Car Tracking

Stanford CS221 Fall 2016-2017

Owner CA: Pujun Bhatnagar

Version: 1

General Instructions

This (and every) assignment has a written part and a programming part.

a. This icon means a written answer is expected in `car.pdf`.

b. This icon means you should write code in `submission.py`.

You should modify the code in `submission.py` between

```
# BEGIN_YOUR_CODE
```

and

```
# END_YOUR_CODE
```

but you can add other helper functions outside this block if you want. Do not make changes to files other than `submission.py`.

Your code will be evaluated on two types of test cases, **basic** and **hidden**, which you can see in `grader.py`. Basic tests, which are fully provided to you, do not stress your code with large inputs or tricky corner cases. Hidden tests are more complex and do stress your code. The inputs of hidden tests are provided in `grader.py`, but the correct outputs are not. To run all the tests, type

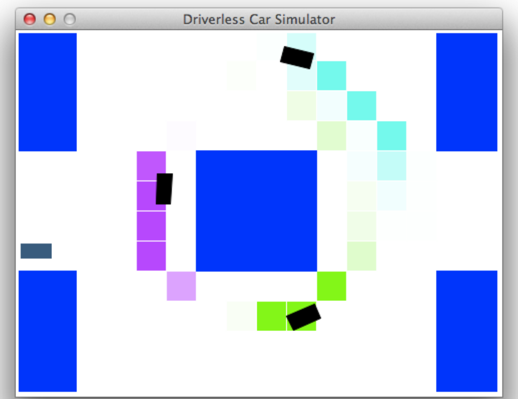
```
python grader.py
```

This will tell you only whether you passed the basic tests. On the hidden tests, the script will alert you if your code takes too long or crashes, but does not say whether you got the correct output. You can also run a single test (e.g., `3a-0-basic`) by typing

```
python grader.py 3a-0-basic
```

We strongly encourage you to read and understand the test cases, create your own test cases, and not just blindly run `grader.py`.

This assignment is a modified version of the Driverless Car assignment written by Chris Piech.



A study by the World Health Organisation found that road accidents kill a shocking 1.24 million people a year worldwide. In response, there has been great interest in developing autonomous driving technology that can can drive with calculated precision and reduce this death toll. Building an autonomous driving system is an incredibly complex endeavor. In this assignment, you will focus on the sensing system, which allows us to track other cars based on noisy sensor readings.

Getting started. Let's start by trying to drive manually:

```
python drive.py -l lombard -i none
```

You can steer by either using the arrow keys or 'w', 'a', and 'd'. The up key and 'w' accelerates your car forward, the left key and 'a' turns the steering wheel to the left, and the right key and 'd' turns the steering wheel to the right. Note that you cannot reverse the car or turn in place. Quit by pressing 'q'. Your goal is to drive from the start to finish (the green box) without getting in an accident. How well can you do on crooked Lombard street without knowing the location of other cars? Don't worry if you aren't very good; the staff was only able to get to the finish line 4/10 times. This 60% accident rate is pretty abysmal, which is why we're going to build an AI to do this.

Flags for `python drive.py`:

- `-a`: Enable autonomous driving (as opposed to manual).
- `-i <inference method>`: Use `none`, `exactInference`, `particleFilter` to (approximately) compute the belief distributions.
- Loading [MathJax]/extensions/MathMenu.js or `lombard`). Defaults to `small`.
- `-d`: Debug by showing all the cars on the map.

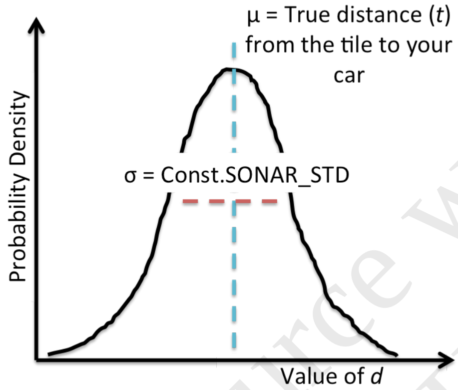
- `-p`: All other cars remain parked (so that they don't move).

