8.5 Modeling w/ Distributions

11/17

PCAP system

mixture distribution

Primitives

Delta
- spike on single element
- DDist({V: 1.0}) ← oh cool

Uniform
- Yn to each
- was in Counter
- p = 1.0/len(eles)
- return DDist(dict([e, p for e in eles]))

Square
- $p = \frac{1}{hi - lo}$
- for all integers lo → hi-1
- (?how is this different than uniform?

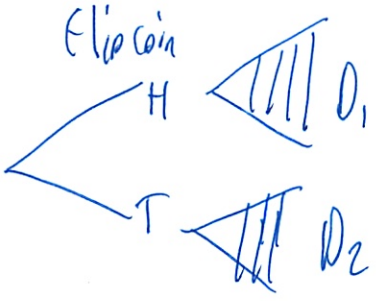
Triangle
- peak, half width

(2)

Mixtures

- Specify 2 dist
- and mixing parameter p $0 \leq p \leq 1$

So Flip coin



$$Pr_{mix}(x) = p \cdot P(D_1 = x) + (1-p) P(D_2 = x)$$

$p = .5$

↑ see here flipped b/w value or 0 So if value, shown

↑ height of each object cut in half

Stochastic State Machines

Homework 4: Distributions

Please turn in a single, printed, stapled, clearly readable document with all your answers, including code and plots. Hand it in to 34-501 during Week 12 software lab, **Nov. 23.**

1 Introduction

Do `athrun 6.01 update` to get the files for this lab, which will be in `Desktop/6.01/lab10/designLab/`, or get the software distribution from the course web page.

The relevant file in the distribution is:

- `distSkeleton.py`: a skeleton for implementing the `DDist` class and some handy utilities.

See the last section for information on debugging.

2 Part 1: Basic distributions

We are going to build a simple parameterized set of discrete distributions on integers, and a method for combining them.

Read section 8.5 of the course readings on modeling with distributions.

Step 1. Define a procedure `squareDist`.

```
squareDist(lo, hi, loLimit = None, hiLimit = None)
```

that constructs and returns an instance of `dist.DDist` that assigns the same probability value p to each integer between `lo` and `hi-1`, such that the distribution sums to 1. If `loLimit` is defined, then do not assign any probability to values lower than `loLimit`; instead, assign any probability that should go to a lower value to `loLimit` itself. So, for example, if `lo` and `hi` are both less than `loLimit`, then `loLimit` should be assigned probability 1. Or, for example, if a square distribution is intended to have uniform probability over the range from 1 to 5 (each at 0.2) but a `loLimit` of 3 is provided, then the distribution would have values of 0.2 for elements 4 and 5, but probability of 0.6 for element 3 since the probabilities for elements 1 and 2 are accumulated at this point. Treat `hiLimit` similarly. There are some examples shown below that demonstrate this property. (The motivation for this definition of limits will become clear in subsequent labs.)

Look at the documentation for the function `util.clip`. This should make handling `loLimit` and `hiLimit` much easier.

Step 2. Define a procedure `triangleDist`.

```
triangleDist(peak, halfWidth, loLimit = None, hiLimit = None)
```

where `peak` and `halfWidth` are integers. It should construct and return an instance of `dist.DDist`, which has maximum probability value at `peak` and has linearly decreasing values at each of `halfWidth-1` points on either side of the peak. It should be clipped at `loLimit` and `hiLimit` in a way similar to `squareDist`.

Here are some examples:

```
>>> squareDist(2, 4)
DDist(2: 0.500000, 3: 0.500000)
>>> squareDist(2, 5)
DDist(2: 0.333333, 3: 0.333333, 4: 0.333333)
>>> squareDist(2, 5, 0, 10)
DDist(2: 0.333333, 3: 0.333333, 4: 0.333333)
>>> squareDist(2, 5, 4, 10)
DDist(4: 1.000000)
>>> squareDist(2, 5, 3, 10)
DDist(3: 0.666667, 4: 0.333333)
>>> squareDist(2, 5, 6, 10)
DDist(6: 1.000000)

>>> triangleDist(5, 1)
DDist(5: 1)
>>> triangleDist(5, 2)
DDist(4: 0.250000, 5: 0.500000, 6: 0.250000)
>>> triangleDist(5, 3)
DDist(3: 0.111111, 4: 0.222222, 5: 0.333333, 6: 0.222222, 7: 0.111111)
>>> triangleDist(5, 3, 0, 10)
DDist(3: 0.111111, 4: 0.222222, 5: 0.333333, 6: 0.222222, 7: 0.111111)
>>> triangleDist(5, 3, 3, 10)
DDist(3: 0.111111, 4: 0.222222, 5: 0.333333, 6: 0.222222, 7: 0.111111)
>>> triangleDist(5, 3, 4, 10)
DDist(4: 0.333333, 5: 0.333333, 6: 0.222222, 7: 0.111111)
>>> triangleDist(5, 3, 5, 10)
DDist(5: 0.666667, 6: 0.222222, 7: 0.111111)
>>> triangleDist(5, 3, 6, 10)
DDist(6: 0.888889, 7: 0.111111)
```

