</td>
</tr>
<tr>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>150</td>
</tr>
<tr>
<td>10</td>
<td>200</td>
</tr>
</tbody>
</table>
compare a transect

at different shifts

pixel value

0 100 200

position

-10 -5 0 5 10

score

0 50 100

shift

-10 -5 0 5 10
compare a transect at different shifts
compare a transect position

pixel value

score

shift
compare a transect at different shifts
compare a transect at different shifts
compare a transect at different shifts
compare a transect at different shifts
compare a transect at different shifts
compare a transect at different shifts
comparison of all transects
minimum score boundary
+ roughness penalty
Compare transects on new image \((y)\) with previous image \((y')\):

\[
f_i(y, \beta) = \sum_{k=-K}^{K} \left( y_{t_{i,\beta+k}} - y'_{t_{ik}} \right)^2 \quad \text{for } i = 1, \ldots, I, \quad \beta \in \mathcal{B}
\]

Then find new boundary, by using DP to solve optimisation problem:

\[
\hat{\beta} = \arg \min_{\beta \in \mathcal{B}^I} \left\{ \sum_{i=1}^{I} f_i(y, \beta_i) + \lambda \sum_{|i-j| = 1 \pmod{I}} (\beta_i - \beta_j)^2 \right\}
\]
Parameters in algorithm:

- number of transects ($I$)
- lengths of transects ($K$)
- range of boundary shift ($B$)
- roughness penalty ($\lambda$)

can be tuned to a specific application using a few manual segmentations
Chicken breast: manual boundary

image 1
Chicken breast: **automatic** + **manual correction** (different boundary)
Chicken breast: automatic + manual correction (second boundary)
Chicken breast: automatic + manual correction (edited boundary)
Chicken breast: automatic (no manual correction)
Chicken breast: automatic (no manual correction)
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: **automatic** + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction
Chicken breast: automatic + any manual correction

last image
The semi-automatic algorithm is generic to handle many 3D segmentations:

- boundaries can be exterior or holes, and of any smooth shape
- pixel value templates are simply based on previous image
- Dynamic Programming is fast and robust for locating new boundaries

Segmentation time is reduced from hours to minutes per animal, provided consecutive images are similar

The basic algorithm is open to many generalisations, such as:

- rules for including/excluding air/bone in segmentation
- using previous two images and 3D roughness penalty
- exploiting accumulated database of manual corrections
3. GRAPH CUT ALGORITHM

GC algorithm can also find minimum cost path, by formulating a network flow problem

There are efficient algorithms for finding max flow: source $\rightarrow$ sink

$\text{max flow} \equiv \text{min cut to disconnect source from sink}$

We construct a network for which min cut is the optimal path
First formulation of network

```plaintext
     source
       /        \
      /          \
    5 /            \ 1
   2 /              \ 2
     /                \
  sink
```

Capacity

First formulation of network
max flow

sink ($\sum$ flow = 6)

capacity

source

flow

max flow
source set of nodes & sink set of nodes

sink ($\sum$ flow = 6)

$\sum$ flow = 6
max flow \equiv \min cut

\text{source}

\text{sink} \ (\sum \text{flow} = 6)

\text{capacity}

\text{min cut} = 6

\text{flow}
max flow $\equiv$ min cut

sink $\sum$ flow $= 6$

capacity

min cut $= 6$

flow
For path smoothness we also need diagonal links.
Cut no longer min: $\infty$ cut source $\rightarrow$ sink!
Ford-Fulkerson alg: find path with spare capacity
Ford-Fulkerson alg: path with spare capacity

sink ($\sum$ flow = 6)
Maximise flow along path

sink (\(\Sigma\) flow = 8)
Repeat until no paths have spare capacity

sink \left( \sum \text{flow} = 10 \right)
max flow $\equiv$ min cut

$\sum$ flow = 10

sink ($\sum$ flow = 10)
For a general roughness penalty $g$:

$$
\hat{\beta} = \arg \min_{\beta \in \mathcal{B}^I} \left\{ \sum_{i=1}^{I} f_i(y, \beta_i) + \sum_{|i-j|=1} g(|\beta_i - \beta_j|) \right\}
$$

where $\mathcal{B}$ has $B$ members.

