Yeast cell recognition using a dynamic programming algorithm for convex contour extraction

Mats Kvarnström

Centre for Biophysical Imaging
Department of Applied Physics

CHALMERS | GÖTEBORG UNIVERSITY

ITWM

Fraunhofer

CHALMERS
Research Centre
Industrial Mathematics

Personal background

- March 2005 PhD in Mathematical Statistics. Thesis title: "Position estimation and tracking in colloidal particle microscopy.
- April 2005-March 2006 Post-doc at the SWEGENE Centre for Biophysical Imaging at Chalmers
- April 2006- **Applied Researcher at Fraunhofer-Chalmers Research Centre for Industrial Mathematics**

Quantitative imaging

Quantitative imaging is image acquisition together with the subsequent image analysis. It should be **objective** (un-biased) and **more than merely descriptive**.

Most commonly, the analysis is **automated**, which forces the methods to be

- **adaptive** (adapt to different illumination conditions etc.)
- **robust** ("know" when something is wrong. If so: tell the user.)

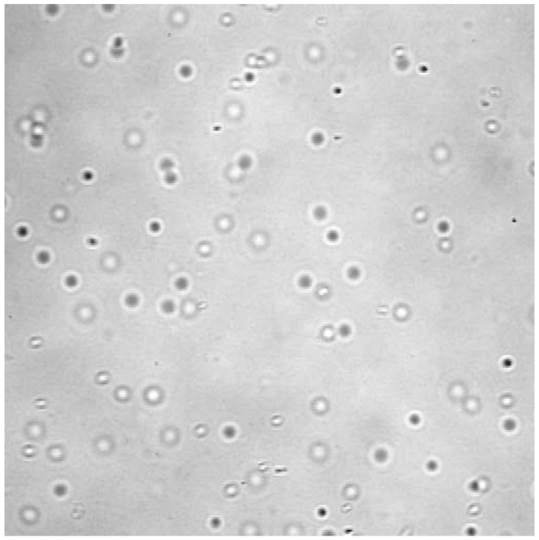
Human vs computer

Even though the human is an extraordinary image analyzer,

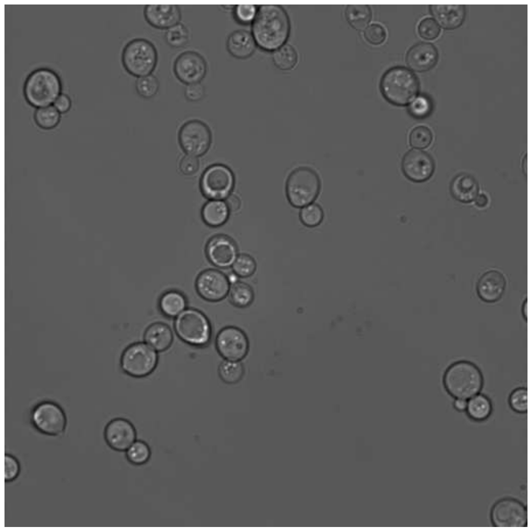
the **benefits** of using computerized image analysis are that they give us

- **more accurate**,
- **faster** and **less tedious**,
- and **more sophisticated**, measurements.

Examples



(a) Particles tracking

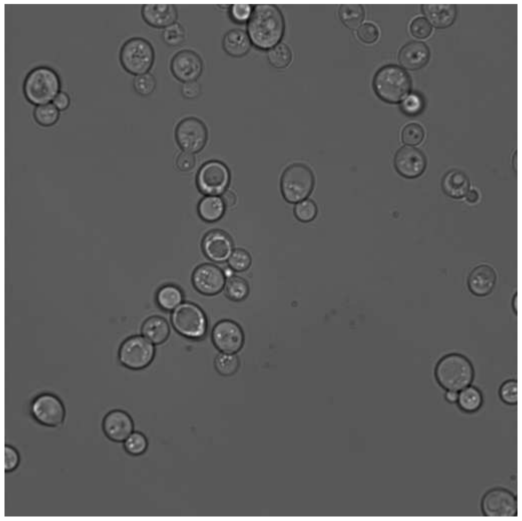


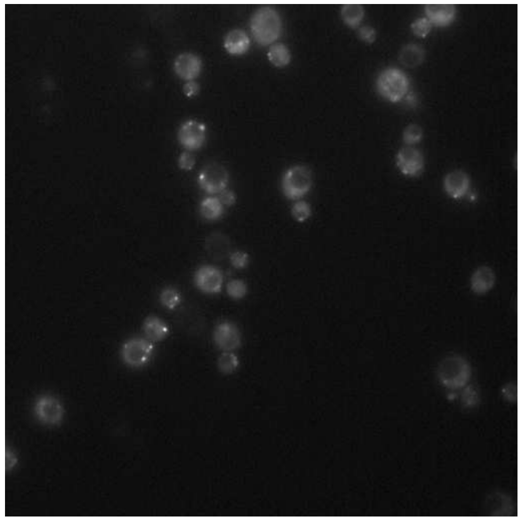
(b) Yeast cell recognition

In particle tracking, typically 50 images per second is produced, and in each image over 100 particles are present.

In yeast cell recognition, we study 20-100 cells over several hours, using both bright-field and fluorescence images.

Yeast Cell Study


(a) Bright-field


(b) Fluorescence

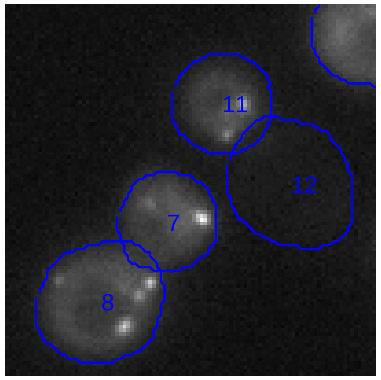
Goal for each pair of images: Measure the amount of fluorescence signal from each cell.

