Chapter 9: Applications to Computing

9.6: Markov Chain Monte Carlo (MCMC)

Slides (Google Drive) Starter Code (GitHub)

9.6.1 Motivation

Markov Chain Monte Carlo (MCMC) is a technique which can be used to solve hard optimization problems (among other things). In this section, we'll design MCMC algorithms to solve the following two problems, and you will be able to solve many more yourself!

- **The Knapsack Problem**: Suppose you have a knapsack which has some maximum weight capacity. There are n items with weights $w_1, \ldots, w_n > 0$ and values $v_1, \ldots, v_n > 0$, and we want to choose the subset of them that maximizes the total value subject to the weight constraint of the knapsack. How can we do this?
- **The Travelling Salesman Problem (TSP)**: Suppose you want to find the best route (minimizing total distance travelled) between the 50 U.S. state capitals that we want to visit! A valid route starts and ends in the same state capital, and visits each capital exactly once (this is known as the **TSP**, and is known to be NP-Hard). We will design an MCMC algorithm for this as well!

As the name suggests, this technique depends a bit on the idea of **Markov Chains**. Most of this section then will actually be building up the foundations of Markov Chains, and MCMC will follow soon after. In fact, you could definitely understand and code up the algorithm without learning this math, but if you care to know how and why it works (you should), then it is important to learn first!

9.6.2 Markov Chains

Before we define Markov chains, we must define what a stochastic process is.

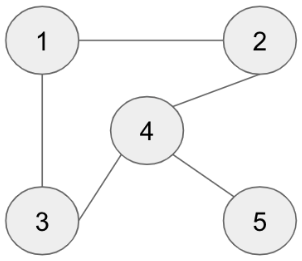
Definition 9.6.1: Discrete-Time Stochastic Process

A **discrete-time stochastic process (DTSP)** is a sequence of random variables $X_0, X_1, X_2, \ldots$ where X_t is the value at time t.

Here are some examples:

- The temperature in Seattle each day. X_0 can be the temperature today, X_1 tomorrow, and so on.
- The price of Google stock at the end of each year. X_0 can be the final price at the end of the year it IPO'd, X_1 the next, and so on.
- The number of people who come to my store each day. X_0 is the number of people who came on the first day, X_1 on the second, and so on.

Consider the following **random walk** on the graph below. You'll see what that means through an example!



Suppose we start at node 1, and at each time step, independently step to a neighboring node with equal probability.

For example, $X_0 = 1$ since at time $t = 0$, we are at node 1. Then, X_1 can be either 2 or 3 (but not 4 or 5 since not neighbors of node 1). And so on. So each X_t just tells us the position we are at at time t, and is always in the set $\{1, 2, 3, 4, 5\}$ (for our example anyway).

This DTSP actually has a lot of structure, and is actually an example of a special type of DTSP called a Markov Chain: can you think about how this particular setup provides a lot of additional constraints over a normal DTSP?

Here are three key properties of a Markov Chain, which we will formalize immediately after:

1. We only have finitely many states (5 in our example: $\{1, 2, 3, 4, 5\}$). (The stock price or temperature example earlier could be any real number).

2. We don't care about the past, **given the present**. That is, the distribution of where we go next ONLY depends on where we are *currently*, and not any past history.

3. The transition probabilities are the *same* at each step (stationary). That is, if we are at node 1 at time $t = 0$ or $t = 152$, we are always equally likely to go to node 2 or 3).

Definition 9.6.2: Markov Chain

A **Markov Chain** is a special DTSP with the following three additional properties:

1. The **state space** $\mathcal{S} = \{s_1, \ldots, s_n\}$ is finite (or countably infinite), so that each $X_t \in \mathcal{S}$.

