# 1 The Image Restoration Problem Reviewed

We have some true image $I_t$ in the world, and but due to random noise $r$ we observe an image $I$ that is (with any luck) similar but not identical to $I_t$. So, at every pixel $p$:

$$I_t(p) + r(p) = I(p)$$

The image restoration problem is then that of trying to regenerate something as close as possible to $I_t$ from $I$.

# 2 The Metropolis Algorithm

## 2.1 Algorithm Outline

Recall that the energy function of some mapping $f : p \rightarrow L$ from pixels to labels consists of a data term and a smoothness term:

$$E(f) = \sum_p c(p, f(p)) \ + \ \lambda \sum_{(p,q \in N)} d(f(p), f(q))$$

The Metropolis algorithm takes $f$ and randomly generates $f'$ by sampling (for example, by just changing the label of one pixel). We look at the value of $\Delta E = E(f') - E(f)$:

- If it is negative ($f'$ has lower energy than $f$), we accept $f'$, set $f = f'$ and repeat.
- Otherwise set $f = f'$ with probability equal[1] to $e^{-\frac{\Delta E}{T}}$.

## 2.2 Markov Chains

We can think of a Markov chain as describing the rules of a game of chance, or as a randomized algorithm.

We saw the following in the last lecture:

**Theorem 1 (Ergodicity Theorem)** *for all Markov chains K*

$$\exists \pi^* \ . \ \pi^* K = \pi^*$$

*and*

$$\forall \pi \ . \ \lim_{n \rightarrow \infty} \pi K^n = \pi^*.$$

[1]Note this differs from last lecture, where we only stated the probability was proportional to this term.

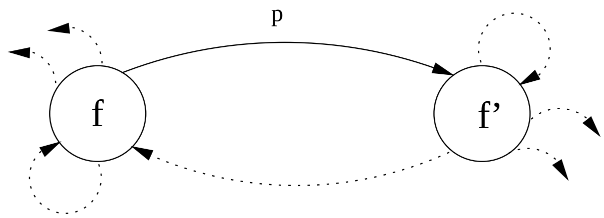


Figure 1: From every labeling $f$ there is some probability $p$ of moving to a neighboring labeling $f'$, other probabilities for moving to other neighboring labelings and some probability of remaining at the same labeling for another step of the algorithm. Note that the ergodicity theorem requires these self loops on all nodes in the Markov chain.

It is obvious that the Metropolis algorithm is a Markov chain — the nodes of the chain correspond to labelings and directed edges from one node to another are labeled with the probability with which the algorithm will move from one labeling to the other.

With some tweaking we can ensure that the chain is uniform, with all nodes having the same out-degree. In this case we can show a direct relationship between the Gibbs distribution and the fixed point $\pi^*$.

**Theorem 2** *For a uniform Markov chain,* $P(s) = Ce^{\frac{-E}{T}}$ *is* $\pi^*$*.*

**Proof** Consider 2 states $R, S$, without loss of generality assume $E(S) \leq E(R)$. We use the frequency interpretation of probability — if $P(H) = p$ then $H$ will occur $Np$ times in $N$ experiments. Now if one step of the Metropolis algorithm is run $N$ (large) times in parallel, the outcome each time will have some associated energy, and it suffices to show that the if the number of outcomes with energy $E$ IS $NCe^{\frac{-E}{T}}$ then taking a step does not change this.

Observe that for any two states $R, S$ with a move between them, the number of outcomes $R \rightarrow S$ is equal to $S \rightarrow R$, in other words the distribution is stationary. Figure 2 illustrates the relationship between $S$ and $R$.

The transition $S \rightarrow R$ is "uphill", so unlikely but still possible with probability depending on the probability of generating $R$ as a successor to $S$, the energy difference between the two labelings and the temperature parameter $T$ for the algorithm. The transition $R \rightarrow S$ is "downhill" and so it's probability depends only on the probability of generating $S$ as a successor to $R$. We use $\nu_{RS}$ to designate the number of possible outcomes for which we started in $R$ and moved to $S$, then

$$\nu_{RS} = NCe^{\frac{-E(R)}{T}} \times P(\text{generate a move to } S)$$

and

$$\nu_{SR} = NCe^{\frac{-E(S)}{T}} \times P(\text{generate a move to } R) \times e^{\frac{-\Delta E}{T}}$$

Because the chain is uniform, the all moves have equal probability and so the probabilities of generating terms cancel out, so we are left with:

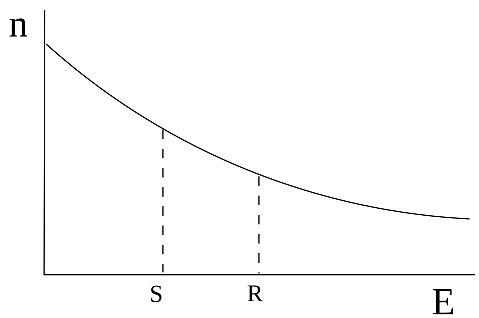


Figure 2: $S$ has lower energy than $R$ and thus is a more probable outcome overall.