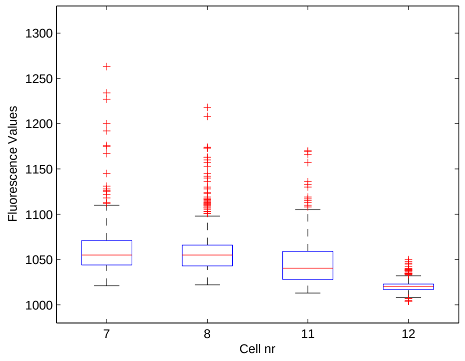
Main goal: Continue to do this as time passes (**2-D lapse microscopy**) and/or compare with large amounts of gene-disruptants of cells (**high-throughput screening**).

The amount of data is huge here, so this must be done via **some kind of automation**.

Software output: example

Example: Compare the fluorescence signals from different cells over time.

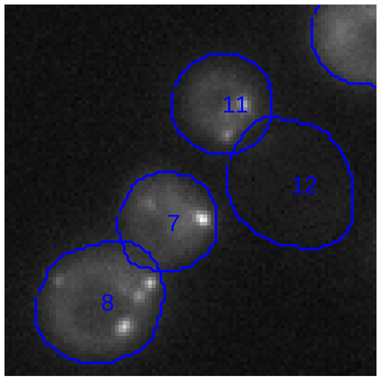


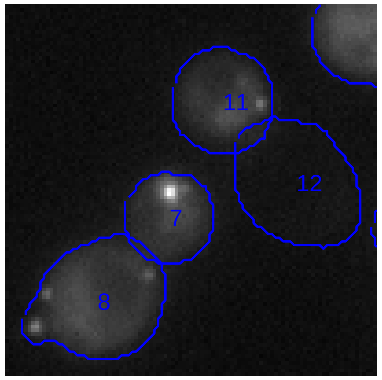


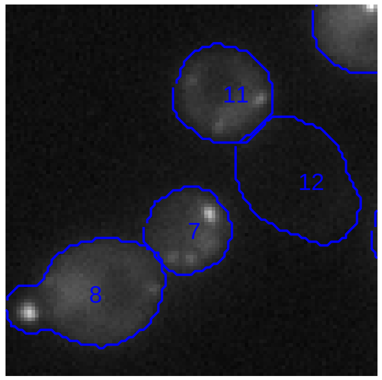
time 0

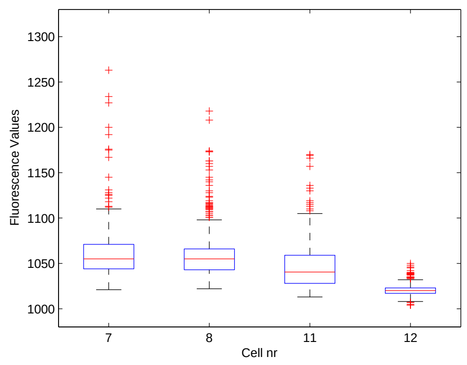
Software output: example

Example: Compare the fluorescence signals from different cells over time.

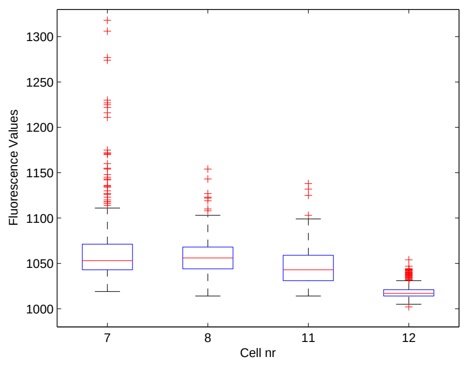




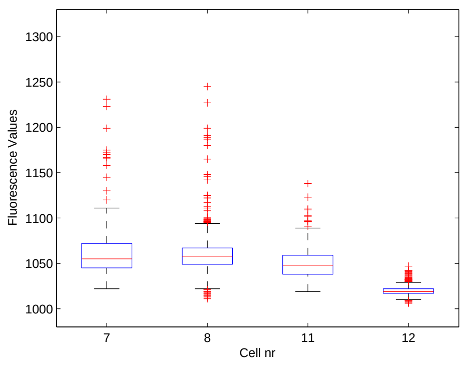




time 0



time 4



time 8

Used paradigm for yeast cell studies

Before one can proceed with any kind of measurements, the cells have to be found. One common method to find is fluorescent staining of the cell membrane.

However, for ***in vivo*** cell studies, **fluorescent staining** of the cell membranes is generally **not appropriate**.

Therefore we use the following methodology:

- Use bright-field images for cell recognition
- Fluorescence images are used only for measuring the protein expression signals

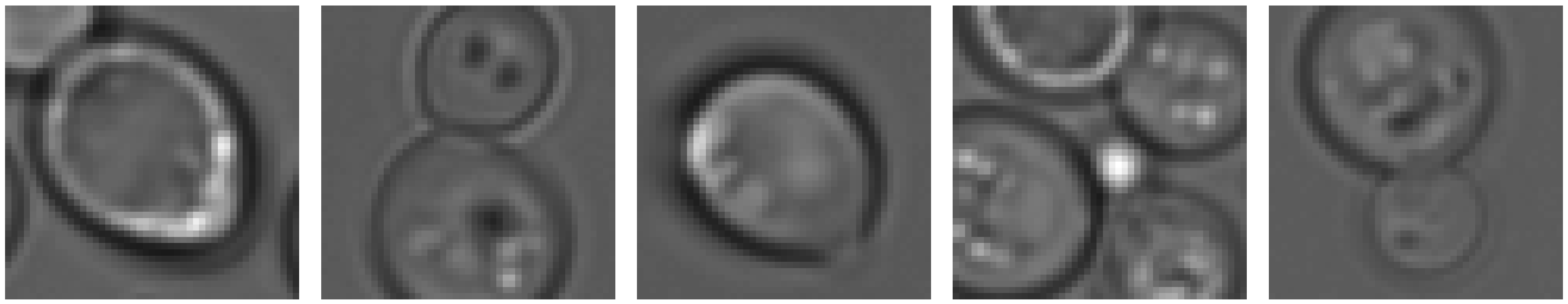
Cell Recognition

The **cell recognition** step will be done in the bright-field image.