3 Part 2: Mixtures of distributions

We can make a new distribution by defining it to be a *mixture* of two existing distributions. By specifying two distributions, `d1` and `d2`, and a mixture probability `p`, the mixture distribution assigns to each element `x`, `p` times the probability of `x` in distribution `d1` plus $(1 - p)$ times the probability of `x` in distribution `d2`.

There are two strategies for implementing this:

- As a new class, `MixtureDist`, whose `__init__` method takes the two distributions and the mixture probability. It needs to provide a method `prob(self, x)`, which returns the probability assigned to `x` in the mixture distribution, and a method `support(self)`, which returns a list of the elements that have non-zero probability in the mixture distribution. Don't write a `__str__` or `__repr__` method.
- As a procedure, `MixtureDist(d1, d2, p)`, which constructs and returns a new instance of `dist.DDist`.

Step 3. Implement `MixtureDist`; you can choose either strategy.

There are plots of several examples of mixtures in the readings. Here are a couple of very simple examples, using an implementation that follows the second strategy above.

```
>>> MixtureDist(squareDist(2, 4), squareDist(10, 12), 0.5)
DDist(11: 0.250000, 2: 0.250000, 3: 0.250000, 10: 0.250000)
>>> MixtureDist(squareDist(2, 4), squareDist(10, 12), 0.9)
DDist(11: 0.050000, 2: 0.450000, 3: 0.450000, 10: 0.050000)
>>> MixtureDist(squareDist(2, 6), squareDist(4, 8), 0.5)
DDist(2: 0.125000, 3: 0.125000, 4: 0.250000, 5: 0.250000, 6: 0.125000, 7: 0.125000)
```

4 Plots for debugging

At the bottom of `distSkeleton.py` there are some instructions and commented-out definitions that will allow you to plot the distributions you are constructing. This can be very helpful, especially for triangles and mixtures. You will need to run this using `idle -n`.

5 What to hand in

Hand in nicely formatted and commented code and a transcript of it working. Show, in the transcript, that you can create distributions of each kind. Try to make your code as simple and concise as possible. It will be graded for style and clarity as well as correctness.

HW 4 Distributions

11/17

dist Skeleton.py file

Read course notes 8.5

- modeling w/ distributions
- mixture distributions
- see other notes

Step 1

Make a square dist

squareDist(lo, hi, loLimit=None, hiLimit=None)

- assigns same value to each integer b/w lo and hi-1
  so it sums to 1
- and then the limits
  - hint: use clip
- seem kinda weird

---

let me do normal 1st

$p = \frac{1}{hi - lo}$

can't use their dict method

need that generate thing

"range"

good - does not include hi in range

(2)

Nice - it works!

Now the hi, lo Limit thing

Clip
- so how to use it

How to dump it all on start

Or assign prob
→ then check and smush

Or can't remove from dict
- ah can - but not when iterating in a loop

Nice got it working
- not particular clean
- no tutor problem to test on
- they will grade for style

And I did not use clip function
- did not see a need for

(3)

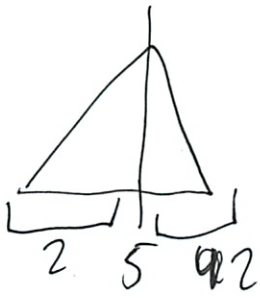
Define triangleDist(peak, halfWidth, loLimit=None, hiLimit=None)

- so limits can be handled same way afterwards
- how to do triangle thing
- so 5,1 = {5: 1}