Modeling car locations. We assume that the world is a two-dimensional rectangular grid on which your car and K other cars reside. At each time step t, your car gets a noisy estimate of the distance to each of the cars. As a simplifying assumption, we assume that each of the K other cars moves independently and that the noise in sensor readings for each car is also independent. Therefore, in the following, we will reason about each car independently (notationally, we will assume there is just one other car).

At each time step t, let $C_t \in \mathbb{R}^2$ be a pair of coordinates representing the actual location of the single other car (which is unobserved). We assume there is a local conditional distribution $p(c_t \mid c_{t-1})$ which governs the car's movement. Let $a_t \in \mathbb{R}^2$ be your car's position, which you observe and also control. To minimize costs, we use a simple sensing system based on a microphone. The microphone provides us with D_t, which is a Gaussian random variable with mean equal to the distance between your car and the other car and variance σ^2 (in the code, σ is `Const.SONAR_STD`, which is about two-thirds the length of a car). In symbols,

$$D_t \sim \mathcal{N}(\|a_t - C_t\|, \sigma^2).$$

For example, if your car is at $a_t = (1, 3)$ and the other car is at $C_t = (4, 7)$, then the actual distance is 5 and D_t might be 4.6 or 5.2, etc. Use `util.pdf(mean, std, value)` to compute the probability density function (PDF) of a Gaussian with given mean and standard deviation, evaluated at `value`. Note that the PDF does not return a probability (densities can exceed 1), but for the purposes of this assignment, you can get away with treating it like a probability. The Gaussian probability density function for the noisy distance observation D_t, which is centered around your distance to the car $\mu = \|a_t - C_t\|$, is shown in the following figure:



Your job is to implement a car tracker that (approximately) computes the posterior distribution $\mathbb{P}(C_t \mid D_1 = d_1, \dots, D_t = d_t)$ (your beliefs of where the other car is) and update it for each $t = 1, 2, \dots$. We will take care of using this information to actual drive the car (i.e., set a_t as to avoid collision with c_t), so you don't have to worry about that part.

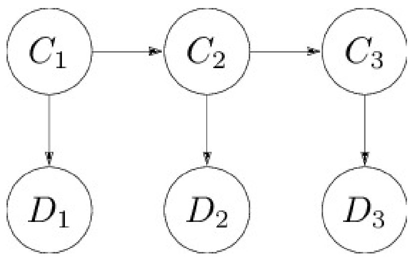
To simplify things, we will discretize the world into **tiles** represented by `(row, col)` pairs, where `0 <= row < numRows` and `0 <= col < numCols`. For each tile, we store a probability distribution whose values can be accessed by `self.belief.getProb(row, col)`. To convert from a tile to a location, use `util.rowToY(row)` and `util.colToX(col)`.

Problem 1 will be a warmup. In Problems 2 and 3, you will implement `ExactInference`, which computes a full distribution over tiles `(row, col)`. In Problem 4, you will implement `ParticleFilter`, which works with particle-based represention of this distribution.

Note: as a notational reminder, the lower case p(x) is the local distribution defined by the user. On the other hand, the quantity $\mathbb{P}(X = x)$ is not defined, but follows from probabilistic inference. Please review lecture slides for more details.

Problem 1: Warmup

First, let us look at a simplified version of the car tracking problem. For this problem only, let $C_t \in \{0, 1\}$ be the actual location of the car we wish to observe at time step $t \in \{1, 2, 3\}$. Let $D_t \in \{0, 1\}$ be a sensor reading for the location of that car measured at time t. Here's what the Bayesian network (it's an HMM, in fact) looks like:



The distribution over the initial car distribution is uniform; that is, for each value $c_1 \in \{0, 1\}$:

$$p(c_1) = 0.5.$$

The following local conditional distribution governs the movement of the car (with probability ϵ, the car moves). For each $t \in \{2, 3\}$:

$$p(c_t \mid c_{t-1}) = \begin{cases} \epsilon & \text{if } c_t \neq c_{t-1} \\ 1-\epsilon & \text{if } c_t = c_{t-1}. \end{cases}$$

The following local conditional distribution governs the noise in the sensor reading (with probability η, the sensor reports the wrong position). For each $t \in \{1, 2, 3\}$:

$$p(d_t \mid c_t) = \begin{cases} \eta & \text{if } d_t \neq c_t \\ 1-\eta & \text{if } d_t = c_t. \end{cases}$$

Below, you will be asked to find the posterior distribution for the car's position at the second time step (C_2) given different sensor readings.

Important: For the following computations, try to follow the general strategy described in lecture (marginalize non-ancestral variables, condition, and perform variable elimination). Try to delay normalization until the very end. You'll get more insight than trying to chug through lots of equations.

a. [2 points] Suppose we have a sensor reading for the second timestep, $D_2 = 0$. Compute the posterior distribution $\mathbb{P}(C_2 = 1 \mid D_2 = 0)$. We encourage you to draw out the (factor) graph.

We can eliminate D_1, D_3, and C_3 because they have nothing attached to them. Now, we are left with three factors, $p(c_1)$, $p(c_2 \mid c_1)$, and $p(d_2 \mid c_2)$. We can marginalize out C_1, which replaces $p(c_1)$ and $p(c_2 \mid c_1)$ with a new factor:

$$f_1(c_2) = \sum_{c_1} p(c_1)p(c_2 \mid c_1) = 0.5\epsilon + 0.5(1-\epsilon) = 0.5.$$

(This is intuitive since C_1 is uniform and the transition is symmetric, so this doesn't tell us anything about C_2.)

$$\mathbb{P}(C_2 = c_2 \mid D_2 = 0) \propto f_1(c_2)p(d_2 = 0 \mid c_2) = \begin{cases} f_1(c_2)\eta & \text{if } c_2 = 1 \\ f_1(c_2)(1-\eta) & \text{if } c_2 = 0. \end{cases}$$

Therefore,

$$\mathbb{P}(C_2 = 1 \mid D_2 = 0) = \eta.$$

Above result is from normalizing $\frac{f_1(c_2)\eta}{f_1(c_2)\eta + f_1(c_2)(1-\eta)}$

b. [2 points] Suppose a time step has elapsed and we got another sensor reading, $D_3 = 1$, but we are still interested in C_2. Compute the posterior distribution $\mathbb{P}(C_2 = 1 \mid D_2 = 0, D_3 = 1)$. The resulting expression might be moderately complex. We encourage you to draw out the (factor) graph.

Now, we cannot ignore C_3 and D_3. But we can marginalize C_3, creating a new factor:

$$f_2(c_2) = \sum_{c_3} p(c_3 \mid c_2)p(d_3 = 1 \mid c_3) = \begin{cases} \epsilon\eta + (1-\epsilon)(1-\eta) & \text{if } c_2 = 1 \\ (1-\epsilon)\eta + \epsilon(1-\eta) & \text{if } c_2 = 0. \end{cases}$$

Now, we compute the desired query by multiplying in all the factors:

$$\begin{aligned} \mathbb{P}(C_2 = c_2 \mid D_2 = 0, D_3 = 1) &\propto f_1(c_2)p(d_2 = 0 \mid c_2)f_2(c_2) \\ &= \begin{cases} 0.5\eta(\epsilon\eta + (1-\epsilon)(1-\eta)) & \text{if } c_2 = 1 \\ 0.5(1-\eta)((1-\epsilon)\eta + \epsilon(1-\eta)) & \text{if } c_2 = 0. \end{cases} \end{aligned}$$

