Finding Optimal Minimum Energy Pathways using Graphical Models

Applying the max-product algorithm to find the minimum energy pathway and avoid local minima.

Adam Carruthers

Abstract

We explore the issue of local minima when finding minimum energy pathways in energy landscapes. We then apply the max-product message passing algorithm to efficiently explore the landscape and hence find a better route. Finally we evaluate the performance of this approach in different scenarios and with respect to different parameters.

Chapter 1

Introduction

Energy landscapes can describe an incredible array of systems, particularly biological and chemical processes. It is therefore vital to understand the stable states in this system, and the transitions between them. For example, if you can find the transition pathway between two stable states in a chemical reaction that has the lowest possible maximum energy then it is possible to deduce information about reaction speed. Such a pathway is called the minimum energy pathway.

A significant amount of research has been devoted to finding these minimum energy pathways. A popular method is called Nudged Elastic Band (NEB), which lets the path be a string of points. It starts with a random guess (normally a straight line between two stable points), and then lets the points fall down the slope of the energy landscape (given by the gradient of the energy function). However it then forces the points to remain equidistant by exerting a spring force on them[1].

This algorithm is a form of gradient descent. As with any form of gradient descent the algorithm can get stuck in local minima, as in Figure 1.1. When the gradient descent is run the path gets stuck in a high energy valley, when going around gives a lower energy result.

In this paper we propose an algorithm that can efficiently explore the landscape and the possible paths. This algorithm is therefore able to find very promising initial path guesses that outperform naive NEB, and it even has the ability to quickly find a selection of good paths that go through different areas of the energy landscape. We provide an analysis of how this algorithm performs in different situations - including the time it takes to run as well as the quality of the outputted paths. We will refer to this algorithm as the Max-Product Message Passing (MPMP) algorithm. This is because the already established MPMP algorithm is the main part of the program that does the calculations to work out the best paths. However, in some charts the process may be referred to as Markov Random Field (MRF), which is closely related to MPMP.

1.1 Basic MRF algorithm description

The algorithm effectively defines the path by a number of points. It works along the path step by step, testing a number of positions for each point and working out the how good that possible position is. When working out the quality of a position it incorporates the quality of all the points leading up to that position using an algorithm called max-produce message passing. If effectively scans the points that lead up to each position and picks the preceeding path that maximises its quality.

[1]This is a slight simplification, as moderations are made to stop the algorithm cutting corners. See the paper cited for implementation details.

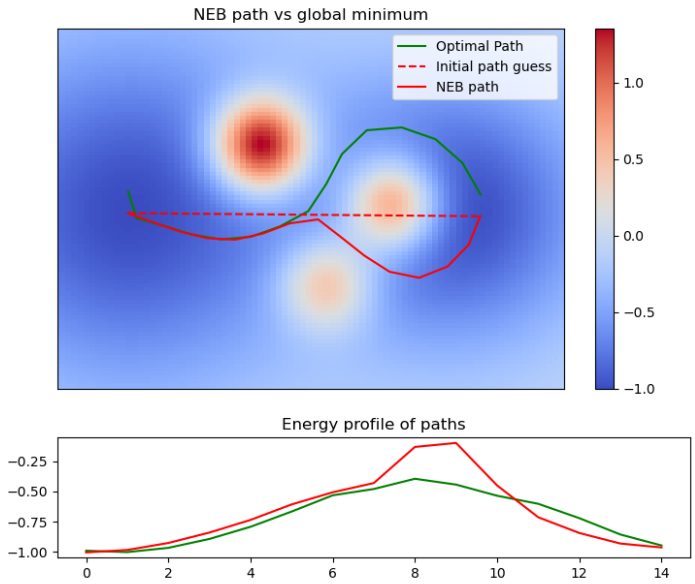


Figure 1.1: The NEB generated and global minimum energy path between two minima (above) with the energy profile of those paths (below).