For humans, it is (in most cases) easy to identify the were the cells are.

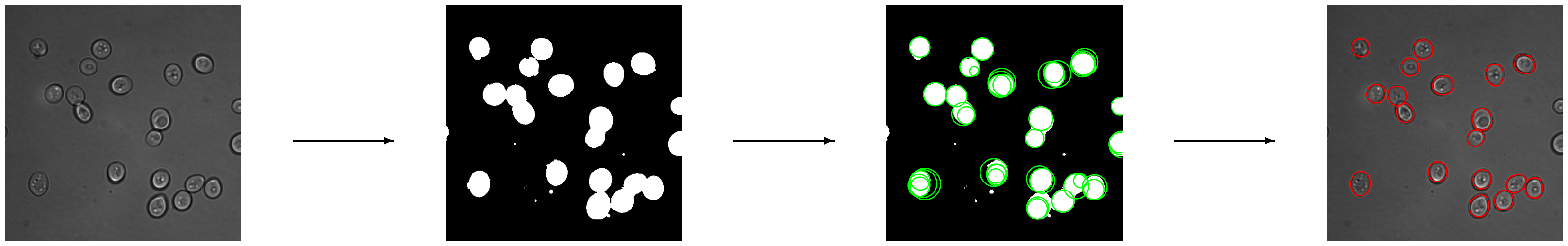
However, it is not a trivial problem to "tell" the computer how to find them, due to the difference in appearance of cells and that they might overlap.

Example cells:



Basic steps for cell recognition

- **Segmentation:** differentiate foreground (the cells) from background. This step must be able to adapt to density of cells and different illumination conditions.
- **Candidate Cell Centres:** Finding suitable locations in the segmented image where cells could be located. Here: see where circles can fit.
- **Find Cell Contours:** For each candidate centre, find a connected sequence of pixels surrounding the centre. Use criteria on both shape and pixel intensities.

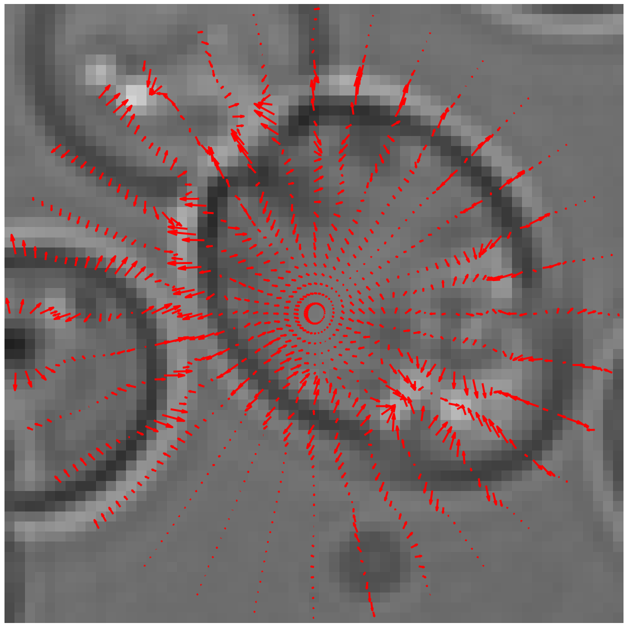


Finding contours surrounding the candidate centres

For each candidate centre, find a reasonable cell contour surrounding this point. Look at the **directional derivatives** at points along the rays emanating from a candidate centre:

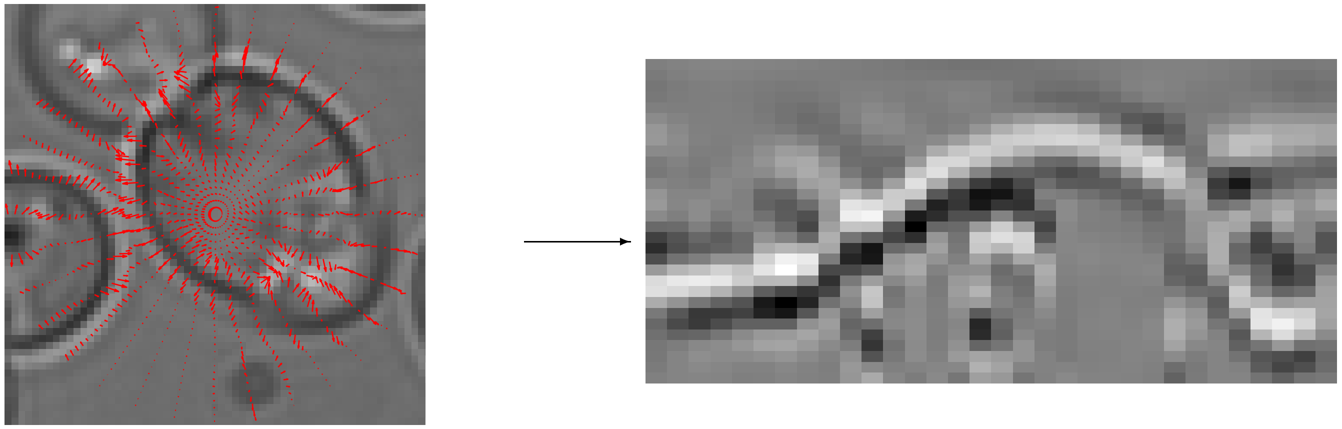
A **reasonable contour** around this cell could be to join the positions along the rays where the directional derivatives change from large magnitude pointing inwards, to large magnitude pointing outwards.

That is, where the projection of the directional second derivative, along the rays pointing outwards, is large.