Normalizing, we obtain:

$$\mathbb{P}(C_2 = 1 \mid D_2 = 0, D_3 = 1) = \frac{\epsilon\eta^2 + (1-\epsilon)(1-\eta)\eta}{\epsilon\eta^2 + 2(1-\epsilon)(1-\eta)\eta + \epsilon(1-\eta)^2}.$$

c. [3 points] Suppose $\epsilon = 0.1$ and $\eta = 0.2$.

i. Compute and compare the probabilities $\mathbb{P}(C_2 = 1 \mid D_2 = 0)$ and $\mathbb{P}(C_2 = 1 \mid D_2 = 0, D_3 = 1)$. Give numbers, round your answer to 4 significant digits.

$$\mathbb{P}(C_2 = 1 \mid D_2 = 0) = 0.2$$

$$\mathbb{P}(C_2 = 1 \mid D_2 = 0, D_3 = 1) \approx 0.4157.$$

ii. How did adding the second sensor reading $D_3 = 1$ change the result? Explain your intuition in terms of the car positions with respect to the observations.

The intuition is that since ϵ is small, the position is unlikely to change across time steps. Therefore, when the new reading of $D_3 = 1$ is observed, it increases the belief that the car was at position 1 at time $t = 2$, despite the sensor reading $D_2 = 0$ that suggests a priori that the car is unlikely (0.2) to be at position 1.

iii. What would you have to set ϵ while keeping $\eta = 0.2$ so that $\mathbb{P}(C_2 = 1 \mid D_2 = 0) = \mathbb{P}(C_2 = 1 \mid D_2 = 0, D_3 = 1)$? Explain your intuition in terms of the car positions with respect to the observations.

Set $\epsilon = 0.5$. This corresponds to the car moving to 0 and 1 with probability 0.5 regardless of the previous position. Therefore, $D_3 = 1$ doesn't matter. This can be checked numerically as well.

Problem 2: Emission probabilities

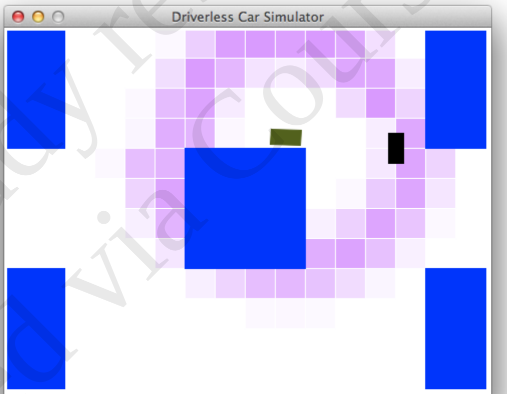
In this problem, we assume that the other car is stationary (e.g., $C_t = C_{t-1}$ for all time steps t). You will implement a function `observe` that upon observing a new distance measurement $D_t = d_t$ updates the current posterior probability from

$$\mathbb{P}(C_t \mid D_1 = d_1, \dots, D_{t-1} = d_{t-1})$$

to

$$\mathbb{P}(C_t \mid D_1 = d_1, \dots, D_t = d_t) \propto \mathbb{P}(C_t \mid D_1 = d_1, \dots, D_{t-1} = d_{t-1}) p(d_t \mid c_t),$$

where we have multiplied in the emission probabilities $p(d_t \mid c_t)$ described earlier. The current posterior probability is stored as `self.belief` in `ExactInference`, which you should update `self.belief` in place.



a. [7 points] Fill in the `observe` method in the `ExactInference` class of `submission.py`. This method should update the posterior probability of each tile given the observed noisy distance. After you're done, you should be able to find the stationary car by driving around it (`-p` means cars don't move):

Notes:

- You can start driving with exact inference now.

```
python drive.py -a -p -d -k 1 -i exactInference
```

  You can also turn off `-a` to drive manually.
- If you do decide to run drive.py on corn, please ssh into corn machines with either option **-X** or **-Y** in order to get the graphical interface, or else you will get some display error message. Note: do expect this graphical interaction to be slow in response; `drive.py` is only for you to enjoy the game :), but not used for grading!
- Remember to normalize the updated posterior probability (see useful functions provided in utils.py).
- Loading [MathJax]/extensions/MathMenu.js ...er will sometimes drive in circles around the middle block before heading for the target area. In general, don't worry too much about driving the car. Instead, focus on if your car tracker correctly infers the

location of other cars.
- Don't worry if your car crashes once in a while! Accidents do happen, whether you are human or AI. However, even if there was an accident, your driver should have been aware that there was a high probability that another car was in the area.