Once we have found the path we then smooth out the generated path with the Nudged Elastic Band method, as if our suggested path is close to the minimum path then NEB should converge to it. This is important since the algorithm must perform a very coarse scan of the landscape for time complexity reasons. The algorithm scales linearly by the number of positions being scanned, multiplied by the number of positions on average that one position is connected to. This means that particularly in high dimensions, with a lot of space that you need to explore, the produced path cannot be very fine grained.

This method leaves us with a few important questions, which we will consider in the paper:

- How do we decide the possible positions and transitions that our path could go through? (Chapter 2)
- How do we define the quality of a point or a transition? (Chapter 2)
- How do we make it all fit together? (Chapter 2)
- How long does this algorithm take to run? How does this compare to NEB?
- What are the other possible algorithms that might be able to do this?
- Are the outputs of this algorithm better than naive NEB?

Chapter 2

Algorithm structure

This chapter will walk through the algorithm step by step, along with why the different parts of the algorithm work well for this problem.

2.1 Markov Random Fields

To understand the Max-Product Message Passing algorithm you must first understand Markov Random Fields. A Markov Random Field is a graph based probability distribution. Each node in the graph G can take a certain set of values. We will denote the random variable describing each node's value Y_i and the random vector denoting all node's values $\mathbf{Y}$. If the nodes take a particular value we will call it an assignment, a node could take assignment y_i and all the nodes could take assignment $\mathbf{y}$. Consider the graph below in Figure 2.1.

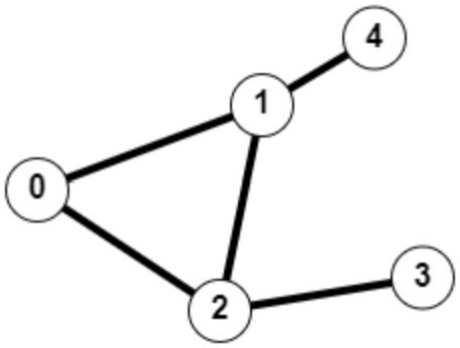


Figure 2.1: A simple graph

In a MRF, the probability of a particular assignment is given by a product over all the cliques of the graph - subsets that are fully connected. Each clique c has a potential function ϕ_c that takes as an argument the assignment to the clique, which we will call $\mathbf{y}_c$. The equation for the probability is

$$\mathbb{P}(\mathbf{Y} = \mathbf{y}) = \frac{1}{V} \prod_{c \in cliques(G)} \phi_c(\mathbf{y}_c) \tag{2.1}$$

$$[\text{in this case}] \ = \frac{1}{V} \phi_{\{0,1,2\}}(y_0, y_1, y_2) \phi_{\{1,4\}}(y_1, y_4) \phi_{\{2,3\}}(y_2, y_3), \tag{2.2}$$

where V is a normalisation constant.

As a warning, the potential functions can also be referred to as the weight or quality functions.

In most (but not all) cases, we can factorise this into a factor graph, where factors represent the clique functions and the nodes they are connected to. We can now refer to variable nodes and factor nodes, and the probability of an assignment is the product of the factor node potential functions. The probability of an assignment is now a product of the factors, and the relationships in the graph become a lot easier to see. We can see how Figure 2.1 factorises in Figure 2.2.

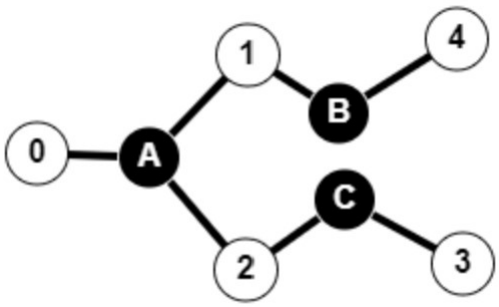


Figure 2.2: Factorisation of Figure 2.1 where nodes with a white background are variables and nodes with a black background are factors.