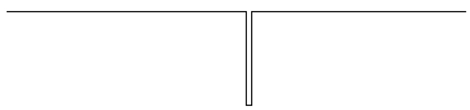

Figure 3: A high-dimensional space with a single low-energy point (well). In a low-temperature system, the Gibbs distribution is such that we are almost always at the bottom of the well in the limit, but the system takes a very long time to get to this equilibrium.

$$
\begin{aligned}
\nu_{SR} &= NCe^{\frac{-E(S)}{T}} \times P(\text{move}) \times e^{\frac{-\Delta E}{T}} \\
&= NCe^{\frac{-E(S)}{T}} \times \times P(\text{move}) \times e^{\frac{E(S)-E(R)}{T}} \\
&= NCe^{\frac{-E(R)}{T}} \times P(\text{move}) \\
&= \nu_{RS}
\end{aligned}
$$

## 2.3 Metropolis Algorithm Performance

Observe that setting $T$ very low gives us an algorithm that returns the true minimum energy labeling with a very high probability, but this take a very long time (the convergence to the equilibrium point is at infinity). It is easy to generate examples where this poor behavior is observed — for example, a high-dimensional space with very low energy at only one place as illustrated in Figure 3.

We refer to the behavior of Markov chains as they approach a stationary distribution as **mixing**. There are very few theorems describing this behavior.

**Fact 1 (Experimental)**

- *At high values for $T$ the stationary distribution is essentially flat (as $T \to \infty$ we approach the uniform distribution) but convergence is fast.*

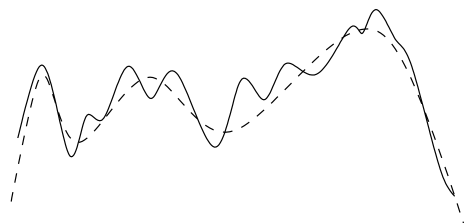

Figure 4: An energy function $E$ is smoothed into an energy function $E'$ with fewer local minima.

- *At low values for $T$ we are almost always at the minimum energy point in the stationary distribution, but convergence is slow.*

# 3 Simulated Annealing (1983)

## 3.1 The Simulated Annealing Algorithm

Simulated annealing is simply the Metropolis algorithm with decreasing $T$. At stage $s$ we run the Metropolis algorithm $M_s$ with temperature $T_s$ until it doesn't seem to be making much progress, then use the end point as the starting point for $M_{s+1}$ with some lower temperature $T_{s+1}$.

Simulated annealing is one of a class of methods called **continuation methods** where a non-convex (and thus hard) problem is approximated by a series of convex problems. To see how simulated annealing falls into this class, we suppose that we can smooth the energy function $E$ with local averaging as shown in Figure 4. We can then find the minimum for the smoothed function, using a parameter $\sigma$ to control the extent of the smoothing.

## 3.2 Performance of Simulated Annealing

We refer to the mechanism for determining for fast the temperature should be reduced as the **schedule**. If the schedule is slow enough, it can be proved that simulated annealing gives the global minimum. The problem is that optimal schedules are completely worthless, since for an optimal schedule we require

$$T_m \geq \frac{nc}{\log m}$$

where $n$ is the number of pixels and $c$ is some large constant. $T_m$ is thus a large number that does not decrease very quickly as $m$ increases since increasing $m$ by a constant will barely increase $\log m$ at all.

As with the low-temperature Metropolis algorithm, this is probably worse than simply performing an exhaustive search over the problem space.

One might reasonably ask why annealing is liked at all. It is true that annealing has a nice physical metaphor. The term itself comes from the process by which a liquid cools and forms a crystalline structure, where the lowest-energy structure (a regular crystal) is achieved through very gradual cooling.

The paper in which the Metropolis algorithm and simulated annealing were originally introduced[2] made very successful use of it to solve the Traveling Salesperson problem. It is worth noting, however, that the moves made available in the algorithm used were very cleverly designed, and it is quite possible that similarly impressive results might have been obtained with exhaustive search. Whenever particularly good performance is claimed for some method it is generally worth examining whether this might be due to particularly clever problem formulation.