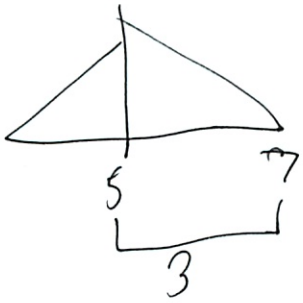
5,2 = {4: .25, 5: 5, 6: .25}

5,3 = {3: .111, 4: .222, 5: .333, 6: .222, 7: .111}

So must = 4 of course



but discretized



each is 1/9

$\frac{1}{2} \cdot 5 \cdot \text{peak} = 1$ (width)

$5 \cdot \text{peak} = 2$

$\text{peak} = \frac{2}{5}$ ⊗ not it

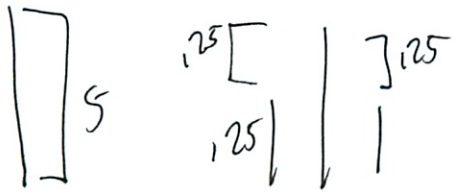
it must add up somehow

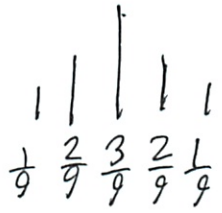
lines are somehow linear

- well they are linear
- should help somewhat

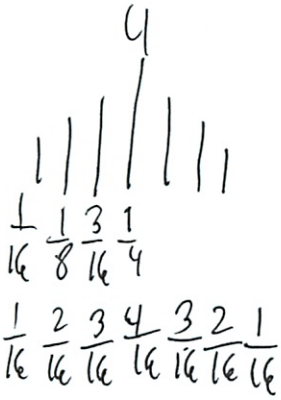
↳ reduction in each even

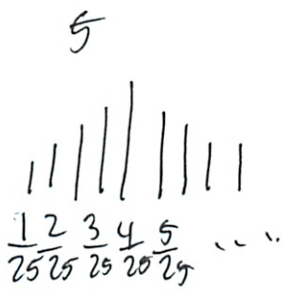
(4)





What would 4 be?
- run lib version





← would not have figured it out w/o this

So denom is halfwidth ^{2}

And then numerator increases to that

range needs to be redone

lo = peak − (halfwidth − 1)

hi = peak + (halfwidth + 1)

And then range(lo, hi + 1)

↑ to include last

Nice - working w/ samples

(5)

Part 2 Mixtures of distributions

- what was this in 6.041 again?
- but the p in front of it is weird
  - actually helps normalize it

New class MixtureDist

--init-- - takes the two prob

prob(self, x)

support(self)

---

mixturedist (d1, d2, p)

2 strategies
← will try

- go for each dist
  - but which is larger?
  - need to look

(what did they think of my style before?)

then what is it for each x value

$$p \cdot d_1 + (1-p)\, d_2$$

nice! got it to work
with the rest being 0s

(6)

Got it to graph
- need to hand in code

Should make code simple + concise
- HW which was code I could not find graded version
- so never got any code comments
- should look for it

The downside to my method is it prints out a bunch of 0 distributions
; should I fix ;
write a cleaner function
which copies

nano 11

11/18

	cola	coffee
5 min	.1	.2
1 hr	.2	.1
Evening	.3	.1

$P(X=\text{cola}) = .1 + .2 + .3 = .6$ ✓ copy error ✗

$P(Y=\text{all evening}) = .3 + .1 = .4$ ✓

$P(X=\text{coffee} \mid Y=\text{all evening}) = \frac{.1}{.4} = .25$ ✓

Part 2

$P(X=\text{coffee}) = .8$

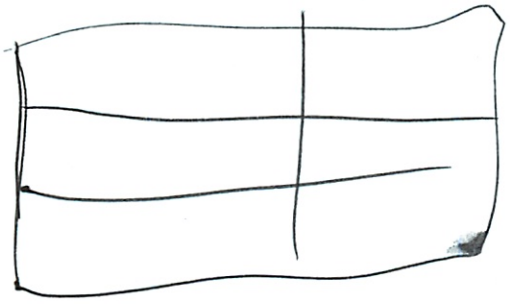
$P(5\text{ min} \mid \text{coffee}) = 1.0$