Polar plot

First, let us plot the second derivative values at 30 equally spaced points along the 32 rays (counter-clockwise) in a **polar plot:**.



Reference system: For each of the 32 rays, pick a point such that in the end, these points represents a cell contour.

However, in all generality, **the state space** to search for an optimal solution, is **huge**.

Conditions on contour shape

There are a lot of conditions on the shape of the cell contour.

The two most obvious ones:

- **Closed** and **continuous**: no jumps along the contour.

Others :

- **Convex** or
- **Elliptical** (Yeast cells are typically elliptical)

We probably also want to have restrictions on the

- **size** of the cell; not too big and not too small

How do we incorporate conditions like these while keeping the **computational complexity low**?

Path from left to right: dynamic programming

If we restrict ourselves to the discrete contour coordinates in the polar plot, there is a method for finding the "cheapest" path from left to right (in the polar plot).

It is called **dynamic programming** (the Viterbi algorithm).

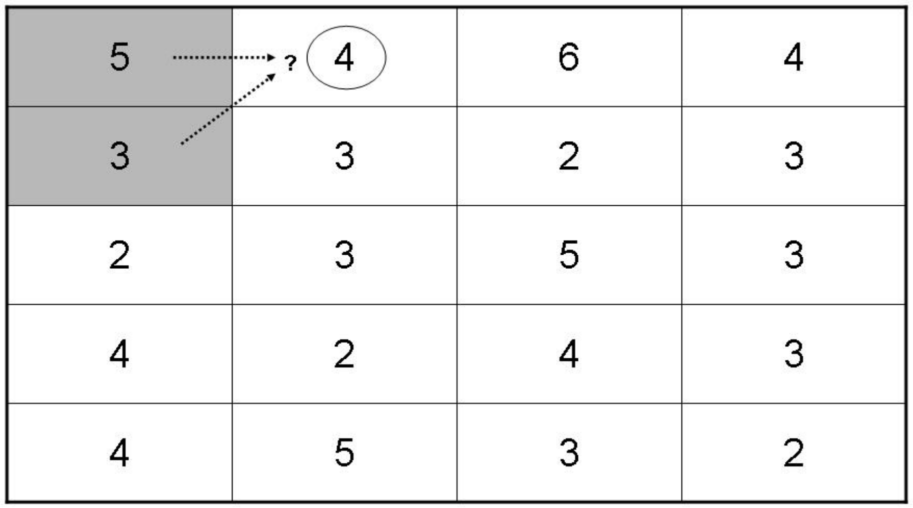
Dynamic Programming (simple example)

Task: Find the path from left to right with lowest cumulative cost!
(Only straight and diagonal movements are allowed.)

5	4	6	4
3	3	2	3
2	3	5	3
4	2	4	3
4	5	3	2

Dynamic Programming (simple example)

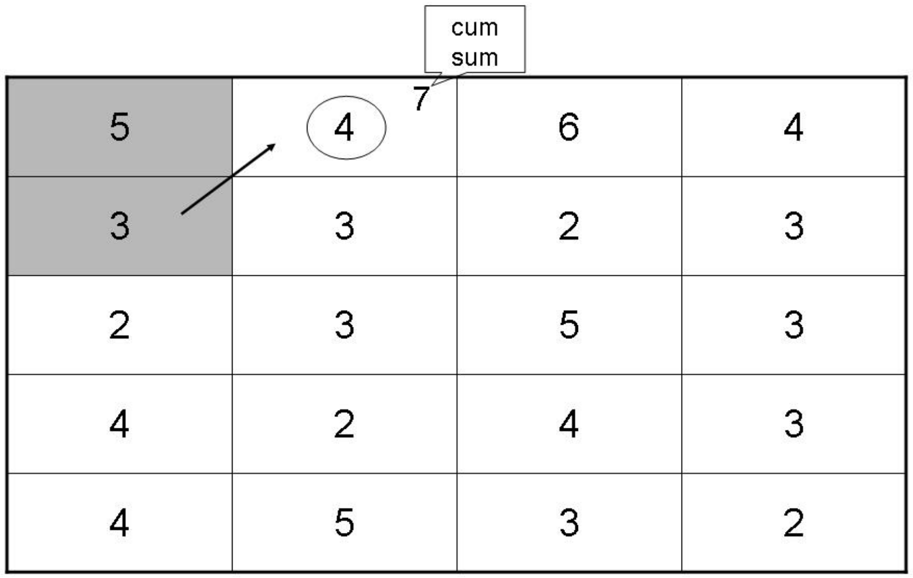
Question: In column 2, row 1: if this is cell is part of the optimal solution, what is the optimal path coming to this cell (i.e. from column 1)?



5	? 4	6	4
3	3	2	3
2	3	5	3
4	2	4	3
4	5	3	2

Dynamic Programming (simple example)

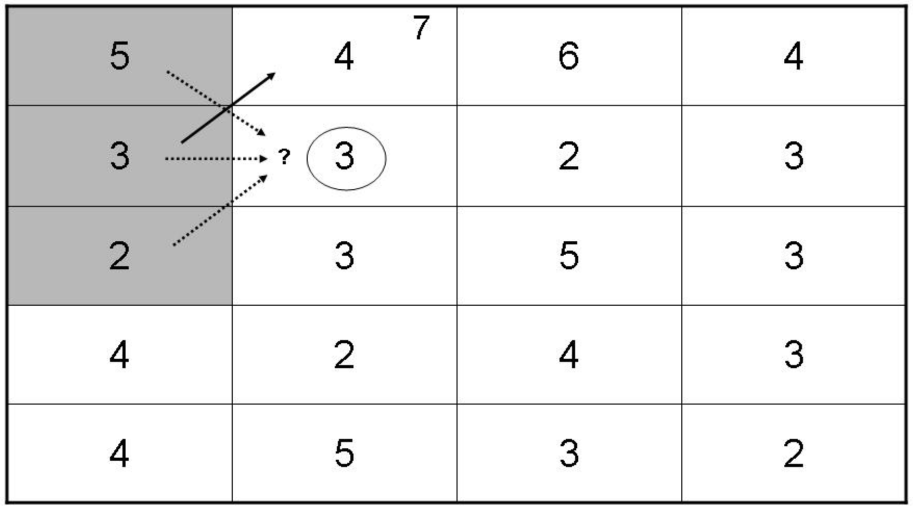
Answer: We must have come from row 2 in column 1. Make a note of this path and the cumulative sum up to this point.