2.2 Representing a path as a factor graph

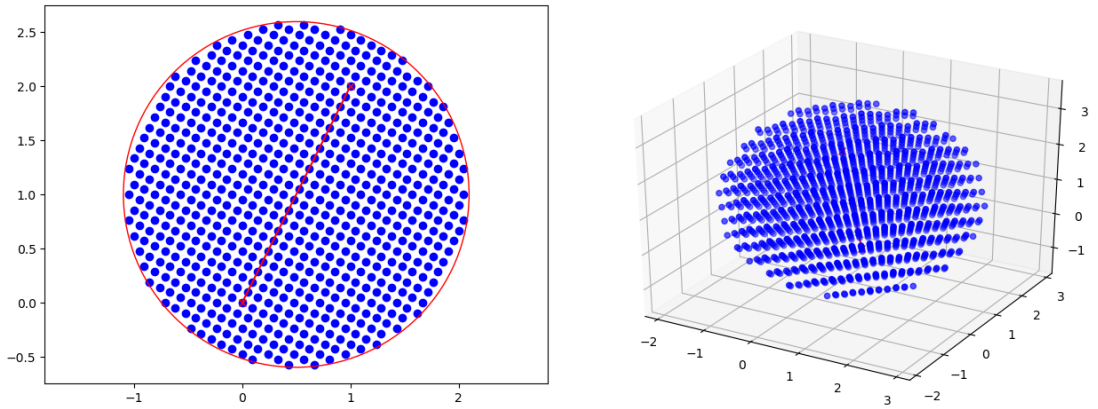
2.2.1 Variables and possible positions

Now is a good time to discuss then how to represent a path through an energy landscape as a factor graph, and how to define those potential functions to give a higher probability to good paths. It makes sense for the variable nodes to represent the points along the path, and then for the possible values they can take to represent the positions they can be in the space. Continuous MRFs do exist but they are not well suited to this problem, hence we also need to pick for each variable (variable node) a finite number of discrete positions it can take.

There are plausibly many possible ways of doing this, the approach that we use seems a reasonable way to create a set of points in a manner that can scale to arbitrary D dimensions. You need to define a set of points that most paths would travel through between the two minima. The algorithm as currently implemented creates a hypersphere of points around the midpoint between the two minima, with the sphere's diameter being slightly wider than the gap between the two points. In order to do this it creates a hypercube grid of points wider than the sphere will be, with S points along each edge. Increasing S will create a more detailed path, but it will also increase complexity, especially in high dimensions where the number of points being considered will be approximately S^D.

The points are also rotated so that the points can be split into slices orthogonal to the vector that connects the two minima. This is achieved by creating the hypercube in a non-standard basis where the first basis vector is in the direction of the minima line, and then transforming it back into the standard basis. You can see the exact implementation in the code, but the results are displayed in Figure 2.3 for $D = 2$ and $D = 3$.

We should not, however, let every variable along the path take every position in the sphere. It would be ridiculous for the path to jump to one end of the sphere and then jump back, and furthermore allowing each variable more possible values increases computational complexity. Therefore we restrict the variables to only take values in a slice of the sphere, with one slice for each variable representing one section of the path, as in Figure 2.4.



(a) Sampling in 2D

(b) Sampling in 3D

Figure 2.3: How the sampling algorithm performs in 2D and 3D. Red crosses connected by a red line denote the two minima that the path will connect, but it is quite difficult to spot in the sphere!

Already at this point the algorithm has a lot of customisable parameters. You can decide how wide you want your slices to be, how many variables you want spanning the gap between the two minima, how many point positions you want spanning the gap between the two minima, how much the slices overlap with each other (slices can overlap but it will increase computational complexity a little bit), how much wider to make the sphere of points than the gap between the two minima. The code can even make the sphere a sphereoid, stretched in certain directions so that you can explore more of the energy landscape. There is no easy way to evaluate how much effect changing these parameters has on path quality, although the effect on time complexity is more obvious.