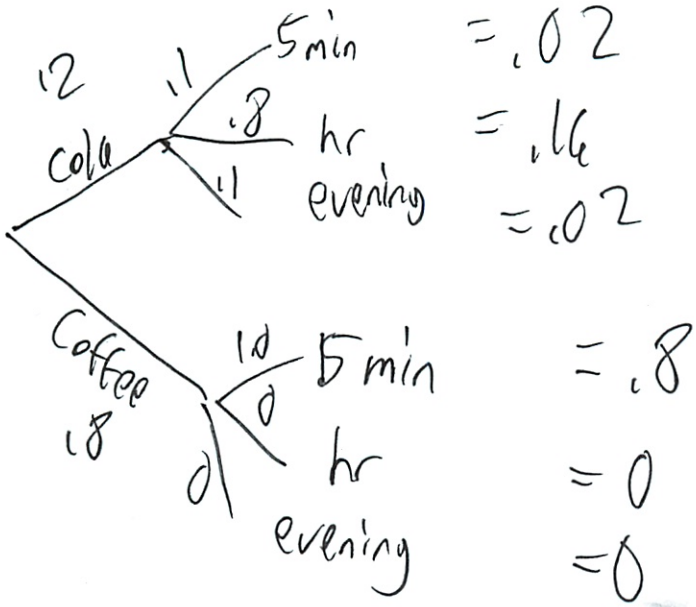
$P(5\text{ min} \mid \text{cola}) = .1$

$P(\text{hr} \mid \text{cola}) = .8$

(2)

need joint





(1)

	cola	coffee
5 min	.02	.8
hr	.16	0
evening	.02	0

all correct - did in 5 min

15 min allocated

Design Lab 11: State Estimation

This lab should be done individually. You can use a lab laptop or a personal laptop. Do `athrun 6.01 update` to get the files for this lab, which will be in `Desktop/6.01/lab11/designLab/`, or get the software distribution from the course web page.

Relevant files include:

- `lib601/ssm.py`: code for the `StochasticSM` class
- `lib601/util.py`: useful utilities
- `dl11Work.py`: place to write and debug code for problems **Wk.11.2.3**

Be sure you have read sections 8.6–8.8 of the readings very carefully.

1 State estimation in the hallway world

When we have a system with internal state that we cannot observe directly, then we can consider the problem of *state estimation*, which is trying to understand something about the internal hidden state of the system based on observations we can make that are related to its state. Examples of such systems include:

- A copy machine, where the hidden state is condition of its internal machinery, the actions we can take are to make copies, and the observations are the quality of the copies.
- A robot moving through a hallway, where the hidden state is the location of the robot, the actions we can take are to move to the east and west, and the observations are colors (which may not always accurately reflect the true underlying colors of the walls).
- A person playing a video game, where the hidden state is which monster the person is trying to kill, there are no explicit actions, and the observations are the moves the person is making.

State estimation is the process of taking in a sequence of inputs we have given to the system (we sometimes call them actions) and observations we have made of the system, and computing a probability distribution over the hidden states of the system. We will build intuition by doing some numerical examples of state estimation by hand. These are types of problems that we will expect you to be able to do in quizzes and exams.

Wk.11.2.1	Do the rest of this problem on state estimation in the hallway world.

Wk.11.2.5 Do this problem now if you feel you need more practice in understanding the state estimation process before writing code to implement it. If not, save it and do it later.

2 Implementing the state estimator efficiently

Now, we will think about how to implement the state estimator algorithmically.

2.1 Stochastic state machines

A state estimator needs to know a *model* of the system whose state it is trying to estimate. This model contains three crucial components:

- initial state distribution
- state transition model
- observation model

You had practice, in software lab, specifying state transition and observation models for the hallway domain.

We will collect these three components together into instances of a Python class, called `ssm.StochasticStateMachine`. It is defined in section 8.6 of the readings: what matters most to us are the three attributes `startDistribution`, `transitionDistribution`, and `observationDistribution`.

Here is an example `ssm.StochasticStateMachine` definition for a hallway world in which there are five rooms with colors white, white, green, white, white. The internal states are the integers 0 through 4 representing which room the robot is really in, the inputs are integers -4 through 4 representing an attempt to move that many squares, and the observations are strings representing colors.

```
def observationModel(s):
    # All of the rooms in our world have white walls, except for room
    # 2, which has green walls;  observations are wrong with
    # probability 0.1
    if s == 2:
        return dist.DDist({'green' : 0.9, 'white' : 0.1})
    else:
        return dist.DDist({'white' : 0.9, 'green' : 0.1})

def transitionModel(action):
    def transGivenA(oldS):
        # Robot moves to the nominal new location (that is, the
        # old location plus the action) with probability 0.8;  some
        # chance that it moves one step too little, or one step too
        # much.  Be careful not to run off the end of the world.
```

```
        nominalNewS = oldS + action
        d = {}
        dist.incrDictEntry(d, util.clip(nominalNewS, 0, 5), 0.8)
        dist.incrDictEntry(d, util.clip(nominalNewS+1, 0, 5), 0.1)
        dist.incrDictEntry(d, util.clip(nominalNewS-1, 0, 5), 0.1)
        return dist.DDist(d)
    return transGivenA

simpleHallway = ssm.StochasticSM(dist.UniformDist(range(5)),
                                 transitionModel,
                                 observationModel)
```