2. Satisfies the **Markov property**: the future is (conditionally) independent of the past given the present. Mathematically,

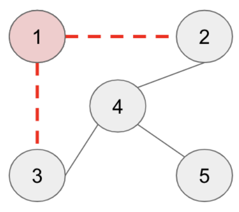
$$\mathbb{P}(X_{t+1} = x_{t+1} \mid X_0 = x_0, X_1 = x_1, \ldots, X_t = x_t) = \mathbb{P}(X_{t+1} = x_{t+1} \mid X_t = x_t)$$

3. Has **stationary** transition probabilities. That is, we always transition from state s_i to s_j with probability independent of the current time. Hence, due to this property and the previous, the transitions are governed by n^2 probabilities: the probability of transitioning to one of n current states to one of n next states. These are stored in a square $n \times n$ **transition probability matrix (TPM)** P, where $P_{ij} = \mathbb{P}(X_{t+1} = s_j \mid X_t = s_i)$ is the probability of transitioning from $s_i \to s_j$ for any and every time t.

If you're a bit confused right now, especially with that last bullet point, this is totally normal and means you are paying attention! Let's construct the TPM for the graph example earlier to see what it means exactly.

9.6.2.1 The Transition Probability Matrix (TPM)

Since we have 5 states, our TPM P will be 5×5. We'll fill out the first row first, which represents the probability of going *from* state s_1 *to* any of the other 5 states.



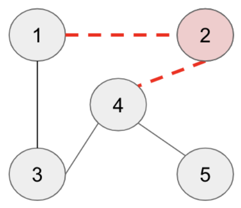
$$\begin{array}{c|ccccc} & \text{To } s_1 & \text{To } s_2 & \text{To } s_3 & \text{To } s_4 & \text{To } s_5 \\ \hline \text{From } s_1 & 0 & \boxed{1/2} & \boxed{1/2} & 0 & 0 \\ \text{From } s_2 & & & & & \\ \text{From } s_3 & & & & & \\ \text{From } s_4 & & & & & \\ \text{From } s_5 & & & & & \end{array}$$

$P(X_{t+1} = 3 \mid X_t = 1)$

$P(X_{t+1} = 2 \mid X_t = 1)$

For example, the second entry of the first row is: given that $X_t = 1$ (we are in state 1 at some time t), what is the probability of going to state 2 next $X_{t+1} = 2$? It's 1/2 because from state 1, we are equally likely to go to state 2 or 3. It isn't possible to go to states 1, 4, and 5, and that's why their respective entries are 0.

Now, how about the second row?

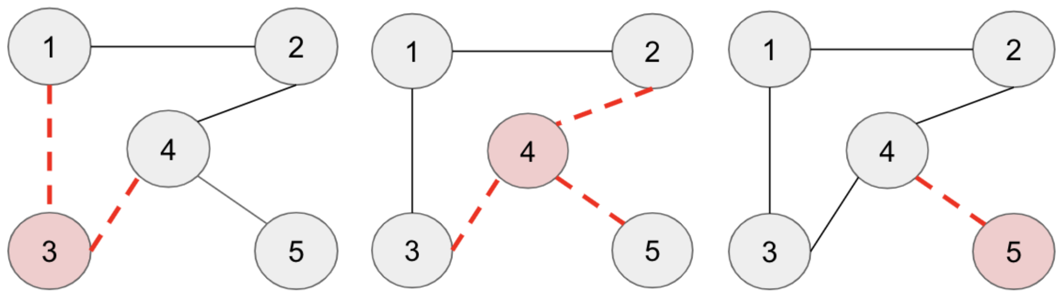


$$\begin{array}{c|ccccc} & \text{To } s_1 & \text{To } s_2 & \text{To } s_3 & \text{To } s_4 & \text{To } s_5 \\ \hline \text{From } s_1 & 0 & 1/2 & 1/2 & 0 & 0 \\ \text{From } s_2 & \boxed{1/2} & 0 & 0 & \boxed{1/2} & 0 \\ \text{From } s_3 & & & & & \\ \text{From } s_4 & & & & & \\ \text{From } s_5 & & & & & \end{array}$$

$P(X_{t+1} = 4 \mid X_t = 2)$

$P(X_{t+1} = 1 \mid X_t = 2)$

From state 2, we can only go to states 1 and 4 as you can see from the graph and the TPM. Try filling out the remaining three rows yourself! These images may help:



Our final answer is:

$$P = \begin{bmatrix} 0 & 1/2 & 1/2 & 0 & 0 \\ 1/2 & 0 & 0 & 1/2 & 0 \\ 1/2 & 0 & 0 & 1/2 & 0 \\ 0 & 1/3 & 1/3 & 0 & 1/3 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Note that in the last row, from state 5, we MUST go to state 4, and so $P_{54} = 1$ and the rest of the row has zero probability. Also note that each ROW sums to 1, but there is no such constraint on the columns. That's because this is secretly a joint PMF right? Given we are in some state s_i ($X_t = s_i$), the probabilities of going to the next state X_{t+1} must sum to 1.

9.6.2.2 Computing Probabilities

The TPM is absolutely crucial; in fact, it defines a Markov chain uniquely. But how can we use it to compute probabilities? The notation is honestly one of the hardest parts of Markov chains. We'll continue to do examples until we are ready for MCMC.

Example(s)

Now let's talk about how to compute some probabilities we may be interested in. Nothing here is "new": it is all based on your core probability knowledge from the previous chapters! Let's say we want to find out the probability we end up at state 5 after two time steps, starting from state 3. That is, compute $\mathbb{P}(X_2 = 5 \mid X_0 = 3)$. Try to do come up with an "intuitive" answer first, and then show your work formally.

Solution You might be able to hack your way around to a solution since it is only two time steps: something like

$$\frac{1}{2} \cdot \frac{1}{3}$$

Intuitively, we can either go to state 4 or 1 from state 3 with equal probability. If we went to state 1, there's no chance we make it to state 5. If we went to state 4, there's a 1/3 chance we go to state 5. So our answer is $1/2 \cdot 1/3 = 1/6$. This is just the LTP conditioning on possible middle states!

Now we'll write this out more generally. This LTP will be a conditional form though: the LTP says that if B_i's partition the sample space:

$$\mathbb{P}(A) = \sum_i \mathbb{P}(A \mid B_i)\,\mathbb{P}(B_i)$$

But what about if we wanted $\mathbb{P}(A \mid C)$? We just condition everything on C as well to get:

$$\mathbb{P}(A \mid C) = \sum_i \mathbb{P}(A \mid B_i, C)\,\mathbb{P}(B_i \mid C)$$

This gives (take B_i to be the event $X_1 = i$: the partition is of size 5):

$$\begin{aligned} \mathbb{P}(X_2 = 5 \mid X_0 = 3) &= \sum_{i=1}^{5} \mathbb{P}(X_2 = 5 \mid X_0 = 3, X_1 = i)\,\mathbb{P}(X_1 = i \mid X_0 = 3) && \text{[LTP]} \\ &= \sum_{i=1}^{5} \mathbb{P}(X_2 = 5 \mid X_1 = i)\,\mathbb{P}(X_1 = i \mid X_0 = 3) && \text{[Markov property]} \end{aligned}$$

The second equation comes because the probability of X_2 given both the positions X_0 and X_1 only depends on X_1 right? Once we know where we are currently, we can forget about the past. But now, we can zero out several of these because $\mathbb{P}(X_1 = i \mid X_0 = 3) = 0$ for $i = 2, 3, 5$. So we are left with just 2 of the 5 terms:

$$= \mathbb{P}(X_2 = 5 \mid X_1 = 1)\,\mathbb{P}(X_1 = 1 \mid X_0 = 3) + \mathbb{P}(X_2 = 5 \mid X_1 = 4)\,\mathbb{P}(X_1 = 4 \mid X_0 = 3)$$

If you have the TPM P (we have this above), try looking up the entries to see if you get the same answer!

$$= P_{15}P_{31} + P_{45}P_{34} = 0 \cdot \frac{1}{2} + \frac{1}{3} \cdot \frac{1}{2} = \frac{1}{6}$$

$\square$

9.6.2.3 The Stationary Distribution

Markov chains have deep ties to not only probability, but also to linear algebra. If you haven't taken a linear algebra class, it's okay; we'll explain everything we need for our application here (it's not too much since we're not going too deep). All we'll assume is that you know what a matrix and a vector are.