5	4	6	4
3	3	2	3
2	3	5	3
4	2	4	3
4	5	3	2

Dynamic Programming (simple example)

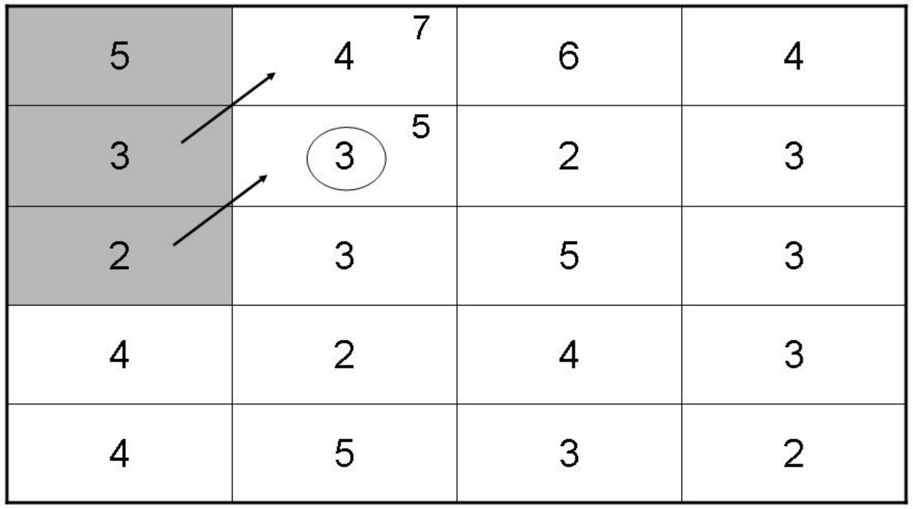
Now do the same analysis for the cell in row 2 in column 2. Here we have three candidate cells.



5	4	6	4
3	3	2	3
2	3	5	3
4	2	4	3
4	5	3	2

Dynamic Programming (simple example)

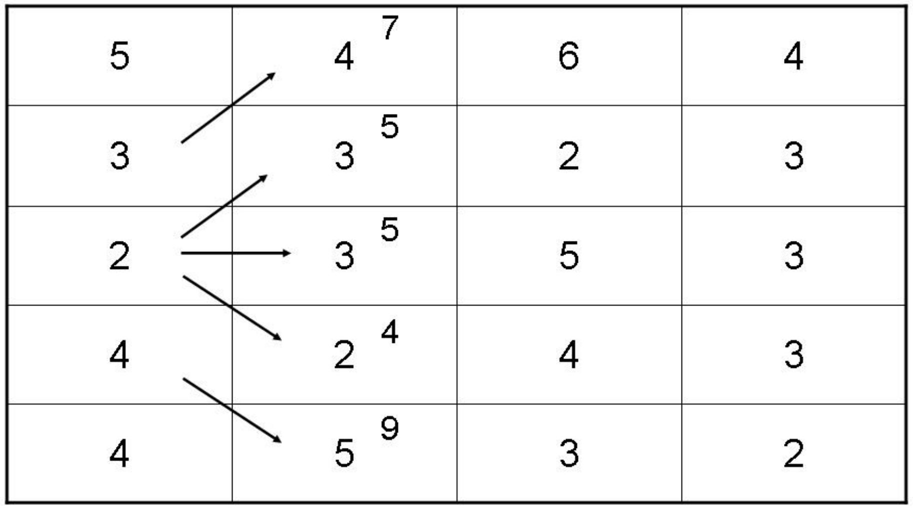
Again, make a note of the cheapest path and the cumulative sum.



5	4	6	4
3	3	2	3
2	3	5	3
4	2	4	3
4	5	3	2

Dynamic Programming (simple example)

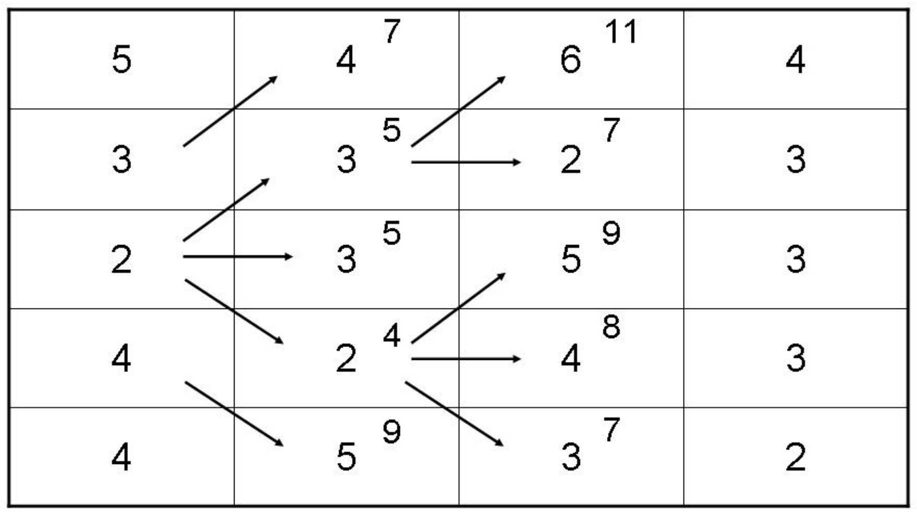
... and continue with the rest of the cells in the column.