In any case simulated annealing is not a very good algorithm for computer vision, where there are lots of pixels, so lots of dimensions (typically hundreds of thousands) and slow sampling, along with energy functions with many local minima.

# 4 Different Energy Functions

## 4.1 Energy Function Variations

We can play with the general energy function structure

$$E(f) = \sum_p c(p, f(p)) + \lambda \sum_{(p,q)\in N} d(f(p), f(q))$$

in various ways. The cost function $c$ can take into account just a single pixel $p$ or a window around the pixel as in the smoothing algorithm proposed by Birchfield and Tomasi. The difference function $d$ can be designed so that labeling similar neighbors differently is very expensive.

## 4.2 The Difference Function $d$

The simplest choice for $d$ that gives good results in the real world is

$$d(\alpha, \beta) = T[\alpha \neq \beta]$$

where $T$ is a truth operator, so $T[b] = \begin{cases} 0 & \text{if } b \text{ false} \\ 1 & \text{if } b \text{ true} \end{cases}$

This is called the Potts model and has the same energy profile as the energy function we saw in Figure 3 — constant everywhere except for a single well of low energy.

Suppose now that we have a spatial coherence problem where each pixel has one two possible labels, let $L = \{0, 1\}$ (either a pixel moves or it doesn't). Then the difference function for the whole image can be calculated

$$d(I_t, I_{t+1}) = |I_t - I_{t+1}|.$$

[2]Metropolis N., Rosenbluth A.W., Rosenbluth M.N., Teller A.H. and Teller E. (1953) *Equations of state calculations by fast computing machines* **J. Chem. Phys., 21, 1087–1091**.

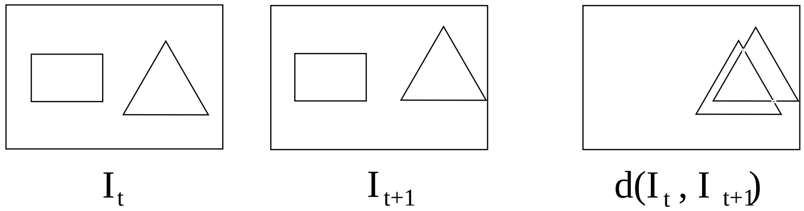


Figure 5: The pixels in the rectangle have the same intensities in $I_t$ and $I_{t+1}$ since it doesn't move, so the difference function $d$ for those pixels evaluates to 0. The triangle moves, so the distance function for the relevant pixels evaluates to 1 and the triangle outline can be seen in the difference image.

Figure 5 shows the difference image, the outline of the object that has moved against a blank background.

The advantage to using energy minimization for the spatial coherence problem instead of simply using a differencing operation is that the former gives us a degree of smoothness. With just differencing, pixels decide independently whether or not they are moving, and it is possible to observe odd situations such as a pixel in the center of an object deciding it is stationary while all around it the other pixels in the same object decide they are moving.

### 4.3 The Ising Model (1924)

The Ising model is a simple combinatorial optimization problem. Given an $n$-pixel image where the label set is $L = \{0, 1\}$, we use $c_0(p)$ to denote the cost associated with assigning pixel $p$ value 0 and $c_1(p)$ to denote the cost associated with assigning it value 1. We want the lowest-cost possible labeling under the constraint that we don't want pixels labeled 0 next to pixels labeled 1, so we are trying to minimize

$$\text{cost} = \sum_{p \text{ labelled } 0} c_0(p) + \sum_{q \text{ labelled } 1} c_1(p) + \lambda \#\text{disconnects}$$

The search space for this minimization is of size $\Theta(2^n)$, but it turns out that the exact minimum can be found in polynomial time. This result is due to Grieg, Porteous and Sehelt (1986), who showed that the problem reduced to minimum cut, for which polynomial time algorithms are well-established.

### 4.4 The Minimum Cut Problem

Given a graph $G = (V, E)$ with distinguished edges $s, t \in V$, an $(S, T)$ **cut** is a partition of $G$ into connected components $S, T$ such that $s \in S$, $t \in T$, $S \cup T = V$

and $S \cap T = \emptyset$. The cost of a cut is given by

$$\text{cost}(S,T) = \sum_{\substack{p \in S \\ q \in T \\ (p,q) \in E}} w(e)$$

The reduction will be given in the next lecture.