Back to our random walk example: suppose we weren't sure where we started. That is, let the vector

$$v = (0.25, 0.45, 0.15, 0.05, 0.10)$$

be such that $P(X_0 = i) = v_i$, where v_i is the i^{th} element of v (these probabilities sum to 1, because we must start in one of these 5 positions). Think of this vector v as our belief distribution of where we are at time $t = 0$. Let's compute vP, the matrix-product of v and P, the transition probability matrix. We'll see what comes out of it after computing and interpreting it! If you haven't taken linear algebra yet, don't worry: vP is the following 5-dimensional row vector:

$$vP = \left(\sum_{i=1}^{5} P_{i1}v_i, \quad \sum_{i=1}^{5} P_{i2}v_i, \quad \sum_{i=1}^{5} P_{i3}v_i, \quad \sum_{i=1}^{5} P_{i4}v_i, \quad \sum_{i=1}^{5} P_{i5}v_i \right)$$

What does vP represent? Let's focus on the first entry, and substitute $v_i = \mathbb{P}(X_0 = i)$ and $P_{i1} = \mathbb{P}(X_1 = 1 \mid X_0 = i)$ (the probability of going from $i \to 1$). We actually get (by LTP over initial states):

$$\sum_{i=1}^{5} P_{i1}v_i = \sum_{i=1}^{5} \mathbb{P}(X_1 = 1 \mid X_0 = i)\,\mathbb{P}(X_0 = i) = \mathbb{P}(X_1 = 1)$$

The second entry is very similar:

$$\sum_{i=1}^{5} P_{i2}v_i = \sum_{i=1}^{5} \mathbb{P}(X_1 = 2 \mid X_0 = i)\,\mathbb{P}(X_0 = i) = \mathbb{P}(X_1 = 2)$$

This is an interesting pattern that holds for the next three entries as well! In fact, the i-th entry of vP is just $\mathbb{P}(X_1 = i)$, so overall, the vector vP **represents your belief distribution at the *next* time step**! That is, right-multiplying by the transition matrix P literally transitions your belief distribution from one time step to the next.

We can also see that for example $vP^2 = (vP)P$ is your belief of where you are after 2 time steps, and by induction, vP^n is your belief of where you are after n time steps.

A natural question might be then, does vP^n have a limit as $n \to \infty$? That is, after a long time, is there a belief distribution (5-dimensional row vector) π such that it never changes again? The answer is unfortunately: it depends. We won't go into the technical details of when it does and doesn't exist (search "Fundamental Theorem of Markov Chains" if you are interested), but this leads us to the following definition:

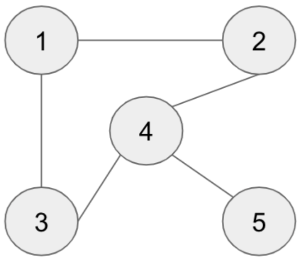
Definition 9.6.3: Stationary Distribution of a Markov Chain

The **stationary distribution** of a Markov Chain with n states (if one exists), is the n-dimensional row vector π (representing a probability distribution: entries which are nonnegative and sum to 1), such that

$$\pi P = \pi$$

Intuitively, it means that the belief distribution at the next time step is the same as the distribution at the current. This typically happens after a "long time" (called the **mixing time**) in the process, meaning after lots of transitions were taken.

We're going to see an example of this visually, which will also help us build our final piece of intuition for MCMC. Consider the Markov Chain we've been using throughout this section:

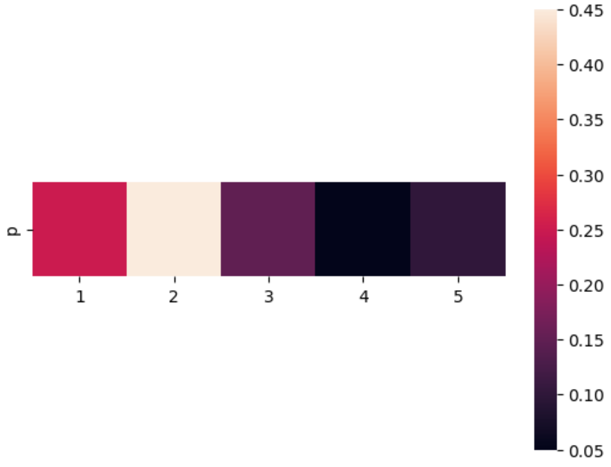


Here is the distribution v that we'll start with. Our Markov Chain happens to have a stationary distribution, so we'll see what happens as we take vP^n for $n \to \infty$ visually.

$$v = (0.25, 0.45, 0.15, 0.05, 0.10)$$

Here is a heatmap of it visually:

Figure 9.6.1: Belief Distribution v at $n = 0$



You can see from the key that darker values mean lower probabilities (hence 4 and 5 are very dark), and that 2 is the lighest value since it has the highest probability.

We'll then show the distribution after 1 step, 5 steps, 10 steps, and 100 steps. Before we continue, what do you think the fifth entry will look like after one time step, the probability of being in node 5? Actually, there is only one way to get to node 5, and that's from node 4, which we start in with probability only 0.05. From there, only a 1/3 chance to get to node 5, so node 5 will only have $0.05/3 = 1/60$ probability at time step 1 and hence be super dark.

Figure 9.6.2: Belief Distribution vP at $n = 1$

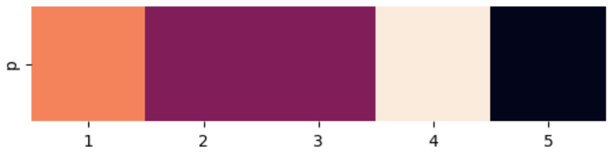


Figure 9.6.3: Belief Distribution vP^5 at $n = 5$

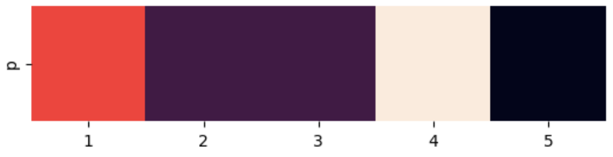


Figure 9.6.4: Belief Distribution vP^{10} at $n = 10$

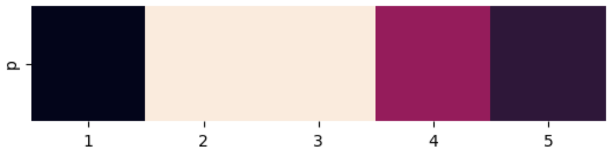
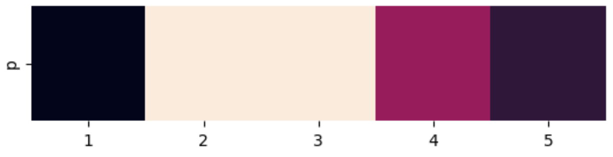


Figure 9.6.5: Belief Distribution vP^{100} at $n = 100$



It turns out that after just $n = 100$ time steps, we start getting the same distribution over and over again (see $t = 10$ and $t = 100$: there's already almost no difference)! This limiting value of vP^n is the stationary distribution!

$$\pi = \lim_{n\to\infty} vP^n = (0.12, 0.28, 0.28, 0.18, 0.14)$$

Suppose $\pi = vP^{100}$ above. Once we find π such that $\pi P = \pi$ for the first time, that means that if we transition again, we get

$$\pi P^2 = (\pi P)P = \pi P = \pi$$

(applying the equality $\pi P = \pi$ twice). **That means, by just running the Markov Chain for several time steps, we actually reached our stationary distribution!** This is the most crucial observation for MCMC.