Stochastic state machines have a `transduce` method, which takes a list of inputs and generates a list of outputs. Because they are stochastic (probabilistic), however, several different runs with the same input sequence may generate different output sequences. This is because the starting state, each state transition, and each output is, in general, chosen probabilistically based on the specified distributions.

So, for example, 10 trips down the hallway, each of which is supposed to be made up of four steps to the right, might generate output like this:

```
>>> for i in range(10):
        print simpleHallway.transduce([1, 1, 1, 1])
['white', 'green', 'white', 'green']
['green', 'white', 'white', 'white']
['white', 'green', 'white', 'white']
['white', 'green', 'white', 'white']
['white', 'green', 'white', 'white']
['white', 'white', 'white', 'white']
['white', 'green', 'green', 'white']
['white', 'green', 'white', 'white']
['white', 'white', 'white', 'white']
['white', 'white', 'white', 'white']
```

Wk.11.2.2 Do this tutor problem to get practice with the types of the components of a stochastic state machine.

2.2 State Estimation

Now, we're going to take the cool-but-potentially-confusing step of making the state estimator. A state estimator is, itself, an instance of `sm.SM`: it is a regular deterministic state machine. Its job is to take inputs that are pairs, `(o, i)`, where `o` is the observation that we made of some system at some time `t` and `i` is the input that was given to the system at time `t`. The internal state of the state estimator is a *belief state*, or probability distribution over the possible hidden states of the system whose internal state we are trying to estimate. The output of the state estimator will be the same as its internal state.

To make an instance of a state estimator, we pass in a model of the system whose state we are trying to estimate, in the form of a stochastic state machine. So, for example, if we wanted to try to estimate the robot's position in the simple hallway described above, we could make a state estimator:

```
hallwayLocalizer = se.StateEstimator(simpleHallway)
```

Now, `hallwayLocalizer` is a regular old state machine, and we can use familiar methods, like `transduce` on it. Here, we feed in a sequence of observation-action pairs, asking what we believe about the robot's location after it observes white, moves to the right one square, observes white again, moves to the right one square, observes green, and then stays where it is:

```
>>> hallwayLocalizer.transduce([('white', 1), ('white', 1), ('green', 0)])
[DDist(0: 0.024324, 1: 0.218919, 2: 0.221622, 3: 0.070270, 4: 0.464865),
 DDist(0: 0.003029, 1: 0.051496, 2: 0.224196, 3: 0.060546, 4: 0.660733),
 DDist(0: 0.002819, 1: 0.087084, 2: 0.581842, 3: 0.113220, 4: 0.215035)]
```

The result of `transduce` is a list of three belief states.

> *Check Yourself 1.* Be sure the results of transduce shown above make sense to you. If they don't, ask a staff member for clarification.

As we observed in section 8.8.1 of the course readings, the state estimation procedure can be seen as taking two steps:

1. Doing a step of Bayesian evidence updating, in which we update the current belief state based on the actual observation received, using the observation distribution stored in the `ssm.StochasticSM` that is the model of the underlying system.
2. Then, using that resulting intermediate belief, doing an update using the law of total probability to apply the transition distribution appropriate for the input that was given to the underlying system.

Given the library of methods we have developed for operating on distributions, this is very straightforward to implement in Python. The starting state of the state estimator is just the initial state distribution of the SSM. To get the next values (that is, the next belief state of the state estimator), we do a step of Bayes evidence with the observation and an application of the law of total probability with a transition distribution that depends on the input.

```
class StateEstimator(sm.SM):
    def __init__(self, model):
        self.model = model
        self.startState = model.startDistribution
```

```
def getNextValues(self, state, inp):
    (o, i) = inp
    sGo = dist.bayesEvidence(state, self.model.observationDistribution, o)
    dSPrime = dist.totalProbability(sGo,
                                    self.model.transitionDistribution(i))
    return (dSPrime, dSPrime)
```

This code works fine, and is conceptually very simple, but each step is fairly inefficent. Our implementation of the `bayesEvidence` method (in a tutor problem from software lab) actually creates a whole joint distribution between the state and observation, and then conditions on the observation, selecting a single row of the joint. We can make this more efficient by more closely mirroring the procedure we showed graphically in the lecture and the readings: multiplying the probability of each state s in the belief state by $\Pr(o \mid s)$, and then normalizing it.