Problem 3: Transition probabilities

Now, let's consider the case where the other car is moving according to transition probabilities $p(c_{t+1} \mid c_t)$. We have provided the transition probabilities for you in `self.transProb`. Specifically, `self.transProb[(oldTile, newTile)]` is the probability of the other car being in `newTile` at time step $t+1$ given that it was in `oldTile` at time step t.

In this part, you will implement a function `elapseTime` that updates the posterior probability about the location of the car at a **current** time t

$$\mathbb{P}(C_t = c_t \mid D_1 = d_1, \dots, D_t = d_t)$$

to the **next** time step $t+1$ conditioned on the same evidence, via the recurrence:

$$\mathbb{P}(C_{t+1} = c_{t+1} \mid D_1 = d_1, \dots, D_t = d_t) \propto \sum_{c_t} \mathbb{P}(C_t = c_t \mid D_1 = d_1, \dots, D_t = d_t) p(c_{t+1} \mid c_t).$$

Again, the posterior probability is stored as `self.belief` in `ExactInference`.

a. [7 points] Finish `ExactInference` by implementing the `elapseTime` method. When you are all done, you should be able to track a moving car well enough to drive autonomously:

```
python drive.py -a -d -k 1 -i exactInference
```

Notes:

- You can also drive autonomously in the presence of more than one car:

```
python drive.py -a -d -k 3 -i exactInference
```

- You can also drive down Lombard:

```
python drive.py -a -d -k 3 -i exactInference -l lombard
```

On Lombard, the autonomous driver may attempt to drive up and down the street before heading towards the target area. Again, focus on the car tracking component, instead of the actual driving.

Problem 4: Particle filtering

Though exact inference works well for the small maps, it wastes a lot of effort computing probabilities for cars being on unlikely tiles. We can solve this problem using a particle filter which has complexity linear in the number of particles rather than linear in the number of tiles. Implement all necessary methods for the `ParticleFilter` class in `submission.py`. When complete, you should be able to track cars nearly as effectively as with exact inference.

a. [18 points] Some of the code has been provided for you. For example, the particles have already been initialized randomly. You need to fill in the `observe` and `elapseTime` functions. These should modify `self.particles`, which is a map from tiles `(row, col)` to the number of times that particle occurs, and `self.belief`, which needs to be updated after you resample the particles.

You should use the same transition probabilities as in exact inference. The belief distribution generated by a particle filter is expected to look noisier compared to the one obtained by exact inference.

```
python drive.py -a -i particleFilter -l lombard
```

To debug, you might want to start with the parked car flag (`-p`) and the display car flag (`-d`).

Note: The random number generator inside util.weightedRandomChoice behaves differently on different systems' versions of Python (e.g., Unix and Windows versions of Python). **Please test this question (run grader.py) on corn.** When copying files to corn, make sure you copy the entire folder using scp with the recursive option `-r`.

Problem 5: Which car is it?

So far, we have assumed that we have a distinct noisy distance reading for each car, but in reality, our microphone would just pick up an undistinguished set of these signals, and we wouldn't know which distance reading corresponds to which car. First, let's extend the notation from before: let $C_{ti} \in \mathbb{R}^2$ be the location of the i-th car at the time step t, for $i = 1, \dots, K$ and $t = 1, \dots, T$. Recall that all the cars move independently according to the transition dynamics as before.

