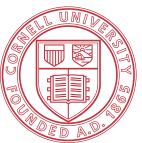
CS5112: Algorithms and Data Structures for Applications

Lecture 22: Computability; Max-flow for vision

Ramin Zabih

Some content from: Wikipedia/Google image search;
Olga Veksler & Yuri Boykov



Administrivia

- Wednesday lecture: review session
  - Come with questions!
- Monday: final exam
  - Students will be assigned to the auditorium or to 071
  - We will let you know by email which room you are in
- The following will not be on the exam
  - Previous lecture (SGD)
  - Second half of today's lecture (max flow)
  - Lower bounds and approximation algorithms

Lecture Outline

- Bad news: Some problems turn out to be impossible to solve
  - Uncomputable problems
  - Ex: determining if a hash function has collisions
- Good news: Some problems are quite solvable
  - Binary image denoising
  - Max flow/min cut (“Graph cuts”)
  - Applications and extensions

The bad news

- Some problems are impossible to solve
- You need to be able to recognize them

Uncomputability

- Some well-defined problems are simply impossible
  - No matter what your hardware, they are uncomputable
- You need to be able to recognize them
  - Similar to NP-hardness
  - BUT: NP-hard problems can be solved
    - just not very fast

Halting problem

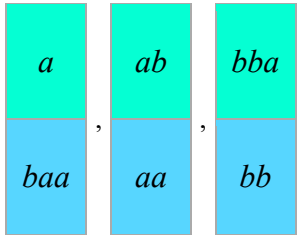
- Most famous example: the halting problem
  - Does a particular program run forever on a specific input?
- This seems easy to avoid, but…
- Consider a new problem such that, if you could solve it, would allow you to solve the halting problem
- There are some very subtle example of this
  - Just like with NP-hardness

When is a problem uncomputable?

- This is very difficult, but there are some rules and clues
- Any non-trivial property of a program is uncomputable
  - Rice's theorem
- Anything you can solve with exhaustive search is computable
  - Infinite/unbounded input is a key clue!
- Small differences in problem formulation can change a computable problem into an uncomputable one

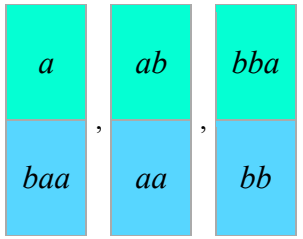
A children's game

- We are given blocks with symbols such as a,b,c. Each block has a top and a bottom.
- There are certain types of blocks, such as a block with "ab" above "bc".
- We have as many blocks of each type as we want.
- Can we find a series of blocks so that the top and bottom symbols are the same?



Game examples

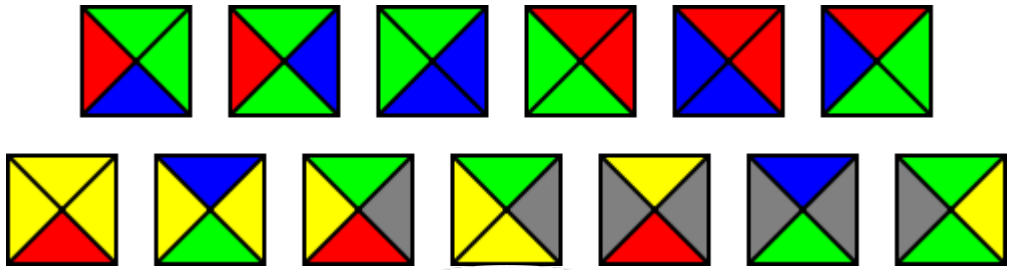
- Example 1:

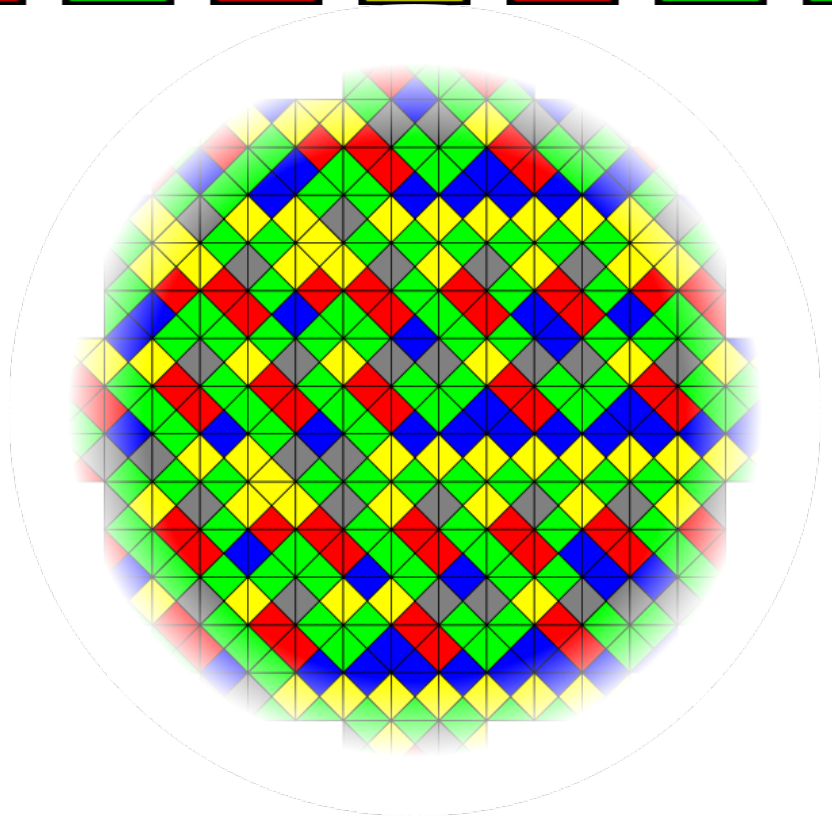


Can we solve the children's game?

- For a binary alphabet (example 1) this is decidable
- For an alphabet with 7 characters or more?
  - undecidable!
- For an alphabet with 3 characters (example 2)?
  - open question
- What if we can use no more than k blocks, including copies, in our solution?
  - It's decidable but NP-hard

Tiling the plane





Source

The good news

- Some challenging looking problems are solvable
- Let’s start by showing off a cool vision application

Interactive Digital Photomontage

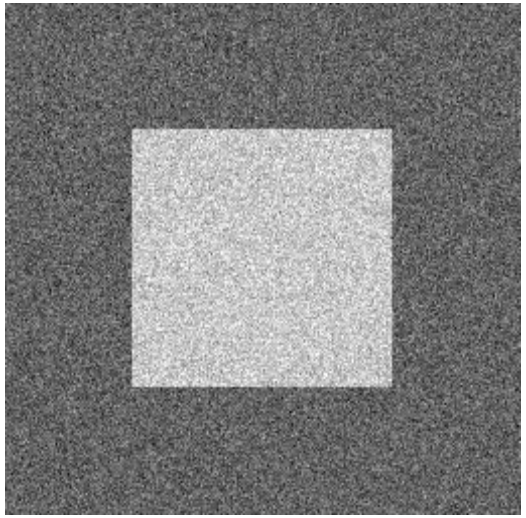
Aseem Agarwala, Mira Dontcheva,
Maneesh Agrawala, Steven Drucker, Alex Colburn,
Brian Curless, David Salesin, Michael Cohen