9.6.3 Markov Chain Monte Carlo (MCMC)

This brings us to our strategy for Markov Chain Monte Carlo. Again, remember that no matter where we start with distribution v, by simulating the Markov Chain many steps, we will eventually reach the stationary distribution $\pi = \lim_{n\to\infty} vP^n$. Meaning, if we start in some state at simulate the chain for a large number of steps (randomly choosing the next transition), **it will give us a sample from the stationary distribution.**

Actually, MCMC is generally a technique to sample from a hard distribution that we can't explicitly write out. Oftentimes we can't compute vP^n for n very large because a Markov Chain usually has way too many states (5 is nothing). Imagine how long it would take a computer to compute vP^{100} even if there were 1000 states (1000×1000 matrix P). We'll see how we can take advantage of this amazing fact below!

Definition 9.6.4: Markov Chain Monte Carlo (MCMC)

Markov Chain Monte Carlo (MCMC) is a technique which can be used to hard optimization problems (though generally it is used to sample from a distribution). The general strategy is as follows:

I. Define a Markov Chain with states being possible solutions, and (implicitly defined) transition probabilities that result in the stationary distribution π having higher probabilities on "good" solutions to our problem. We don't actually compute π, but we just want to define the Markov Chain such that the stationary distribution would have higher probabilities on more desirable solutions.

II. Run MCMC (simulate the Markov Chain for many iterations until we reach a "good" state/-solution). This means: start at some initial state, and transition according to the transition probability matrix (TPM) for a long time. This will eventually take us to our stationary distribution which has high probability on "good" solutions!

Again, if this doesn't make sense yet, that's totally fine. We will apply this two-step procedure to two examples below so you can understand better how it works!

9.6.3.1 Knapsack Problem

Definition 9.6.5: Knapsack Problem

The **0-1 Knapsack Problem** is defined as follows:

- Given n items with weights $w_1, \dots, w_n > 0$ and values $v_1, \dots, v_n > 0$, and a knapsack with weight limit W.
- Goal: Find the most valuable subset of items which satisfy the weight constraint of the knapsack!

More formally, we let $\mathbf{x} = (x_1, \dots, x_n) \in \{0,1\}^n$ be the n-dimensional vector of whether or not we take each item (1 means take, 0 means don't take). Our goal is to maximize the total value $\sum_{i=1}^n v_i x_i$ in our knapsack subject to our weight constraint $\sum_{i=1}^n w_i x_i \leq W$.

Note that our total value is the sum of the values of the items we take: think about why $\sum v_i x_i$ is the total value (remember that x_i is either 0 or 1). This problem has 2^n possible solutions (either take each item or don't), and so is combinatorially hard (exponentially many solutions). If I asked you to write a program to do this, would you even know where to begin, except by writing the brute-force solution?

MCMC to the rescue!

I. **Define a Markov Chain with states being possible solutions, and (implicitly defined) transition probabilities that result in the stationary distribution π having higher probabilities on "good" solutions to our problem.**

We'll define a Markov Chain with 2^n states (that's huge!). The states will be all possible solutions: binary vectors $\mathbf{x}$ of length n (only having 0/1 entries). We'll then define our transitions to go to "good" states (ones that satisfy our weight constraint), while keeping track of the best solution so far. This way, our stationary distribution has higher probabilities on good solutions than bad ones. Hence, when we sample from the distribution (simulating the Markov chain), we are likely to get a good solution!

Algorithm 1 MCMC for 0-1 Knapsack Problem

```
1: x ← vector of $n$ zeros, where $x_i$ is a binary vector in $\{0,1\}^n$ which represents whether or not we have
   item $i$. (Initially, start with an empty knapsack).
2: best_x ← x
3: for $t = 1, \ldots,$ NUM_ITER do
4:     $k$ ← a random integer in $\{1, 2, \ldots, n\}$.
5:     new_x ← x but with x[$k$] flipped ($0 \to 1$ or $1 \to 0$).
6:     if new_x satisfies weight constraint then
7:         x ← new_x
8:     if value(x) > value(best_x) then
9:         best_x ← x
```

II. **Run MCMC (simulate the Markov Chain for many iterations until we reach a "good" state/solution). This means: start at some initial state, and transition according to the transition probability matrix (TPM) for a long time. This will eventually take us to our stationary distribution which has high probability on "good" solutions!**

Basically, this algorithm starts with the guess of x being all zeros (no items). Then, for NUM_ITER steps, we simulate the Markov Chain. Again, what this does is give us a sample from our stationary distribution. Inside the loop, we literally just choose a random object and flip whether or not we have it. We maintain track of the best solution so far and return it.