5	4 ^{7}	6	4
3	3 ^{5}	2	3
2	3 ^{5}	5	3
4	2 ^{4}	4	3
4	5 ^{9}	3	2

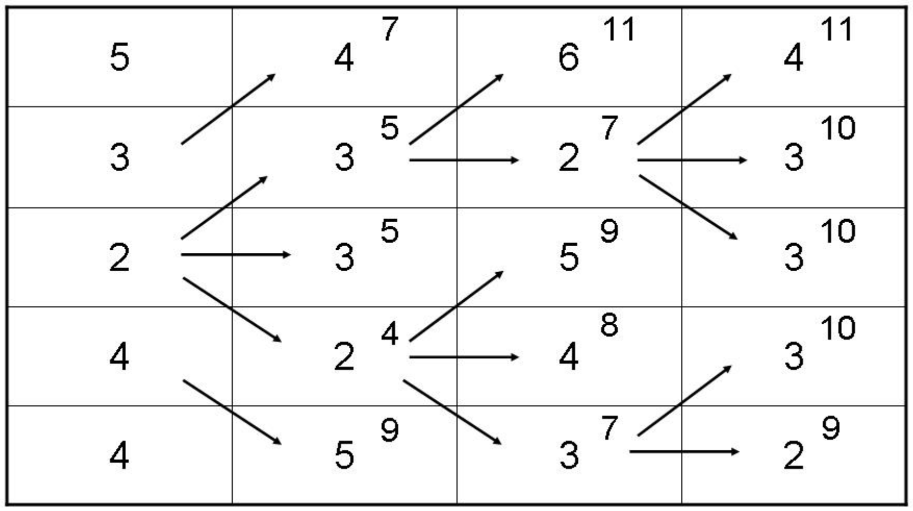
Dynamic Programming (simple example)

In column 3, do the same. Here, be sure to compare the cumulative sums for the previous cell candidates.



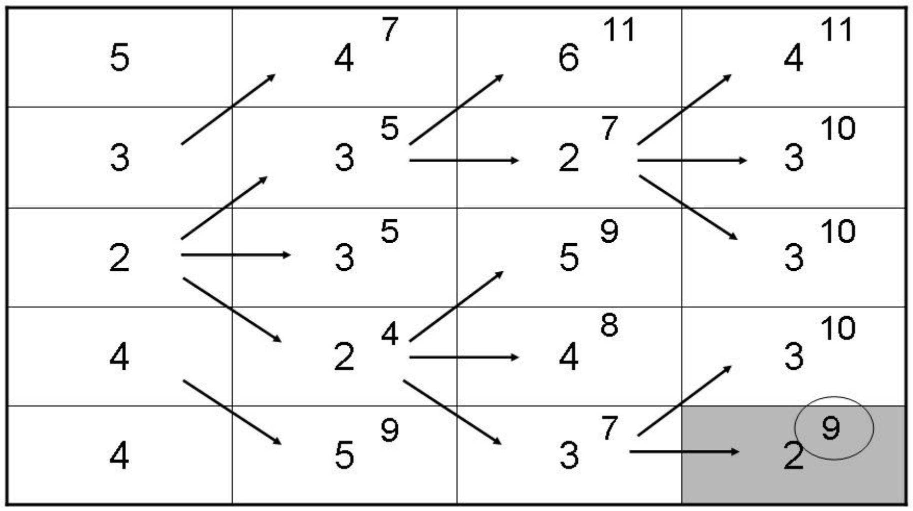
Dynamic Programming (simple example)

... and the same again for column 4. Now the first step of the algorithm is completed.



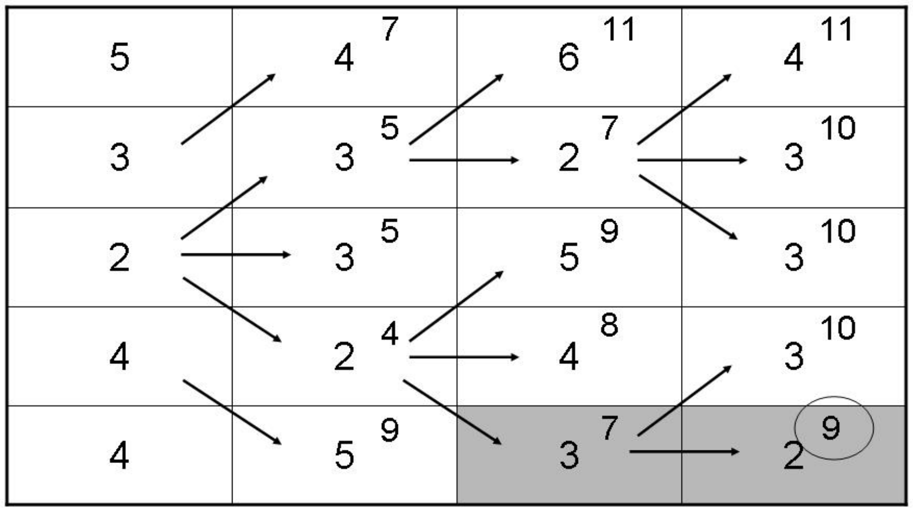
Dynamic Programming (simple example)

Step 2 is to find the cell in the last column with the minimum cumulative sum, which is the cell in row 4.