In a similar way, our implementation of `totalProbability` (also in a tutor problem from software lab) is also quite inefficient. It constructs the entire joint distribution between `s` (the state at time t) and `sPrime` (the state at time $t + 1$), and then immediately marginalizes out `s`. Again, we can more closely emulate the procedure shown graphically in the lecture and the readings: making a new empty distribution over `sPrime`, then iterating over the states `s`, and adding, to each state `sPrime` the probability of being in state `s` times the probability of making a transition to state `sPrime`, given the input.

Your job is to implement the `StateEstimator.getNextValues` method. We would encourage you to write two internal helper procedures, one for Bayes evidence, and one for total probability, that have the same results as our old implementations, but which are more efficient.

Rules:

- Never access the dictionary of a distribution directly (use the `prob` method instead).
- Do not modify any of the distributions in the model. If you want a copy of a distribution, you can use the `dictCopy` method of `DDist` copy the dictionary, and then use it to make a new `DDist`.
- Do no make any assumptions about the states, observations and actions. You do not need to assume that they are integers or strings or anything else in particular. Hint: use the `support` method of the `DDist` class.
- Inside the tutor problem and the work file, the module `dist` has been imported; if you need to use functions or classes from there, you can access them with expressions like `dist.x`.
- The tutor will test the number of times you access the input probability distributions, and if your code is too inefficient (it makes too many accesses to the dictionaries) it will fail the check even if it generates the correct next belief.

  When you check your code in the tutor, the results of the test cases show the number of calls to the `prob` method of `dist.DDist`, followed by the actual result of state estimation. You don't have to have precisely the same number of calls as our solution, but it can't be too many more.

The tutor will test your code on the copy machine example from the readings:

```
transitionTable =
   {'good': dist.DDist({'good' : 0.7, 'bad' : 0.3}),
    'bad' : dist.DDist({'good' : 0.1, 'bad' : 0.9})}
observationTable =
   {'good': dist.DDist({'perfect' : 0.8, 'smudged' : 0.1, 'black' : 0.1}),
    'bad': dist.DDist({'perfect' : 0.1, 'smudged' : 0.7, 'black' : 0.2})}
copyMachine =
 ssm.StochasticSM(dist.DDist({'good' : 0.9, 'bad' : 0.1}),
                  lambda i: lambda s: transitionTable[s],
                  lambda s: observationTable[s])
obs = [('perfect', 'step'), ('smudged', 'step'), ('perfect', 'step')]
```

You can use these values for testing, but don't build any of them into your code. Your code should work for any stochastic state machine (the `model` argument). This example is provided only so you can understand the test cases. The variable `obs` is set to a sequence of observation, action pairs. In this copy-machine example, the action is ignored, but it has to be present in order to make all of the types match up correctly.

Wk.11.2.3 Do this tutor problem on how to implement the state estimator efficiently.

Checkoff 1. Explain your solution to a staff member.

desLab 11

11/18

State estimation

- internal state (on markov chain)
  ↑ can't be seen
- states of Copy machine
- location of robot

Tutor Problem 11.2.1

- 5 rooms
- W W 6 W W
  0 1 2 3 4) rooms
- don't know initial state, all equally
- certain motion + sensing
- S_t = state at time t
- I_t = Action at t
- O_t = Observation at t

$$\sum_{s \in D_S} P(S_0 = s) = 1.0$$

- enter decimals

(2)

a) Prior prob of each room $B_0(s) = P(S_0 = s)$

.2 .2 .2 .2 .2 (✓) for each s

b) Robot makes observation $O_0 = \text{white}$

$P(O_0 = \text{white} \mid S_0 = s)$

.25 (✗) .25 (✗) 0 (✓) .25 (✗) .25 (✗)

b2) $P(O_0 = \text{white} \mid S_0 = s)\, P(S_0 = s)$

note same as $P(O_0 = \text{white}, S_0 = s)$ for each s

S_0 .25 · .2 = .05

.05 (✗) .05 (✗) 0 (✓) .05 (✗) .05 (✗)

b1 again) Oh was I jumping ahead

$P(O_0 = \text{white} \mid S_0 = s) = \frac{1}{2}$ (✗) since rooms can be green or white - but not evenly distributed - must not add to 1