University of Washington & Microsoft Research

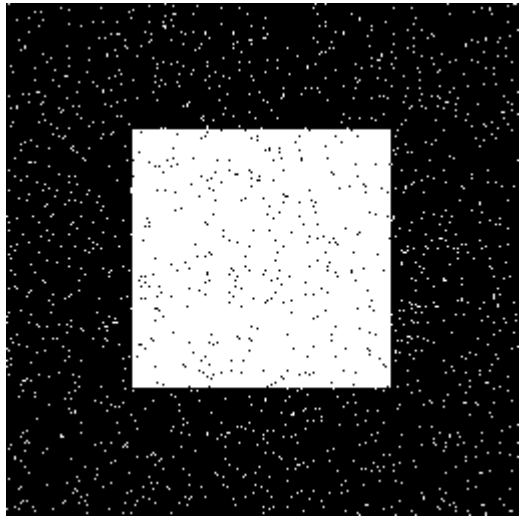
Image denoising

- Suppose we want to find a bright object against a dark background
  - But some of the pixel values are slightly wrong

Input



Best thresholded image



Optimization viewpoint

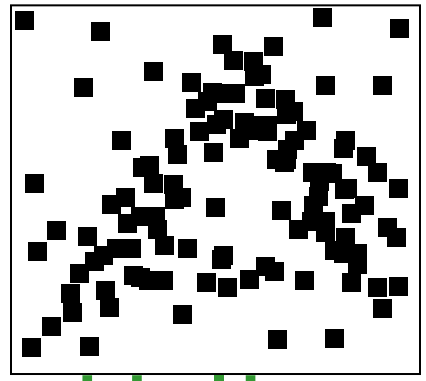
- Find best (least expensive) binary image
  - Costs: C1 (labeling) and C2 (boundary)
- C1: Labeling a dark pixel as foreground
  - Or, a bright pixel as background
- If we only had labeling costs, the cheapest solution is the thresholded output
- C2: The length of the boundary between foreground and background
  - Penalizes isolated pixels or ragged boundaries

Binary labeling problems

- There is a natural multi-label version
  - But 2 labels are surprisingly important
    - Lots of nice applications
    - Same basic ideas used for more labels
- Fast exact solution!
  - Turn a hard problem into a problem you can solve (reduction)
    - Due to Hammer (1965) originally
    - Job scheduling problems: Stone (1977)
    - Images: Greig, Porteus & Seheult (1989)

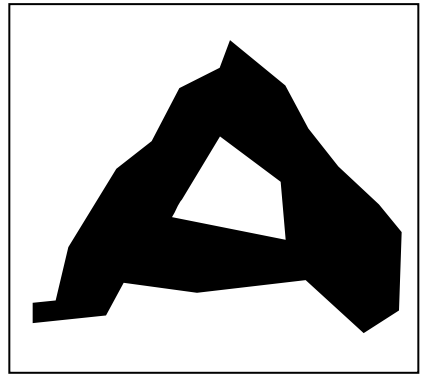
Binary Labeling: Motivation

- Suppose we receive a noisy fax:
  - Some black pixels in the original image were flipped to white pixels, and some white pixels were flipped to black



original image

- We want to try to clean up (or **restore**) the original image:
- This problem is called **image restoration (denoising)**

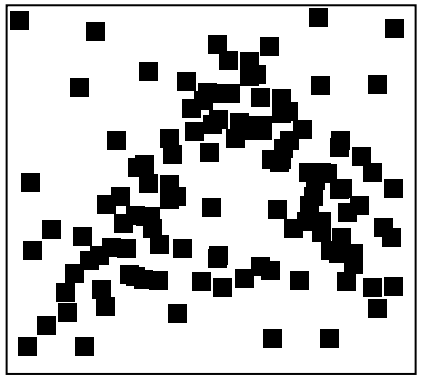


restored image

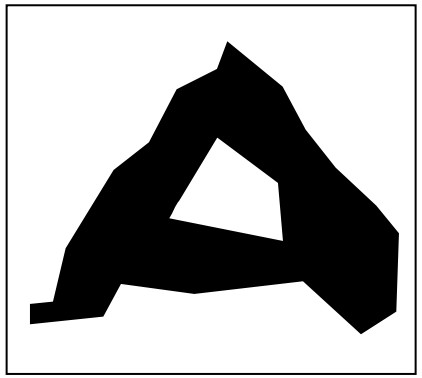
Binary Labeling Problem: Motivation

- Fax image is binary: each pixel is either
  - black (stored as 0)
  - or white (stored as 1)

- What we know:
  1) In the restored image, each pixels should also be either black (0) or white (1)
  2) Data Constraint: if a pixel is black in the original image, it is more likely to be black in the restored image; and a white pixel in the original image is more likely to be white in the restored image
  3) Prior Constraint: in the restored image, white pixels should form coherent groups, as should black pixels



original image



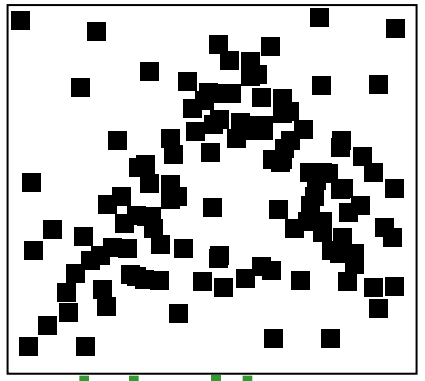
restored image

Binary Labeling Problem

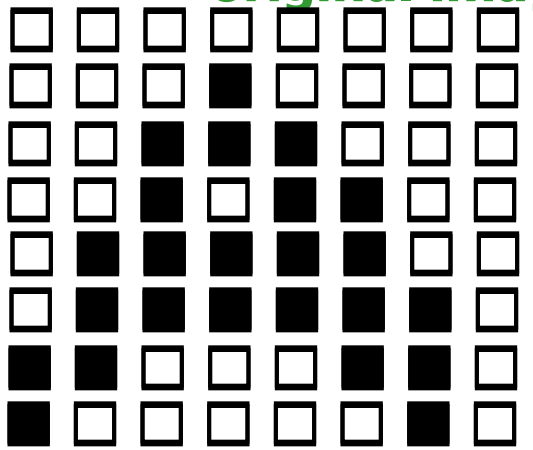
- We can formulate our restoration problem as a labeling problem:
  - Labels: black (0) and white (1)
  - Set of sites: all image pixels
    - Will use “sites” or pixels interchangeably

Assign a label to each site (either black or white) s.t.