Let $D_{ti} \in \mathbb{R}$ be the noisy distance measurement of the i-th car, which is now not observed. Instead, we observe the **set** of distances $D_t = \{D_{t1}, \dots, D_{tK}\}$ (assume that all distances are all distinct). Alternatively, you can think of $E_t = (E_{t1}, \dots, E_{tK})$ as a list which is a uniformly random permutation of the noisy distances $(D_{t1}, \dots, D_{tK})$. For example, suppose $K = 2$ and $T = 2$. Before, we might have gotten distance readings of 1 and 2 for the first car and 3 and 4 for the second car. Now, our sensor readings would be permutations of $\{1, 3\}$ and $\{2, 4\}$. Thus, even if we knew the second car was distance 3 away at time $t = 1$, we wouldn't know if it moved farther (4 away) or closer (2 away) at time $t = 2$.

a. [5 points] Suppose we have $K = 2$ cars and one time step $T = 1$. Write an expression for the conditional distribution $\mathbb{P}(C_{11}, C_{12} \mid E_1 = e_1)$ as a function of the PDF of a Gaussian $p_{\mathcal{N}}(v; \mu, \sigma^2)$ and the prior probability $p(c_{11})$ and $p(c_{12})$ over car locations. Your final answer should not contain variables d_{11}, d_{12}.

$p_{\mathcal{N}}(v; \mu, \sigma^2)$ is the likelihood of a random variable, v, in a Gaussian distribution with mean μ and standard deviation σ

Hint: for $K = 1$, the answer would be

$$\mathbb{P}(C_{11} = c_{11} \mid E_1 = e_1) \propto p(c_{11}) p_{\mathcal{N}}(e_{11}; \|a_1 - c_{11}\|, \sigma^2).$$

where a_t is the position of the car at time t. You might find it useful to draw the Bayesian network and think about the distribution of E_t given $D_{t1}, \dots, D_{tK}$.

Note: to reduce notation, we will write, for example, $p(c_{11} \mid e_{11})$ instead of $\mathbb{P}(C_{11} = c_{11} \mid E_{11} = e_{11})$. Using the fact that $e_1 = (e_{11}, e_{12})$ and Bayes rule, we can equivalently express $p(c_{11}, c_{12} \mid e_{11}, e_{12})$ as

$$p(c_{11}, c_{12} \mid e_{11}, e_{12}) \propto p(c_{11}, c_{12}) p(e_{11}, e_{12} | c_{11}, c_{12}).$$

There two ways that e_{11} and e_{12} could have arisen (two permutations):

$$p(e_{11}, e_{12} \mid c_{11}, c_{12}) \propto p(d_{11} = e_{11}, d_{12} = e_{12} \mid c_{11}, c_{12}) + p(d_{11} = e_{12}, d_{12} = e_{11} \mid c_{11}, c_{12}).$$

Once we know the order of $\{e_{1i}\}$, observation of each car becomes independent:

$$p(e_{11}, e_{12} \mid c_{11}, c_{12}) \propto p(d_{11} = e_{11} \mid c_{11}) p(d_{12} = e_{12} \mid c_{12}) + p(d_{11} = e_{12} \mid c_{11}) p(d_{12} = e_{11} \mid c_{12}).$$

As hint suggests, $d_{ti} \sim \mathcal{N}(\|a_t - c_{ti}\|, \sigma^2)$ and this yields

$$\begin{aligned} p(e_{11}, e_{12} \mid c_{11}, c_{12}) &\propto p_{\mathcal{N}}(e_{11}; \|a_1 - c_{11}\|, \sigma^2) p_{\mathcal{N}}(e_{12}; \|a_1 - c_{12}\|, \sigma^2) + \\ &\quad p_{\mathcal{N}}(e_{12}; \|a_1 - c_{11}\|, \sigma^2) p_{\mathcal{N}}(e_{11}; \|a_1 - c_{12}\|, \sigma^2). \end{aligned}$$