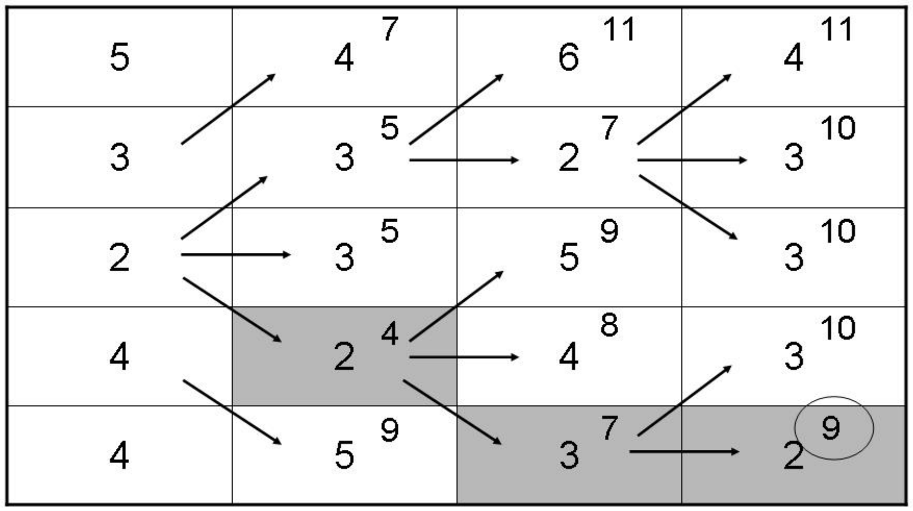
Dynamic Programming (simple example)

Step 3 is to trace back in the opposite direction of pointing arrows...



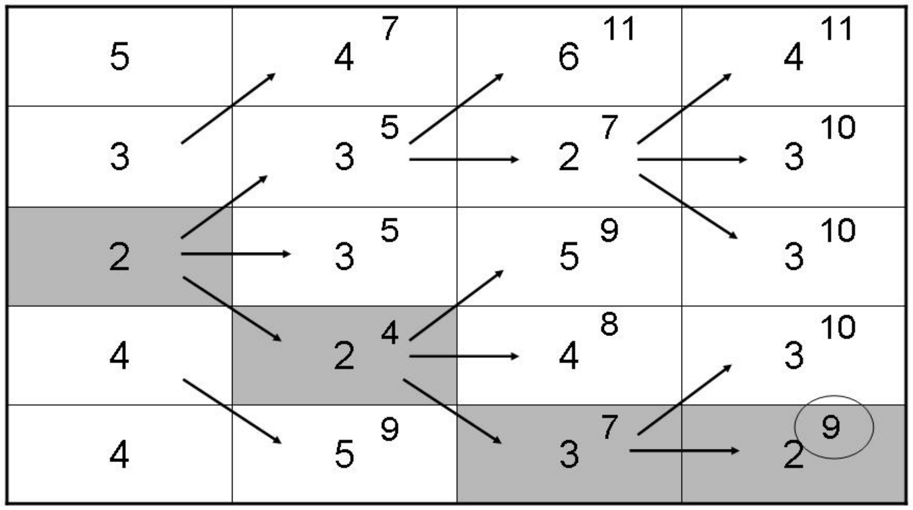
Dynamic Programming (simple example)

Step 3 is to trace back in the opposite direction of pointing arrows...



Dynamic Programming (simple example)

Step 3 is to trace back in the opposite direction of pointing arrows... After arriving at the start column, we are done. This is the dynamic programming scheme.



Dynamic programming (restrictions)

There are certain **restrictions**:

- First, a "direction" is needed: go from left to right in the polar plot
- "Markovianity": the cost of joining two points (along two rays), given that the previous point (according to the chosen direction) is a part of the solution, must depend on these two points only.

Unmodified, the second restriction implies that it is **not even possible to guarantee a closed contour**. This is of course a serious drawback of the method.

Dynamic Programming (closed and continuous contour)

Suppose that we want to have a **closed** and **continuous** contour.

- Continuity can, heuristically, be imposed by **not letting the solution move "too far" between ray points** along the contour. This we did in the simple example.

This does however not always work, e.g. for elliptical contours, where the candidate centre is located to far from the centre (of gravity) of the ellipse.

(Note that we have not defined what we mean with a continuous contour in discrete polar coordinates...)

Dynamic Programming (closed and continuous contour)

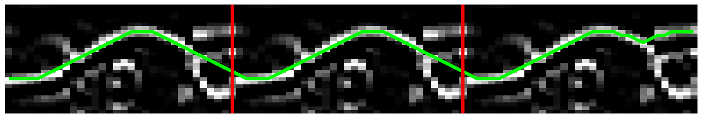
Closedness seems harder. However:

- We can solve the problem for each initial point in the first column separately, putting infinite weights in the cost matrix for all other points in the last column.
- A heuristic resort for closedness, is to let the algorithm work on **several laps**, e.g. three, and the solution be the middle lap.

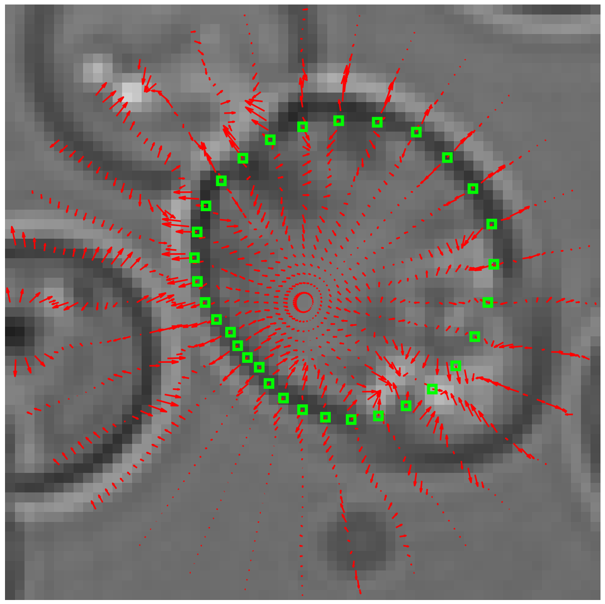
The first method will guarantee a closed contour, but will severely increase the computational cost.

Resulting Erroneous contour

Here, we have used the three laps method for a closed contour, and the restriction of not letting the optimal contour change more than one one unit (radial wise) between two consecutive points.