- If a pixel is black (white) in the original image, it is more likely to get the black (white) label
- Black labeled pixels tend to group together, and white labeled pixels tend to group together



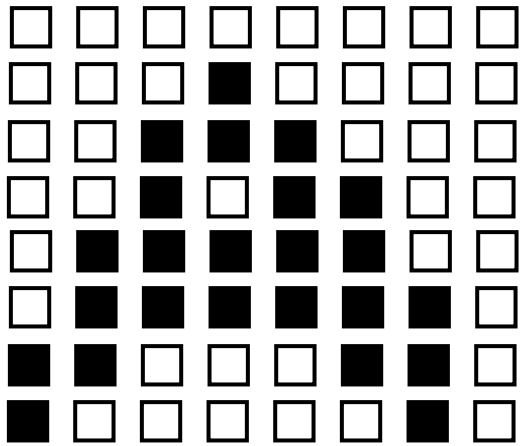
original image



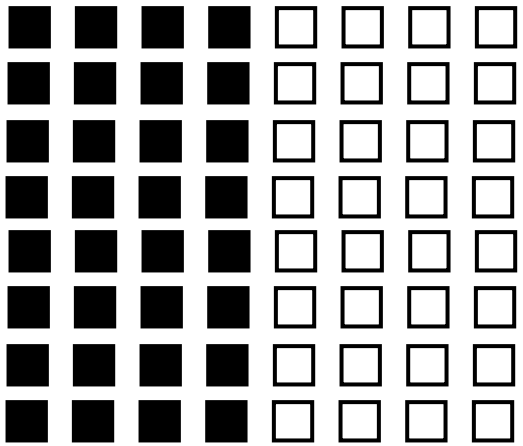
set of sites *P*, and one possible labeling

Binary Labeling Problem: Constraints

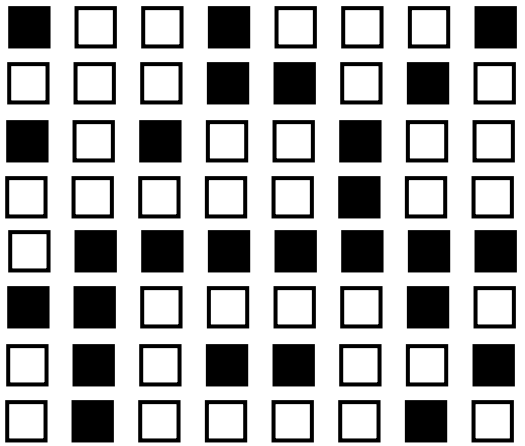
- If a pixel is black (white) in the original image, it is more likely to get the black (white) label
- Black labeled pixels tend to group together, and white labeled pixels tend to group together



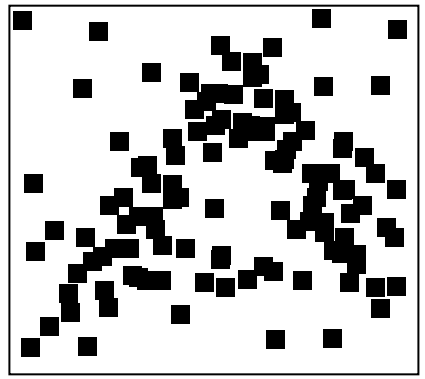
good labeling



bad labeling
(constraint 1)



bad labeling
(constraint 2)



original image

Binary Labeling Problem: Data Constraints

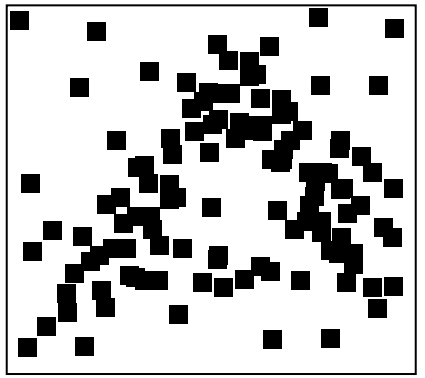
- How can we find a good labeling?
  - i.e., satisfying our constraints

- Formulate and minimize an energy function on labelings L:

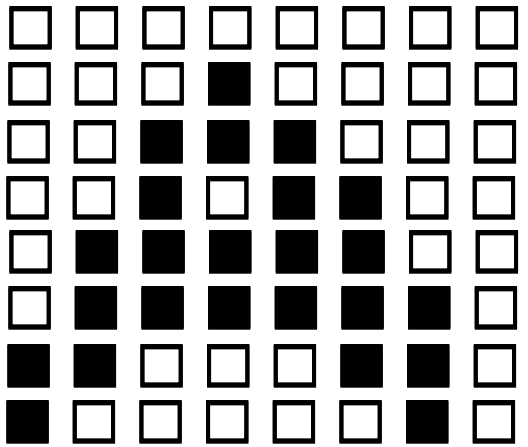
  Let L_p be the label assigned to pixel p
  - $L_p = 0$ or $L_p = 1$

  Let $I(p)$ be the intensity of pixel p in the original image

  Data constraint is modeled by the Data Penalty $D_P(L_P)$
  - $D_P(0) < D_P(1)$ if $I(p) = 0$
  - $D_P(0) > D_P(1)$ if $I(p) = 1$



original image

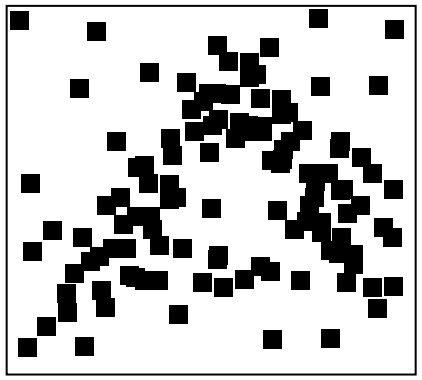


a good labeling L

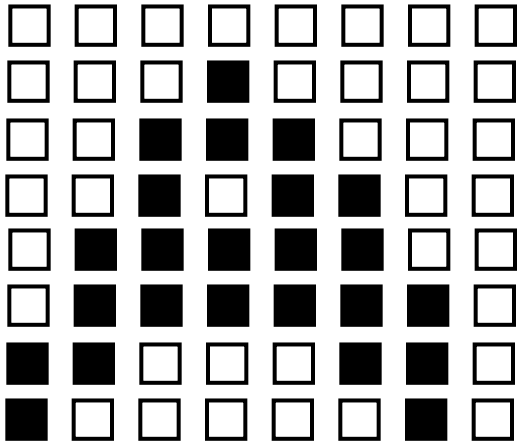
Binary Labeling Problem: Data Constraints

- In our example, we can make
  - if I(p) = black then
    - D_P(black) = 0
    - D_P(white) = 4
  - if I(p) = white then
    - D_P(black) = 4
    - DP(white) = 0