In general, for the code we use later, the number of variables was 5, the number of positions spanning the gap between the two minima was 5 and the width of a layer was 1, no overlap, spherical shape. This level of parameters produced decent parameters and ran quickly enough to allow a lot of samples of even higher dimensional problems. In addition it produced acceptable results when checked manually in lower dimensions (the same was not true of reducing the number of variables or points spanning the gap to 4, which seems to be too coarse).

We also add a variable at the start and end of the paths to represent the start and end positions, however we only allow these nodes to be at one position, the start and end position respectively. This tethers our paths at the ends.

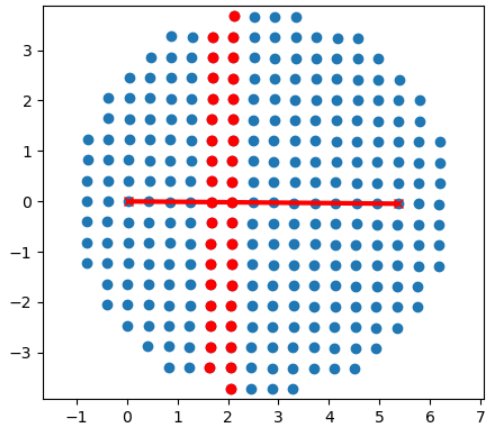


Figure 2.4: The points generated for a sphere (blue) with the points that are in one variable's slice highlighted in red. In addition the two minima are marked with a red x and connected by a red line. As you can see the slice is perpendicular to the minima direction.

2.2.2 Factors and quality functions

Next we need to choose how we arrange our factors and with it how we choose our potential functions to give the highest weight to good paths. For the factor arrangement it may seem optimal to just connect a factor to every single variable, and from there define a clever function that gives high scores to very good paths by considering the whole path. However we will see later that the MPMP algorithm requires that we reduce as far as possible how many variables each factor node is connected to. It also requires that our factor graph doesn't have any cycles in it, making it a factor tree. Therefore we connect each variable node to its neighbor, creating a chain from the start of the path to the end, with no other connections. You can see an example of a path factor tree generated by the MPMP algorithm in Figure 2.5.

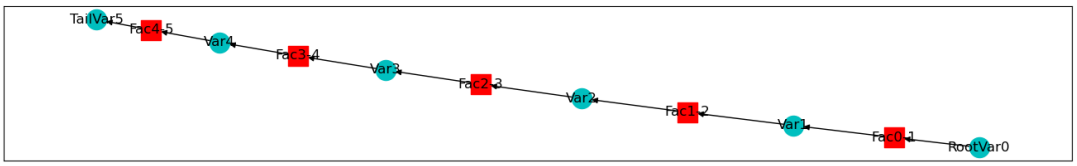


Figure 2.5: A computer visualisation of the factor tree generated for the MPMP algorithm, with variables as circles and factors as squares.

That then leaves the question of how to define the factor potentials or qualities. The primary aim of the path is to reduce the maximum energy level it goes through, however the path must also go through the minimum possible energy path to get to that max energy point. Furthermore the path must not be excessively long.

As you will recall, the quality of the overall path is the product of the quality of each factor, and each factor can only take into account the nodes it is connected to. We use a linear combination of different features to create a score for the transition. Features measure one aspect of the quality of the transition, for example there is one feature that penalises jumps that are longer so that paths are reluctant to take large jumps. Qualities must be at least zero however, so to allow score to be any value, we take the exponent of

the score.