We construct a network with $(B + 1) \times I$ nodes, and edge capacities:

- $C\{\text{source} \to (1, i)\} = \infty$
- $C\{(B + 1, i) \to \text{sink}\} = \infty$
- $C\{(\beta, i) \to (\beta + 1, i)\} = f_i(y, \beta)$
- $C\{(\beta, i) \leftrightarrow (\beta, j) | |i-j|=1\} = g(1)$
- $C\{(\beta, i) \to (\beta - k, k) | |i-j|=1\} = \{g(k + 1) - 2g(k) + g(k - 1)\}$ for all $k \geq 1$

Note: $C \geq 0$ if and only if $g$ is convex
In toy example

\[ g(z) = \begin{cases} 
0 & \text{if } |z| \leq 1 \\
\infty & \text{otherwise}
\end{cases} \]

So edge capacities are:

\[
\begin{align*}
C\{\text{source } \rightarrow (1, i)\} &= \infty \\
C\{(B + 1, i) \rightarrow \text{sink}\} &= \infty \\
C\{(eta, i) \rightarrow (\beta + 1, i)\} &= f_i(y, \beta) \\
C\{(eta, i) \leftrightarrow (\beta, j) \mid |i - j| = 1\} &= 0 \\
C\{(eta, i) \rightarrow (\beta - 1, j) \mid |i - j| = 1\} &= \infty \\
C\{(eta, i) \rightarrow (\beta - k, j) \mid |i - j| = 1\} &= 0 \quad \text{for all } k \geq 2
\end{align*}
\]
For ultrasound application $g(z) = \lambda z^2$

So edge capacities are:

$$C\{\text{source} \rightarrow (1, i)\} = \infty$$
$$C\{(B + 1, i) \rightarrow \text{sink}\} = \infty$$
$$C\{(\beta, i) \rightarrow (\beta + 1, i)\} = f_i(y, \beta)$$
$$C\{(\beta, i) \leftrightarrow (\beta, j) \mid |i - j| = 1\} = \lambda$$
$$C\{(\beta, i) \rightarrow (\beta - k, j) \mid |i - j| = 1\} = 2\lambda \quad \text{for all } k \geq 1$$

For ultrasound image $B = 340$ and $I = 400$

⇒ network has 136K nodes and 46M edges!

Result is same optimal path as DP (though considerably more CPU time)
What if image is 3D?

1D line
\( \beta: 1D \text{ array of 2D vectors} \)
DP: YES
GC: NO

2D surface
\( \beta: 2D \text{ array of scalars} \)
DP: NO
GC: YES
Pepper plants $\Rightarrow$ stereo pair of images $\Rightarrow$ ToF image
4. APPLICATION TO PLANT PHENOTYPING (WITH WAGENINGEN)

Pepper plants

⇒

stereo pair of images

ToF image

3D can be inferred, because: \[ \text{depth} \propto \frac{1}{\text{shift}} \]
We estimate a 2D array of horizontal shifts, $\beta$, to match images $y$ and $y'$:

$$
\hat{\beta} = \arg \min_{\beta \in \mathcal{B}^I} \left\{ \sum_i |y_i - y'_{i+(\beta_i,0)}| + \lambda \sum_{\|i-j\|=1} |\beta_i - \beta_j| \right\}
$$

We can solve using Graph Cut algorithm with edge capacities:

- $C\{\text{source} \rightarrow (1,i)\} = \infty$
- $C\{(B+1,i) \rightarrow \text{sink}\} = \infty$
- $C\{(\beta,i) \rightarrow (\beta+1,i)\} = f_i(y, \beta) \equiv |y_i - y'_{i+(\beta,0)}|$
- $C\{(\beta,i) \leftrightarrow (\beta,j) | \|i-j\|=1\} = \lambda$
- $C\{(\beta,i) \rightarrow (\beta-k,j) | \|i-j\|=1\} = 0$ for all $k \geq 1$
Image size $I = 480 \times 1280$ and $B = \{50, 51, \ldots, 130\}$

$\Rightarrow$ network has 50M nodes, 150M edges!
Yu Song’s (BioSS) results:

He used slightly more complicated $f$’s and $g$’s, and $\beta$-ranges constrained by ToF image.
Foreground leaves can then be identified:

and phenotyping achieved
For example, leaf areas agree well with manual measurements:
5. SUMMARY

In our post-genomic world, phenotyping is often the bottleneck to progress.