- To figure out how well image as a whole satisfies the data constraint, sum up data terms over all image pixels: $\sum_p D_P(L_p)$



original image

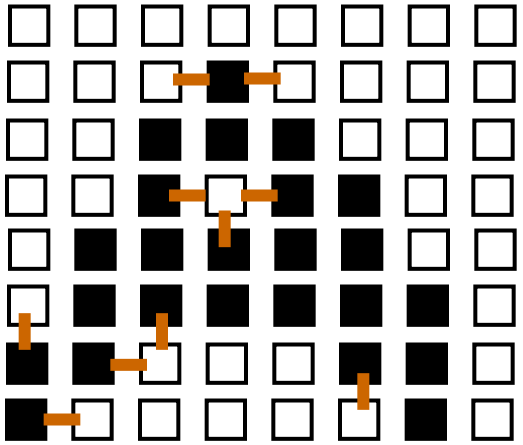


a good labeling ***L***

Binary Labeling Problem: Prior Constraints

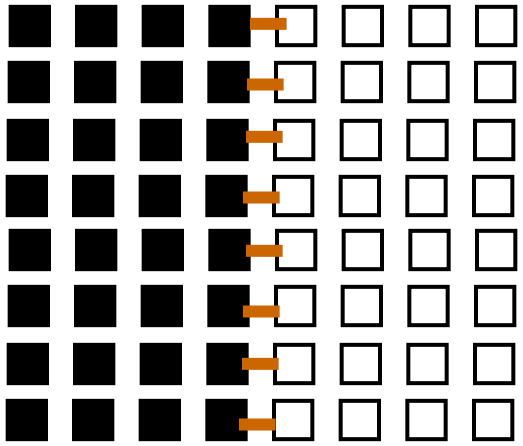
- To model our prior constraints, simply count the number "discontinuities" in a labeling L:
  - There is a discontinuity between pixels p and q if labels p and q have different labels
  - The more pixels with equal labels group together in coherent groups, the less discontinuities there are

some discontinuities are marked



31 discontinuities

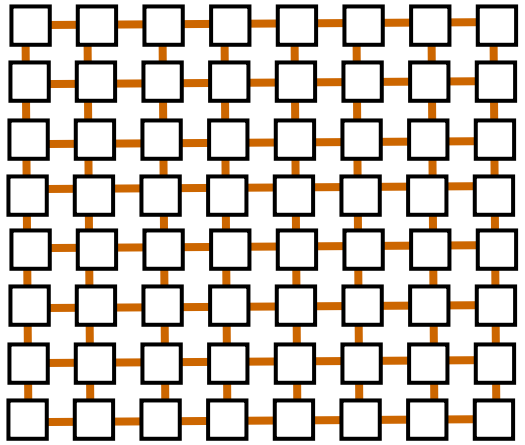
all discontinuities are marked



8 discontinuities

Binary Labeling: Data Constraints

- How to count the number of discontinuities in a labeling L?
  - Let N be the set of all adjacent pixels
  - Thus our N consists of edges between immediately adjacent pixels



N consists of all edges

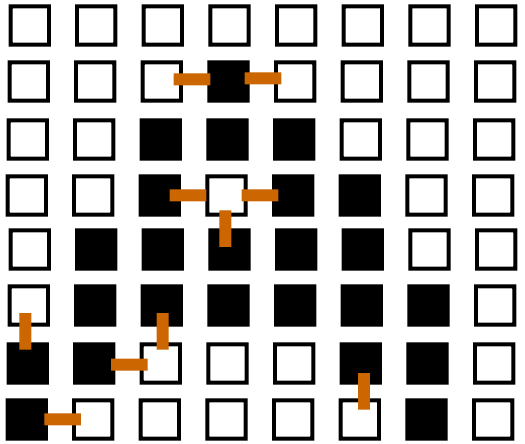
Let $\mathcal{I}(b)$ be the identity function

$\mathcal{I}(b)$ =1 when its argument b is true

$\mathcal{I}(b)$ =0 when its argument b is false

To count all discontinuities in a labeling L,

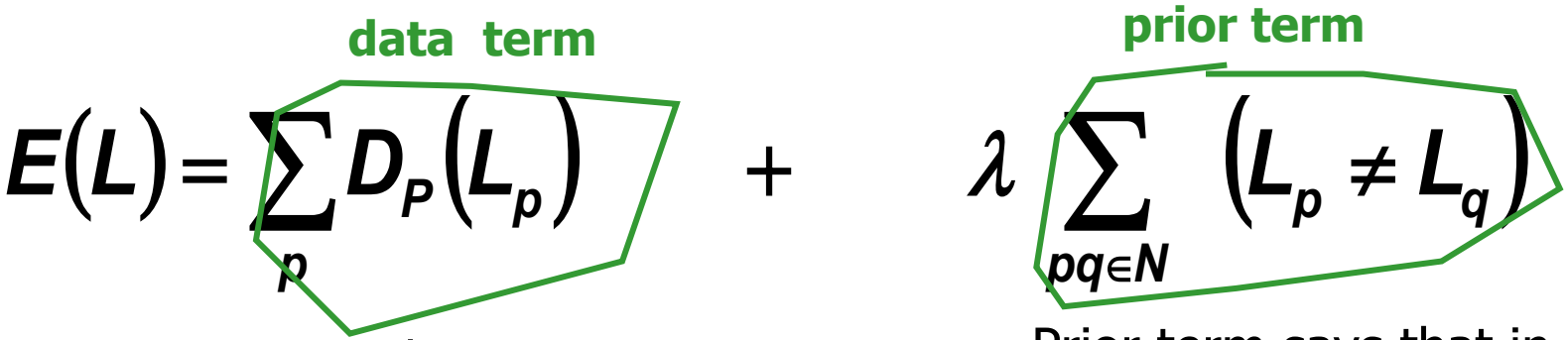
$$\sum_{pq \in N} \left(L_p \neq L_q\right)$$



31 discontinuities

Binary Labeling Problem: Energy Formulation

- Our final energy, which takes into consideration the data and the prior constraints is:



Data term says that in a good labeling L pixels should be labeled as close as possible to their colors in the original (noisy) image

Prior term says that in a good labeling L pixels should be labeled as to form spatially coherent blocks (as few discontinuities as possible)

Binary Labeling Problem: Energy Formulation

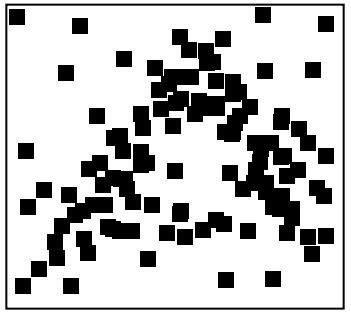
data term

prior term

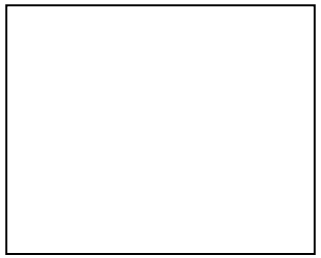
$$E(L) = \sum_p D_P(L_p) + \lambda \sum_{pq \in N} (L_p \neq L_q)$$

- Note that data term and prior term want different things:

Best labeling as far as the data term is concerned is the original image:



Best labeling considering only the prior term is completely black or completely white:



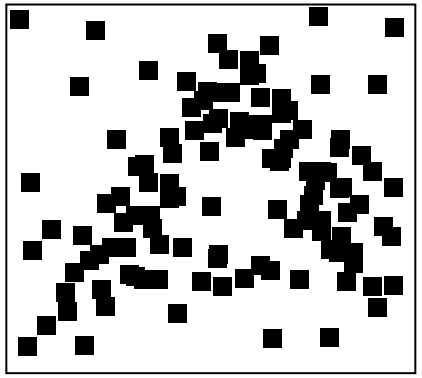
Binary Labeling Problem: Energy Formulation

data term prior term

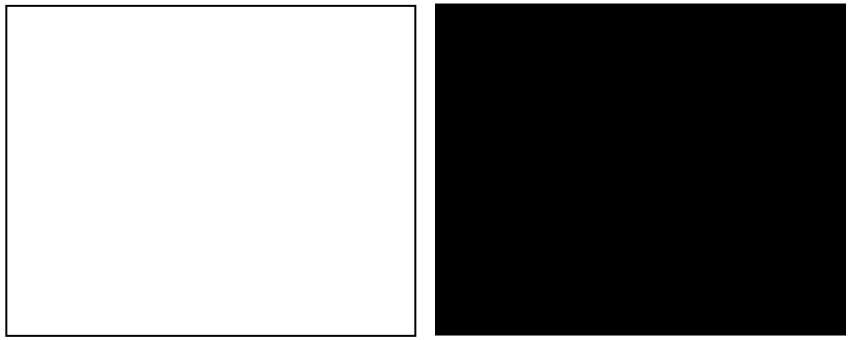
$$E(L) = \sum_{p} D_P(L_p) + \lambda \sum_{pq \in N} (L_p \neq L_q)$$

- λ serves as a balancing parameter between the data term and the prior term:
  - The larger the λ, the less discontinuities in the optimal labeling ***L***
  - The smaller the λ, the more the optimal labeling ***L*** looks like the original image

λ=0



λ=huge number



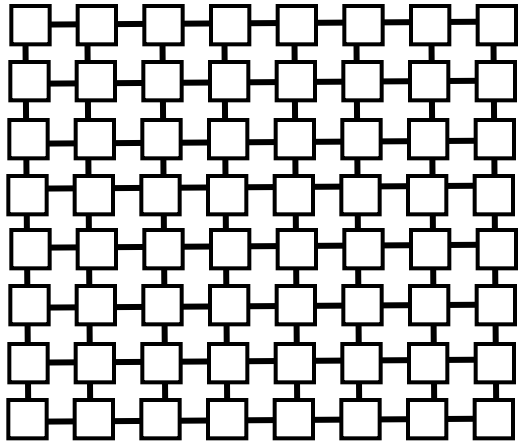
The Hard Computational Problem

data term prior term

$$E(L) = \sum_{p} D_P(L_p) \quad + \quad \lambda \sum_{pq \in N} (L_p \neq L_q)$$

- Now that we have an energy function, the big question is how do we minimize it?

- Exhaustive search is exponential: if ***n*** is the number of pixels, there are 2^n possible labelings ***L***



Graph Cuts for Binary Energy Minimization

- Can minimize this energy exactly with graph cuts!

$$E(L) = \sum_p D_p(L_p) + \lambda \sum_{pq \in N} (L_p \neq L_q)$$

  - Build a graph with 2 terminals ***s***,***t***

    Image pixels are nodes in the graph

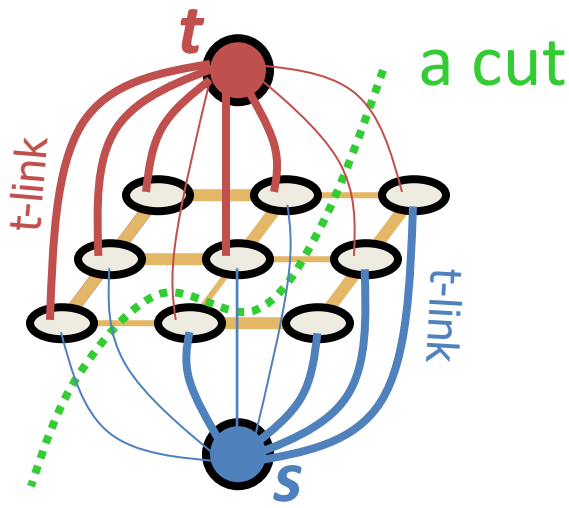
    nearby pixels(nodes) connected by an edge, which we call n-link

    Terminal ***s*** is identified with label 0, and connected by edge called t-link with every image pixel

    Terminal ***t*** is identified with label 1and connected by edge called t-link with every image pixel

    A cut separates ***t*** from ***s***

    Each pixel stays connected to either ***t*** or ***s*** (label 1 or 0)



Graph Cuts for Binary Energy Minimization

$$E(L) = \sum_{p} D_P(L_p) + \lambda \sum_{pq \in N} (L_p \neq L_q)$$

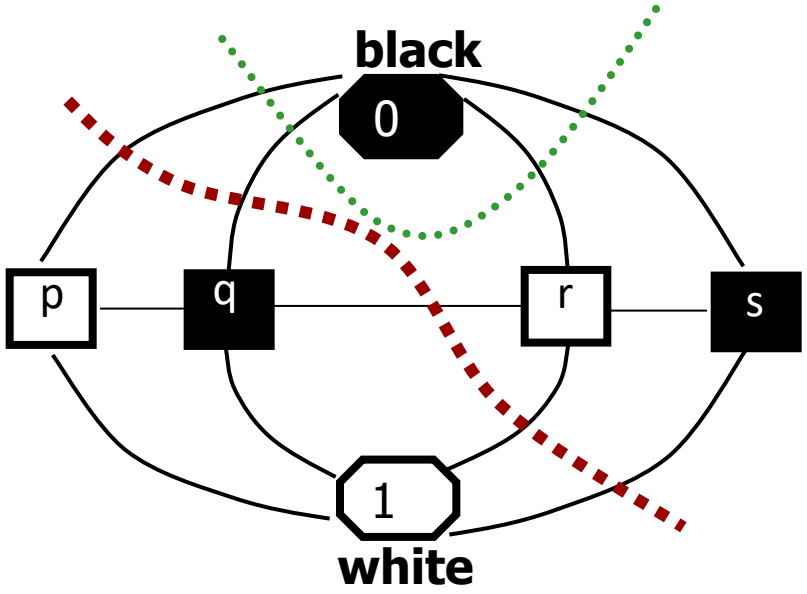
- Build a graph with 2 terminals ***s***,***t***
  - Image pixels are nodes in the graph
    - nearby pixels(nodes) connected by an edge, which called an n-link
  - Terminal ***s*** is identified with label 0, and connected by edge called t-link with every image pixel
  - Terminal ***t*** is identified with label 1and connected by edge called t-link with every image pixel
  - A cut separates ***t*** from ***s***
    - Each pixel stays connected to either ***t*** or ***s***
    - If pixel ***p*** stays connected to terminal ***s***, assign label 0 to ***p***
    - If pixel ***p*** stays connected to terminal ***t***, assign label 1 to ***p***
    - Cuts correspond to labelings, and with right edge weights cost is same
    - Thus the minimum cut gives the minimum energy labeling

Example

$$E(L) = \sum_{p} D_P(L_p) + \lambda \sum_{pq \in N} (L_p \neq L_q)$$

- For clarity, let's look at 1 dimensional example but everything works in 2 or higher dimensions. Suppose our image has 4 pixels:

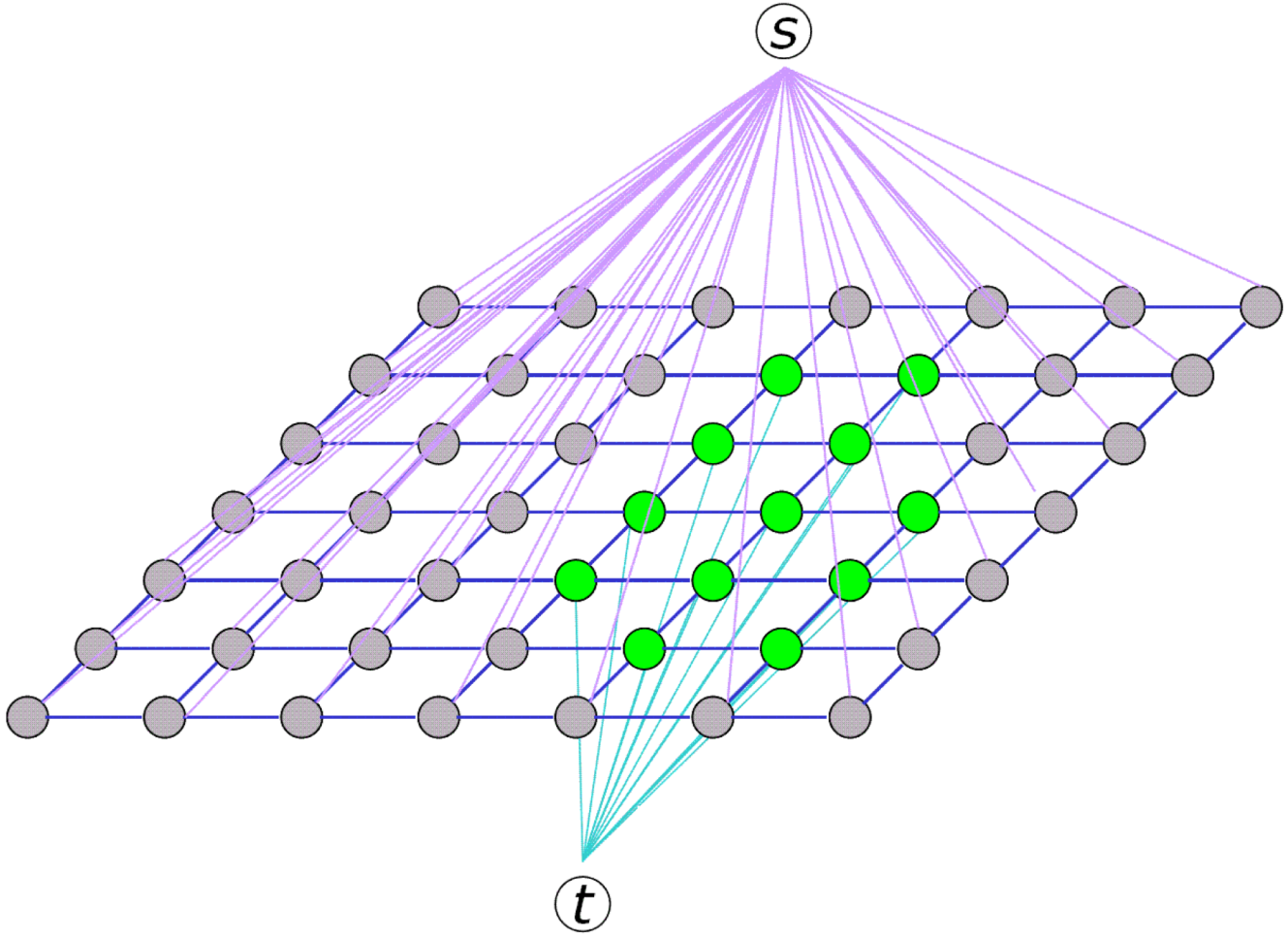
- We build a graph:



- The cut in red corresponds to labeling

- The cut in green corresponds to labeling

Graph construction



Beyond binary faxes

- All we really need is a cost function
  - Suppose that label 1 means “foreground” and label 0 means “background”
- How do we figure out if a pixel prefers to be in the foreground or background?
  - Predefined intensities: for instance, the foreground object tends to have intensities in the range 50-75
    - So if we observed an intensity in this range at p, $D_p(1)$ is small
  - This is **not** image thresholding!

Better intensity models

- We can compute the range of intensities dynamically rather than statically
  - Both for foreground and for background
- User marks some pixels as being foreground, and some as background
  - Compute a D_p based on this
  - For instance, $D_p(1)$ is small if p looks like pixels marked as being foreground
- Based on the resulting segmentation, mark additional pixels

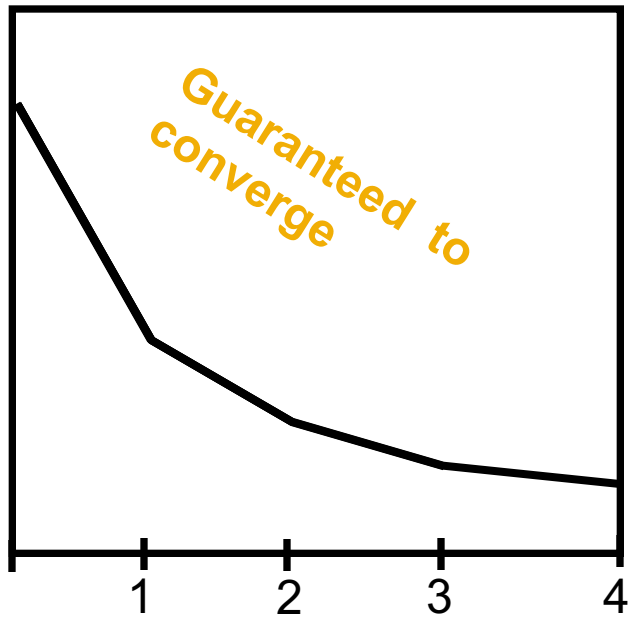
A serious implementation

- This basic segmentation algorithm is now in Microsoft office
  - SIGGRAPH paper on “GrabCut”
- Same basic idea, but simpler UI
  - Assume the background is outside and the foreground is inside a user-supplied box
  - $D_p(1)$ is small if p looks like the pixels inside the box, large if like the pixels outside
  - Create a new segmentation and use it to re-estimate foreground and background

GrabCut

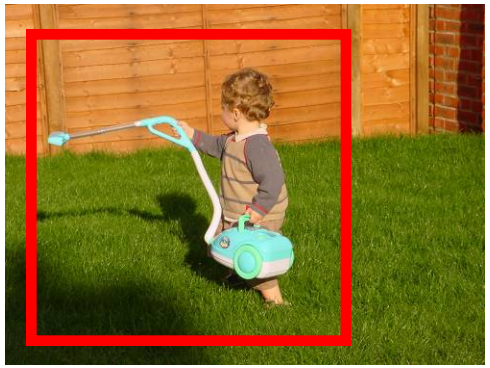
Iterated Graph Cuts

Result



See: http://www.youtube.com/watch?v=9jNB6fza0nA&feature=related

Moderately straightforward examples






… GrabCut completes automatically

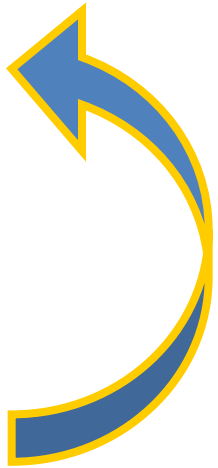
Can this be generalized?