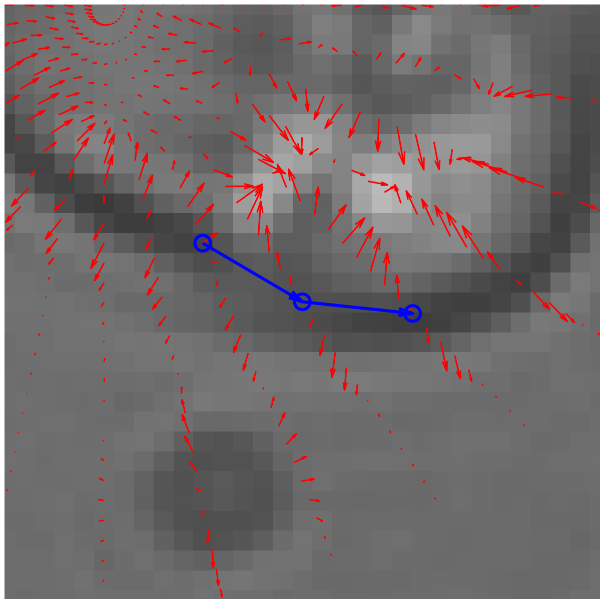
$\longrightarrow$



Dynamic Programming (Convex Contour)

Suppose instead that that we want to have a **convex** contour.

For a **convex contour**, the vectors joining three consecutive points (in the polar plot) must **always take "left-turns"**:



Remark 1: This is a **necessary condition for convexity**.

Remark 2: Together with **closedness**, it is also a **sufficient condition for convexity**. (But remember that closedness is however not guaranteed.)

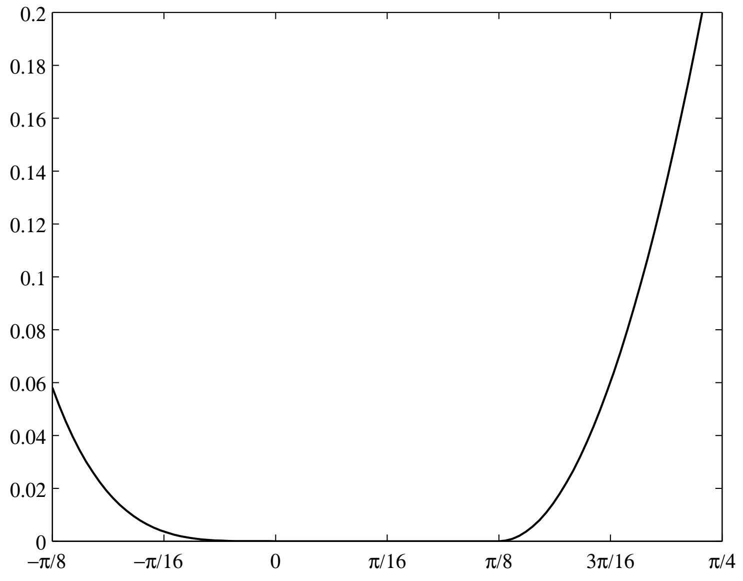
Dynamic Programming (Convex Contour)

One problem remains: Convexity does not follow the Markovian rule needed for dynamic programming.

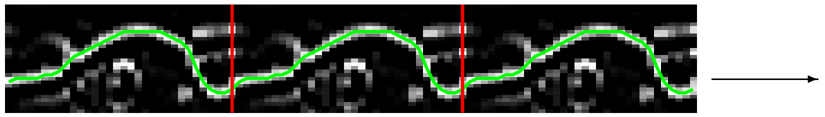
Easy remedy: **Expand the state space to pairs of two consecutive points**. This is an quadratic increase in the number of states, but the problem is still manageable.

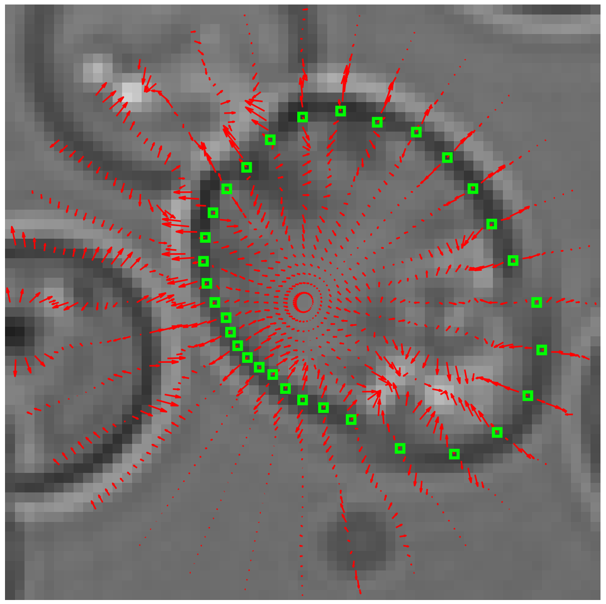
We use a **soft penalizing condition** on turning angles: penalty on right turns and left turns sharper than $\pi/8$ (when 32 rays are used).

(Remember that $2\pi/32$ is the nominal turn rate for a circle.)



Resulting contour





Dynamic Programming (Conclusions)

Compared to using deformable templates in a Bayesian framework (MCMC and ICM) for finding cells in microscope images (e.g. Rue & Husby (1998), Mardia et al. (1997), Gray (1999)) and Active Contour Models (snakes), the benefits and drawbacks are

+ Deterministic stopping rule
+ Repeatable outcome
+ Fast

- Local rules only $\longrightarrow$ closedness not guaranteed
- We have no continuous representation of the contour.

Possible modifications to our method

- Add/combine the angle penalty with conditions on the change in radial distance.
- Modify and iterate if global conditions are not met.

Acknowledgements

- This work was conducted using a post-doc grant from SWEGENE.
- The yeast cell images, were produced by **Alfredo Diez**, my post-doc colleague at the Centre for Biophysical Imaging.