Let the start point of a feature be $\mathbf{x}_0$ and the end point $\mathbf{x}_1$, with start meaning the point in the path is closer to the beginning of the path, and moving on to the end point which is further down. Let $a_0, a_1, \ldots, a_{n_f} \in \mathbb{R}$ be the coefficients for the features $f_0, f_1, \ldots, f_{n_f} \in \mathbb{R}^D \times \mathbb{R}^D \mapsto \mathbb{R}$. In equation form the potential for factor F is

$$\phi_F(\mathbf{x}_0, \mathbf{x}_1) = \exp(a_0 f_0(\mathbf{x}_0, \mathbf{x}_1) + a_1 f_1(\mathbf{x}_0, \mathbf{x}_1) + \ldots + a_{n_f} f_{n_f}(\mathbf{x}_0, \mathbf{x}_1)). \tag{2.3}$$

Let $E(\mathbf{x})$ be the energy strength at point $\mathbf{x}$. Let $\mathbf{x}_{0.5} = (\mathbf{x}_0 + \mathbf{x}_1)/2$ be the midpoint of the two points. The features we tested for the score were (with their names in (brackets))

- (Length) minus the length of the transition
  - ☐ $-|\mathbf{x}_0 - \mathbf{x}_1|$
  - ☐ This means that longer jumps are given lower scores.
  - ☐ In general the length is scaled by how far apart the points in the grid are so that the algorithm works well in different situations and settings.
  - ☐ Turned up to high this would make the algorithm "cut corners", and go in as straight a line as possible between the minima, so we tended to keep this feature near zero anyway.
- (Length Cutoff) a score of $-\infty$ if the path's length is over a chosen cutoff
  - ☐ This prevents truly crazy jumps, as that jump will have quality zero.
  - ☐ In addition, you can code it so it reduces the number of calculations the algorithm has to do. By eliminating far away points the algorithm has fewer combinations of points to consider, saving time.
- (Strength) minus the average energy of the midpoint and endpoint
  - ☐ $-(E(\mathbf{x}_{0.5}) + E(\mathbf{x}_1))$
  - ☐ This makes the path favor going through lower energies.
  - ☐ It has the negative side effect of making the path stay in areas of low energy even if it then forces them to go through a relatively high energy maximum path energy later.
  - ☐ The midpoint energy is calculated so that if the MPMP algorithm is run with coarse settings (not many points) it can't "jump" over a high energy area without paying a cost.
  - ☐ The start point is not used because then the end points of the path would be double counted in different factors.
- (Strength Diff) minus the energy increase from the start point to the highest of the midpoint or the endpoint, 0 if the energy is decreasing from the start point.
  - ☐ $-(\max(E(\mathbf{x}_1), E(\mathbf{x}_{0.5})) - E(\mathbf{x}_0))$ if the energy is increasing from $\mathbf{x}_0$ else 0, aka this feature is at most 0.
  - ☐ Gives a negative score to a big energy increase from the start node, so tries to reduce the maximum energy that the path goes through.
  - ☐ When you take the product of all the factor qualities it will be the exponent of the sum of all the factor scores. If the path goes uniformly up and then uniformly down, then the sum of the strength diff scores will be the maximum point the path goes through, minus the initial starting energy.

- ☐ Hence, this feature minimises the maximum strength the path goes through.
- ☐ This feature only takes into account the highest point the path goes through, therefore relying on this feature alone can lead to paths that do have a low maximum, but take a terrible path to get there.
- ☐ In addition, this feature might favor skirting along the mountainside of a valley when it should be falling down into the centre of the valley, if it prevents the path from dipping down and going back up again. This is because the "going back up again" part will be penalised by this feature.

- (Gradient) the length of the grad vector at the midpoint, after making the gradient vector orthogonal to the direction of the path.
  - ☐ This feature was borrowed from the Nudged Elastic Band method. In Nudged Elastic Band they make the path fall down the slope into the valleys by pushing the points of the path down the gradient of the Energy scalar field. However, they only let the points be pushed in the direction orthogonal to the path, so that the points could stay equidistant to each other. When you're in a valley moving between the minima on the minimum energy pathway your path will have gradient zero in every direction, because otherwise you would be able to move to get a lower path. Every direction that is, except the direction of the path, because it may need to be moving up a gradient to be getting to a goal in the direction of the path.
  - ☐ This feature is meant to incentivise a small orthogonal gradient, meant to favor paths at the bottom of the valley.
  - ☐ This feature did not work well, because in addition to favoring the bottom of the valley the feature also favors the ridge of a hill, since using the first derivative alone cannot distinguish between a minima and a maxima. See Figure 2.6 for an example.