That's all there is to it! This is such a "dumb" solution right? We just start somewhere and randomly transition for a long time and hope our answer is good. So MCMC definitely won't guarantee us to get the best solution, but it leads to "dumb" solutions that actually work quite well in practice. We are guaranteed though (provided we take enough transitions), to sample from the stationary distribution which has higher probabilities on good solutions. This is because we only transition to solutions that maintain feasibility.

Note: This is just one version of MCMC for the knapsack problem, there are definitely probably better versions. It would be better to transition to solutions which have higher value, not just feasible solutions like we did. The next example does a better job of this!

9.6.3.2 Travelling Salesman Problem (TSP)

9.6.3.3 Travelling Salesman Problem

Definition 9.6.6: Travelling Salesman Problem

Given n locations and distances between each pair, we want to find an ordering of them that:

- Starts and ends in the same location.
- Visits each location exactly once (except the starting location twice).
- Minimizes the total distance travelled.

You can imagine an instantiation of this problem for the US Postal Service. A mail delivery person wants to start and end at the post office, and find the most efficient route which delivers all the mail to the residents.

Again, where would you even begin on trying to solve this, other than brute-force? MCMC to the rescue again! This time, our algorithm will be more clever than the previous.

I. **Define a Markov Chain with states being possible solutions, and (implicitly defined) transition probabilities that result in the stationary distribution π having higher probabilities on "good" solutions to our problem.**

   We'll define a Markov Chain with $n!$ states (that's huge!). The states will be all possible solutions (state=route): all orderings of the n locations. We'll then define our transitions to go to "good" states (ones that go to lower-distance routes), while keeping track of the best solution so far. This way, our stationary distribution has higher probabilities on good solutions than bad ones. Hence, when we sample from the distribution (simulating the Markov chain), we are likely to get a good solution!

Algorithm 2 MCMC for Travelling Salesman Problem (TSP)

```
1: route ← random permutation of the $n$ locations.
2: best_route ← route
3: for $i = 1, \ldots,$ NUM_ITER do
4:     new_route ← route; but with two successive locations in route swapped.
5:     $\Delta \leftarrow$ dist(new_route) − dist(route)
6:     if $\Delta < 0$ OR ($T > 0$ AND $Unif(0,1) < e^{-\Delta/T}$) then
7:         route ← new_route
8:     if dist(route) < dist(best_route) then
9:         best_route ← route
```

II. **Run MCMC (simulate the Markov Chain for many iterations until we reach a "good" state/solution). This means: start at some initial state, and transition according to the transition probability matrix (TPM) for a long time. This will eventually take us to our stationary distribution which has high probability on "good" solutions!**

   We will start with a random state (route). At at each iteration, propose a new state (route) as follows: choose a random index from $\{1, 2, \ldots, n\}$, and swap that location with the successive (next) location in the route, possibly with wraparound if index 50 is chosen. If the proposed route has lower total distance (is better) than the current route, we will always transition to it (exploitation). Otherwise, if $T > 0$, with probability $e^{-\Delta/T}$, update the current route to the proposed route, where $\Delta > 0$ is the increase in total distance. This allows us to transition to a "worse" route occasionally (exploration), and get out of local optima! Repeat this for NUM_ITER transitions from the initial state (route), and output the shortest route during the entire process (which may not be the last route).

Again, this is such a "dumb" solution right? But also very clever! We just start somewhere and randomly transition for a long time and hope our answer is good. And it should be: after a long time, our route distance increasingly gets better and better, so we should expect a rather good solution!

9.6.4 Summary

Once again, we've used probability to make our lives easier. There are definitely papers and research on how to solve these problems deterministically, but this is one of the simplest algorithms you can get, and it uses randomness! Again, the idea of MCMC for optimization is: define the state space to be all possible solutions, define transitions to go to better states, and just run it and wait!