Image analysis is one solution, and can sometimes be formulated as:

\[
\hat{\beta} = \arg \min_{\beta \in \mathcal{B}^I} \left\{ \sum_i f_i(y, \beta_i) + \sum_{\|i-j\|=1} g(\|\beta_i - \beta_j\|) \right\}
\]

where \(\beta\) is an array of scalars or vectors.

Global optimum can be found by:

- Dynamic programming if \(\beta\)-array is 1-dimensional
- Graph cut algorithm if \(\beta\)'s are scalars and \(g\) convex

Otherwise, local optima may be found by:

- Iterated DP (Glasbey, 2009, *Stats and Computing*)
- GC swap algorithm (Boykov et al, 2001, *IEEE PAMI*)
However, GC is a much more complicated algorithm than DP

- Which makes it less flexible to adapt to new problems
- Or use as local optimiser when global optimality conditions are not met
- GC cannot handle MRF priors with higher-order neighbourhoods
- Different versions of the GC algorithm run at very different speeds

For further details, see papers and talks on http://www.bioss.ac.uk/people/chris.html
6. EXTENSION OF GC

If the smoothness penalty $g$ is not convex, such as the indicator function:

$$g(z) = \begin{cases} 
0 & \text{if } z = 0 \\
\lambda & \text{otherwise}
\end{cases}$$

GC cannot be used to find the global optimum, because

$$C\{(\beta, i) \rightarrow (\beta - 1, j) \mid ||i - j|| = 1\} = \{g(2) - 2g(1) + g(0)\} = -\lambda$$

However GC can still sometimes find local optima by iterative search
GC swap algorithm (Boykov et al, 2001, *IEEE PAMI*)

0) Initialise $\beta$

1) For every pair of values in $S$, say $(\alpha, \gamma)$

use GC to optimise swapping any $\beta_i = \alpha \rightarrow \gamma$ and any $\beta_i = \gamma \rightarrow \alpha$

Using a network which only includes nodes for which $\beta_i = \alpha, \gamma$

with edge capacities:

$$C\{\text{source } \rightarrow i\} = f_i(y, \alpha) + \sum_{\{j: \|i-j\|=1, \beta_j \neq \alpha, \gamma\}} g(\|\alpha - \beta_j\|)$$

$$C\{i \rightarrow \text{sink}\} = f_i(y, \gamma) + \sum_{\{j: \|i-j\|=1, \beta_j \neq \alpha, \gamma\}} g(\|\gamma - \beta_j\|)$$

$$C\{i \leftrightarrow j \mid \|i - j\| = 1\} = g(\|\alpha - \gamma\|)$$

2) Repeat (1) until no further improvement
Initial path

Network for $\beta : 2 \leftrightarrow 3$
Initial path

Network for $\beta : 2 \leftrightarrow 3$
Initial path

Network for $\beta : 2 \leftrightarrow 3$
Initial path

Network for $\beta : 2 \leftrightarrow 3$

Initial path

Network for $\beta : 2 \leftrightarrow 3$
Improved path

Network for $\beta : 2 \leftrightarrow 3$
For an image smoothing application with non-convex $g$:

$$\hat{\beta} = \arg \min_{\beta \in S} \left\{ \sum_{i} (y_i - \beta_i)^2 + \sum_{\|i-j\|=1} g(\|\beta_i - \beta_j\|) \right\}$$

where  

$$g(z) = \begin{cases} 
0 & \text{if } z = 0 \\
2000 & \text{otherwise}
\end{cases}$$

The GC swap algorithm converged in 5-9 iterations from $\beta = y$ or 1000 random starts, to minimised cost between 430-435.

In comparison, starting from $\beta = y$, Iterated DP converged to 515.

and best found using simulated annealing type methods was 438.