- NP-hard for Potts model
- Two main approaches

1. Exact solution [Ishikawa 03]
    - Large graph, convex V (arbitrary D)
    - Not the considered the right prior for vision
2. Approximate solutions
    - Solve a binary labeling problem, repeatedly
    - Expansion move algorithm

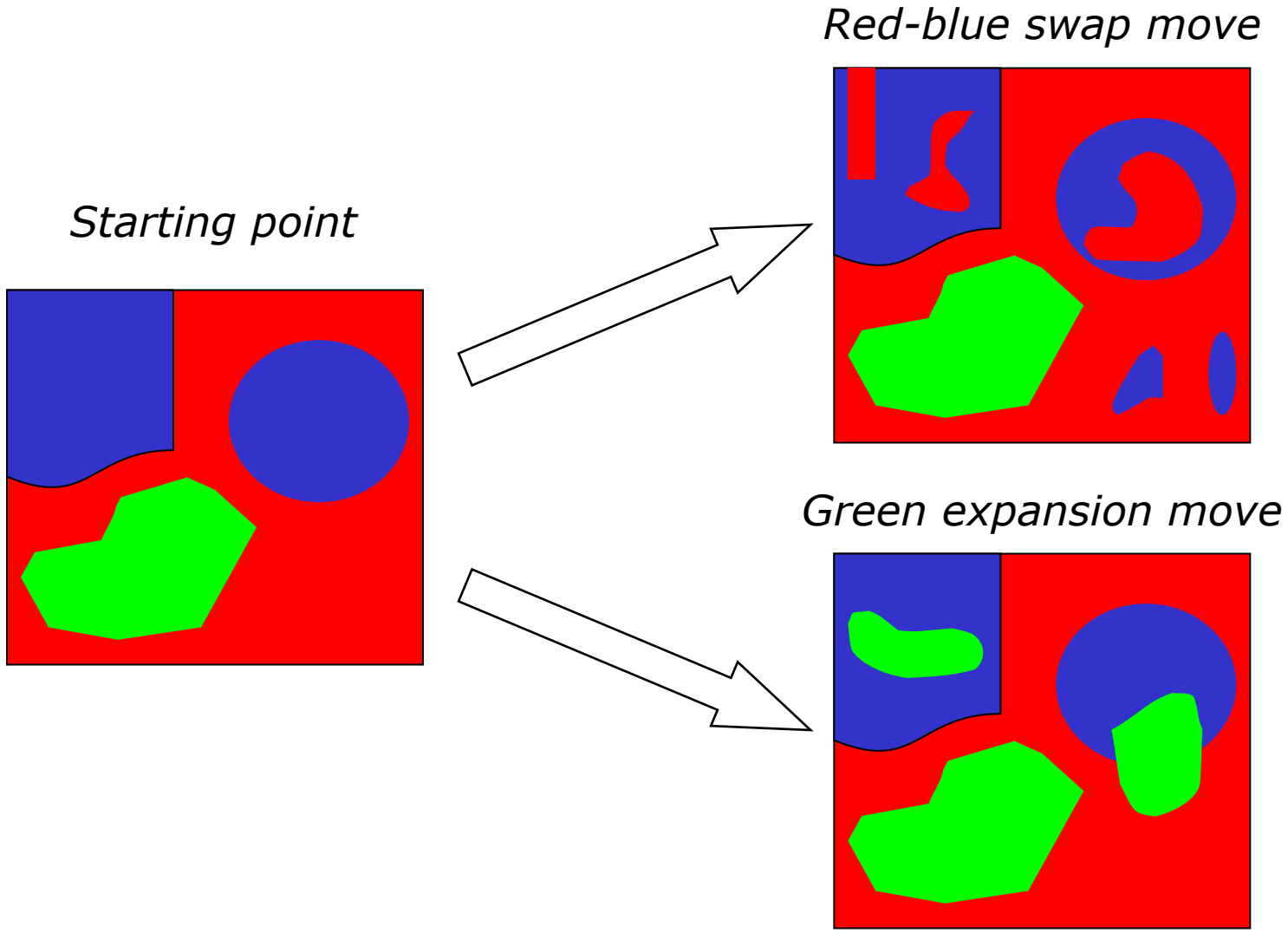
Coordinate descent (local improvement)

$$
\begin{aligned}
x_1 &:= \arg\min_x E(x, x_2, x_3, \ldots, x_n) \\
x_2 &:= \arg\min_x E(x_1, x, x_3, \ldots, x_n) \\
&\vdots \\
x_n &:= \arg\min_x E(x_1, x_2, x_3, \ldots, x)
\end{aligned}
$$



- Subproblem: pick a pixel, find the label that minimizes E, repeat
  - Minimize restricted version of E (line search)
  - Computes a local minimum

