Transcript MICCAI Educational challenge 2015: "Random walks" (M. Heinrich)

Slide 1: Hi I am Mattias Heinrich from the University of Lübeck and for this year's MICCAI Educational Challenge, I'd like to introduce you to the concept of random walks, and how it's used for medical image processing.

Slide 2: The application for which random walks are mainly known is interactive image segmentation — For a general overview of medical image segmentation, I suggest you view Ramesh's entry from the last year's challenge.
The random walker is in fact a generic optimisation algorithm that can be used for spatial regularisation of all kinds of images, parameter maps, and even arbitrary graphs. - I'll show some examples later. If we want to interactively segment this given coronal slice of a CT scan, we first have to provide some manual user scribbles for foreground and background. I'll use green for the kidneys and purple for the rest of the image. Based on this user input the random walk algorithm would give us the following output. I could now refine the scribbles to further improve the segmentation.
To understand how the algorithm works.
Slide 3: Let's look at a small section of the pixel grid and mark our seeded locations with green and purple as before. We now insert a 'random walker' which should start from the dark square. Our friend can walk in any of the four directions (up, right, down and left) and each of them has a certain transition (or edge) probability w. We want to estimate the probability u of the walker to first reach a foreground seed. We can evaluate a number of these random walks, as shown in the animation, and count the occurrences of the walker to reach either green or purple seeds. Once we've done enough walks we can convert this histogram into probabilities and assigned to the most likely label to the starting pixel, here the foreground/green.
This would be a very slow algorithm in practice, but fortunately we can instead solve a Dirichlet problem with certain boundary conditions using simple linear algebra. It minimises the discrepancy of label probabilities of neighbouring pixels given the edge weights represented by a Laplacian matrix L.

Slide 4: To get in intuition about this Laplacian matrix, let's look at a neighbourhood of the image grid. The squares have the grey value of the underlying pixels. We define edge weights based on squared intensity differences by this Gaussian formula which has a free parameter sigma in the denominator. This leads to similar label probabilities within a homogenous intensity areas and encourages different labels across intensity boundaries. In this case the neighbours above and to the right have the same intensity and thus a maximal weight of 1, the others have smaller weights and are more likely to take a different label.
In order to represent the whole image as a graph, we define a sparse matrix L which has the sum of incoming weights on its main diagonal and the weights for each of the four directions on side diagonals. Using MATLAB the matrix L could be build with the following commands. Now that we have set up the image dependent Laplacian matrix, we need to incorporate the seed labels — our boundary conditions.

Slide 5: Let's look at our image grid again: we have given a set of seeded and unseeded pixels x_s and x_u. We can notice that the edges, drawn as black lines, have different conditions depending on where in the image they are. For edges, that lie on the boundary between seeded and unseeded regions (X), the label probability can only be changed on one end, we'll call them B and mark them in red. Edges L_U within the unseeded region have flexible label probabilities on both ends and are drawn in blue. And finally the edges that are within seeded regions do not have any influence on the segmentation and are greyed out. By reordering the columns and rows, we can now write our Laplacian matrix as a combination of four block matrices, L_s, B, B transposed and L_u.

Slide 6: (X) The solution to the Dirichlet problem, which provides the label probabilities for the whole image, can be given by solving this equation (X). L_u times u(x_u) = - B transposed times u(x_s). We have specified B and L_u, but need to define the vector u of foreground probabilities. u is a column vector (X) with a length equalling the number of pixels in the image. For every unseeded pixel x_u the value is unknown (X) (and therefore on the LHS of the equation). But for seeded voxels we know that the probability of belonging to the foreground (X) is either 1 or 0 depending on the colour of the user input. (X) Solving the sparse system of equations for u(x_s) fills in the missing values.

Slide 7: Here are the foreground probabilities for our CT image. For a binary segmentation problem, the label probabilities of the background are simply 1 minus the foreground probabilities. So we can obtain a segmentation by thresholding u at 0.5. Using a higher value for the parameter sigma of the edge weights yields a smoother contour. Extending the random walker to multi-organ segmentation is trivial: we simply define a different vector u for each label, and then set the foreground label to 1 for its seeds. We then solve the system of linear equations for each label. The label for which the probability is maximal is chosen for each pixel.

Slide 8: As mentioned early, this algorithm can also be used for regularising any input data. To keep it simple, we assume no user seeds are provided and we are using the full matrix L. Let’s assume we have a pixel-wise prior model Lambda for the left kidney. This could be based on an intensity-model, atlas-based registration or a machine learning algorithms. Here I used a random forest. If we make a segmentation based on this information alone the outcome would be quite rough. But we can now use the random walk algorithm to regularise the spatial map. For this we introduce a scalar weighting parameter gamma and obtain the following equation. Note that the edge matrix L is still derived from the original image intensities.
Solving for u gives us the regularised foreground probabilities and a much nicer segmentation.

The same idea can be used to denoise our input image. The prior Lambda is know equal to the image intensities and the solution for u gives us the denoised image.

This precludes my lecture
Slide 9: and I’ll just listed some more references for further reading, for example how to speed-up computation of random walks in 3D, for image registration based on random-walks, and related mathematical concepts.
The MATLAB code for the examples and the transcript of this talk can be found at mpheinrich.de/mec.html