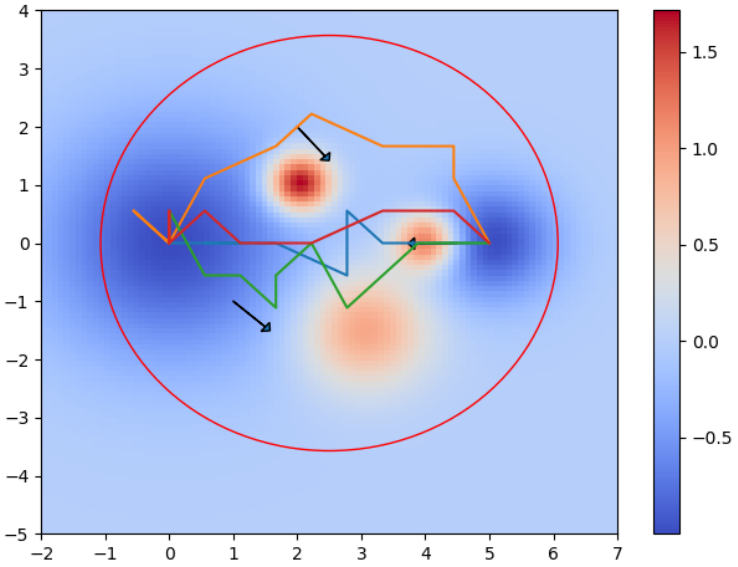


Figure 2.6: A selection of MPMP generated paths with the gradient feature on (no second order). You can see that the green and blue paths head directly for the maxima.

- (Second Order) the second order derivative in the direction of the orthogonal component of the gradient.

- ☐ This feature was added to stop the issue of the first order derivative favoring maxima, by favoring a second order derivative that was high (a minima).
- ☐ It turned out to be a non-ideal feature because it would draw the path to strange areas of the energy landscape that happened to have high second derivatives.
- ☐ It also required a lot of extra computation, which made the algorithm a lot slower.

Overall, out of these features, we only used Length Cutoff, Strength and Strength Diff. For the reasons mentioned above, the gradient based and length features were ineffective. We discovered the strength and strength diff feature needed to be scaled so that in environments with dramatically different energy levels they would perform similarly. We would take a scan of the energy along the straight line between the two minima and rescale all the energies such that the max seen on that line was an energy of 1 and the minimum was 0 $E_{\text{rescaled}} = (E - E_{\min})/(E_{\max} - E_{\min})$.

We are not yet done with deciding quality features, as we need to decide the relative weights of the features in the linear combination that makes up the score (the $a_0, a_1, \ldots$ from (2.3)). We tried out the algorithm on 2D and 3D environments where the best paths were obvious, and deduced good settings from this experimentation. The length feature should be near zero, but keep the length cutoff on. Strength should have tuning coefficient 1 and Strength Diff tuning coefficient 1.5.

The best method of evaluating the paths was to create a function that would take in the paths produced by the algorithm and print the factor by factor and overall scores for each feature. This made it very obvious which features were most influencing the path. It was ideal to have the Strength Diff feature about 25% larger than the Strength feature. This meant the algorithm would first prioritize reducing the path's maximum point, but could also secondarily focus on reducing the path's energy in how it got to that minimised max (minimax) point.

Now we have a way to define paths as an MRF and evaluate a path's quality, we can now look at the algorithm to maximise the quality.

2.3 Max-Product Message Passing

Defining a set of nodes that can take certain values, along with a quality function, is a good first step. However, how do you choose the best one? It is clearly impractical to test out every possible combination of values and work out the quality as this method would take an incredibly long, exponentially increasing amount of time.