Finally, combining these and the fact that cars move independently, we obtain

$$\begin{aligned} p(c_{11}, c_{12} \mid e_1) &\propto p(c_{11}) p(c_{12}) (p_{\mathcal{N}}(e_{11}; \|a_1 - c_{11}\|, \sigma^2) p_{\mathcal{N}}(e_{12}; \|a_1 - c_{12}\|, \sigma^2) + \\ &\quad p_{\mathcal{N}}(e_{12}; \|a_1 - c_{11}\|, \sigma^2) p_{\mathcal{N}}(e_{11}; \|a_1 - c_{12}\|, \sigma^2)). \end{aligned}$$

b. [4 points] Assuming the prior $p(c_{1i})$ is the same for all i, show that the number of assignments for all K cars $(c_{11}, \dots, c_{1K})$ that obtain the maximum value of $\mathbb{P}(C_{11} = c_{11}, \dots, C_{1K} = c_{1K} \mid E_1 = e_1)$ is at least $K!$.

You can also assume that the car locations that maximize the probability above are unique ($C_{1i} \neq c_{1j}$ for all $i \neq j$).

Since the prior is the same, there is no innate preference for how to assign observations to cars. Thus, for any setting $(c_{11}, \dots, c_{1K})$ that maximizes the probability, we can permute these car locations in $K!$ ways without changing the probability.

c. [2 points] For general K, what is the treewidth corresponding to the posterior distribution over all K car locations at all T time steps conditioned on all the sensor readings:

$$\mathbb{P}(C_{11} = c_{11}, \dots, C_{1K} = c_{1K}, \dots, C_{T1} = c_{T1}, \dots, C_{TK} = c_{TK} \mid E_1 = e_1, \dots, E_T = e_T)?$$

Briefly justify your answer.

The treewidth is K. After conditioning, we'll have a factor between all the K cars for each timestep. There is also a factor between each car in time t and time $t + 1$ (for all relevant times t).
With the elimination order from time step 1 to T, we will create new factors on (at most) K variables at a time because there are K variables in the eliminated variable's Markov blanket.

d. [6 points] (extra credit) Now suppose you change your sensors so that at each time step t, they return the list of exact positions of the K cars, but shifted (with wrap around) by a random amount. For example, if the true car positions at time step 1 are $c_{11} = 1, c_{12} = 3, c_{13} = 8, c_{14} = 5$, then e_1 would be $[1, 3, 8, 5]$, $[3, 8, 5, 1]$, $[8, 5, 1, 3]$, or $[5, 1, 3, 8]$, each with probability $1/4$. Describe an efficient algorithm for computing $p(c_{ti} \mid e_1, \dots, e_T)$ for any time step t and car i. Your algorithm should not be exponential in K or T.

The key observation is to define an auxiliary variable $z_t \in \{1, \dots, K\}$, which is the amount by which your observations have been shifted, and note that c_t is a deterministic function of (e_t, z_t); given e_t, the only uncertainty in c_t is z_t, which importantly takes on K values rather than $K!$

The rest is a matter of getting the notation right without getting confused. For clarity, let the transition function be denoted as $A(c_{ti}, c_{t-1,i}) = p(c_{ti} \mid c_{t-1,i})$. So now the transitions can be rewritten as:

$$p(c_t \mid c_{t-1}) = \prod_{i=1}^{K} A(e_{t,M(i-z_t)}, e_{t-1,M(i-z_{t-1})})$$

where $M(a) = ((a + K - 1) \bmod K) + 1$ wraps a to the range $1, \dots, K$. Note that this is a function of z_{t-1} and z_t only. The emissions $p(e_t \mid c_t) = 1/K$ if e_t is a shifted (and wrapped around) version of c_t, and 0 otherwise. Since we are considering those c_t constructed from (e_t, z_t), we can ignore the emissions.

The result is a factor graph over $(z_1, \dots, z_T)$, where the only potentials are the adjacent ones between z_{t-1} and z_t. (At this point, you should only think about it as a factor graph without worrying about the Bayesian network.) This is now just a simple chain-structured factor graph with domain K. All the marginals can be computed exactly in $O(K^2T)$ time via variable elimination.