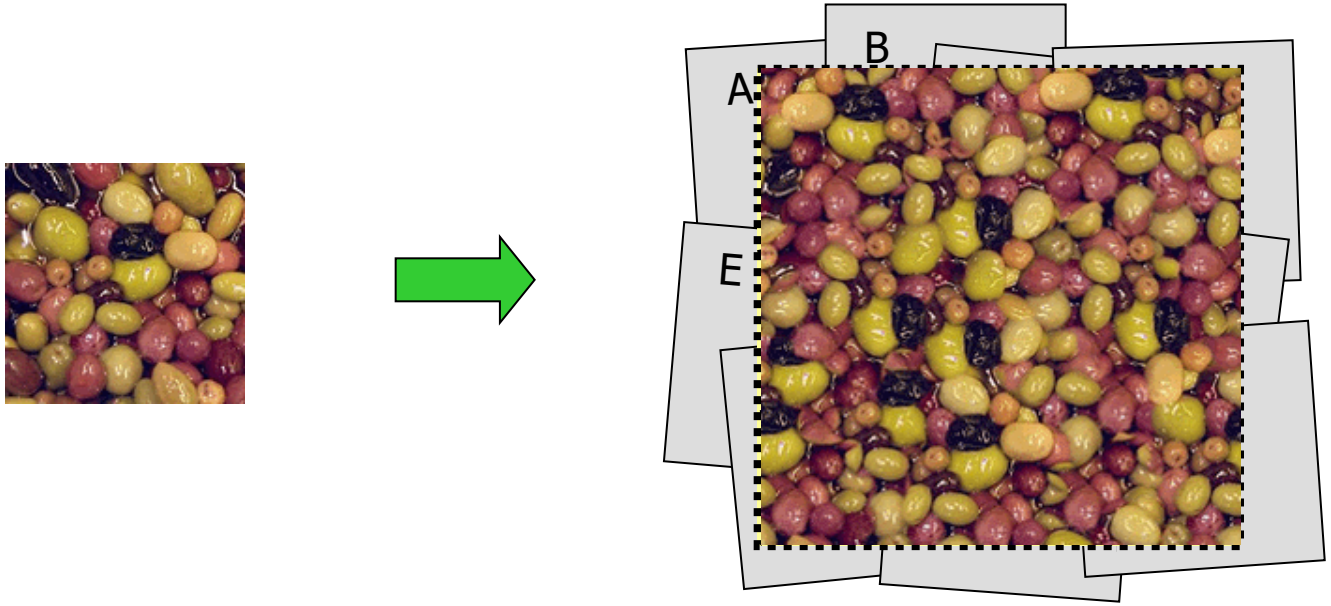
Swap moves and expansion moves



Important properties

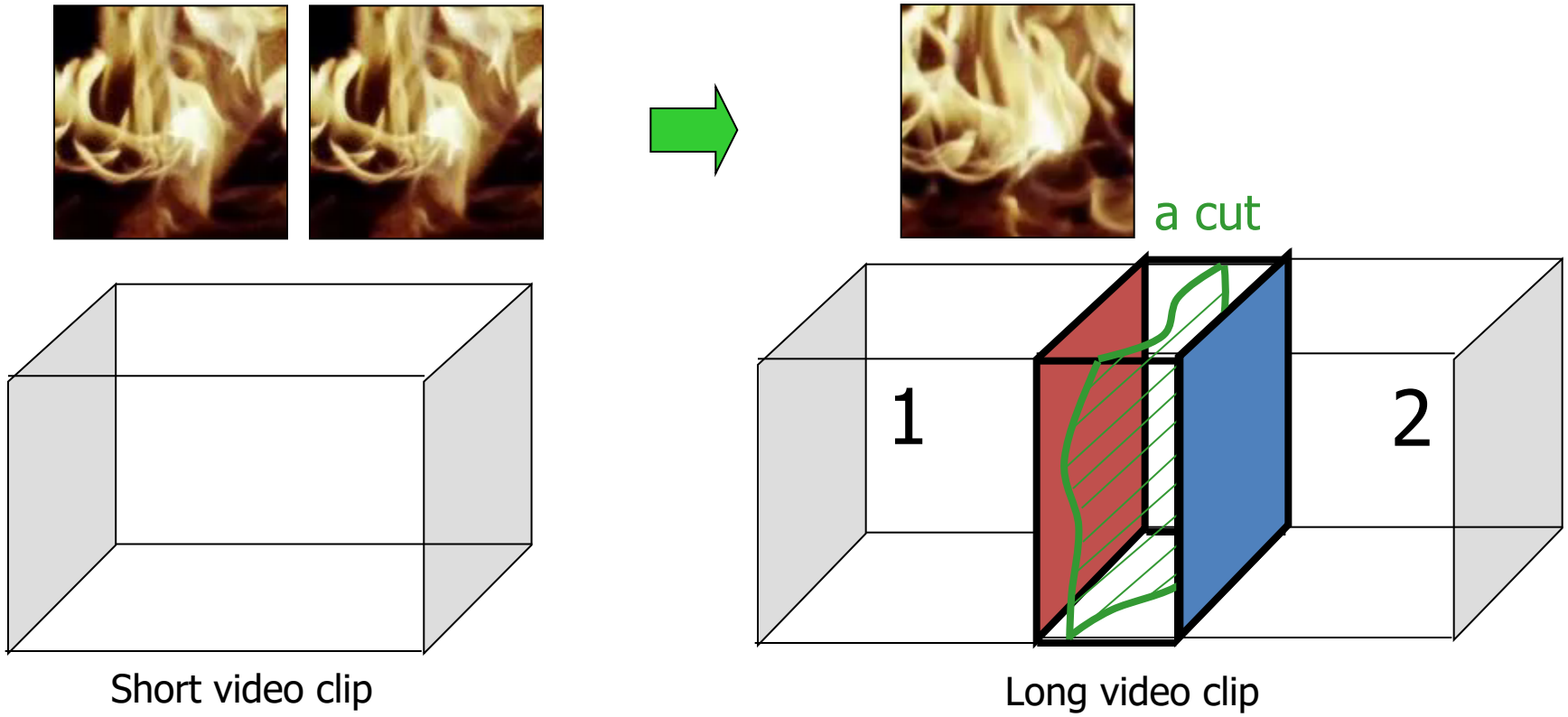
- Very efficient in practice
  - Lots of short paths, so roughly linear
- Construction is symmetric (0 vs 1)
- Specific to 2 labels
  - Min cut with >2 labels is NP-hard

Application: texture synthesis



“Graphcut textures” [Kwatra et al 03]

Graphcuts video textures



Another example

original short clip

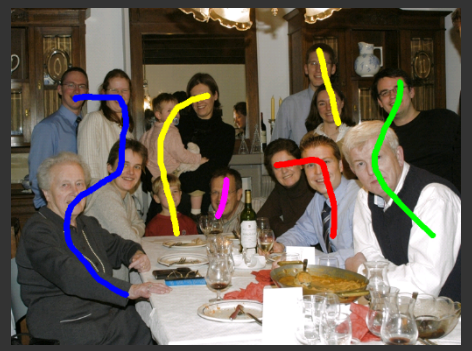
synthetic infinite texture

set of originals

perceived

Source images

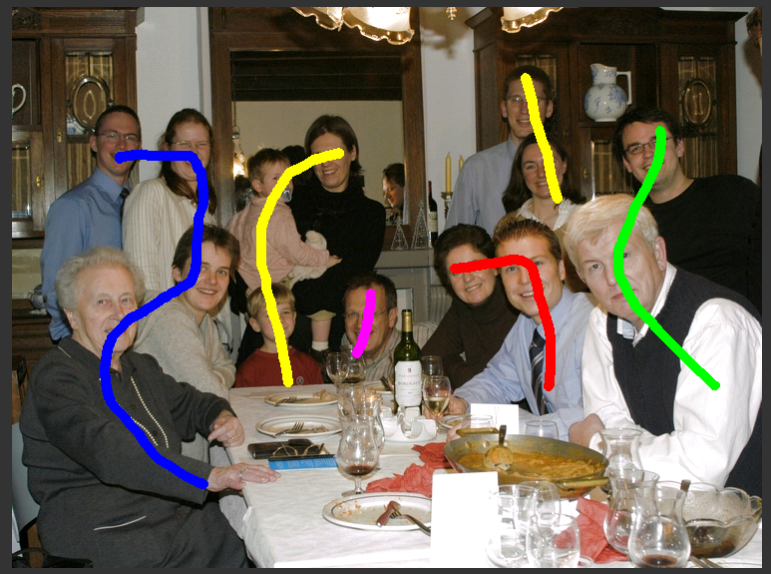
Brush strokes



Computed labeling

Composite

Brush strokes



Computed labeling

Image objective