(3)

b1) The robot knows what the state is supposed to be
- like having a map

So $P(\text{observe white} \mid \text{room is white}) = 1$

1 1 0 1 1 ✓

b2) Now 1 .2

.2 .2 0 .2 .2 ✓

b3) p (white)

- So marginal sum

.8 ✓

b4) $B_0(s) = P(S_0 = s \mid O_0 = \text{white})$

so now is this what is prob I hum in each state given room is white

.25 .25 0 .25 .25 ✓

(so I jumped ahead too far)

- could do w/ Bayes:

$$P(S = s \mid O = \text{white}) = \frac{P(O = \text{white} \mid S = s)\ P(S = s)}{P(O = \text{white})} = \frac{1 \cdot .2}{.8}$$

$= .25$ ✓

(So I did it all in my head - too fast)

(4)

c) Told robot to go $\xrightarrow{1}$ $[I_0 = 1]$ what is belief state
- transition perfect

$B_1(s) = P(s_1 = s \mid O_0 = \text{white}, I_0 = 1)$

"just shift prob to right + smush"
"ah that is why hi, lo limit

0 .25 .25 0 .5

d) Now assume robot observes white again

$B_1'(s) = P(s_1 = s \mid O_0 = \text{white}, I_0 = 1, O_1 = \text{white})$

- So this means we are not in 2

0 .25 0 0 .5

.75 ← sum of what is left ← normalize

0 1/3 0 0 2/3

e) Told robot to go $\xrightarrow{1}$ again $[I_1 = 1]$

$B_2(s) = P(s_2 = s \mid O_0 = \text{white}, I_0 = 1, O_1 = \text{white}, I_1 = 1)$

0 0 1/3 0 2/3

(5)

f) What action could robot do to make location unambiguous?

1, -1, any action, none

1 would ✓ ⇐ ✗

-1 would not ✗

any since -1 no ✗

none since 1 ✗

oh after observation O_2

- then any action since after O_2 it knows for sure ✓

g) What is belief state if O_2 = white and $I_2 = 1$

$P(S_3 = s \mid O_0 = \text{white}, I_0 = 1, O_1 = \text{white}, I_1 = 1, O_2 = \text{white}, I_2 = 1)$

So if O_2 = white then we know in #4

then it tries to move → again, still in #4

0 0 0 0 1

g2) What states could robot have started in for this to be possible

1, all

- assumes if you think hitting the wall is possible

TA says yes

0, 1, 2, 3, 4 ✗

#6

also do they mean this last qu - or the whole sequence?
- oh it needs to go white → white → white

0 white → white → green ✗
1 white → green → white ✗
2 green → white → white ✗
3 white → white → hit white ✓
4 white → hit white → hit white ✓

) guessing must count hits ✓

b) Belief state if O_2 = green $I_2 = 1$

$P(S_3 = s \mid O_0 = \text{white}, I_0 = 1, O_1 = \text{white}, I_1 = 1, O_2 = \text{green}, I_2 = 1)$

- so it must be in 3
- was in 2 and went ↑

0 0 0 1 0 ✓

b2) What states could it have started in
- will only be 1

see above

0 ✓

(1)

Part 2 ~~Noisy~~ Noisy sensor, perfect action

- still same hallway
- 5 possible color sensor outputs
  - black
  - white
  - red
  - green
  - blue

P(color observed) = { ~~nominal~~ nominal/correct color: .8, others: .05 each }

- don't know inital, all = ly likely
- perfect motion

1. Robots prior belief $B_0(s) = P(S_0 = s)$

.2 .2 .2 .2 .2 ✓

2. What is distribution of what robot sees? $P(O_0 = 0)$
   - ? is it .2 since does not know room yet

.2 .2 .2 .2 .2

✗ ✗ ✗ ✓ ✗

green

- or again knowing what hall way is like - and dist of colors

0 .8 0 .2 0 ✗

black white red green blue

✗ ✗ ✗ ✓ ✗

(8) TA help

First room

know white

P(correct) = .8

Need to do ~~X~~ multiply P(are in R 0) = .2

.8 · .2 = .16

add up for 4 rooms = .64

P(green | first room) = .05
× P(first room) = .2
.01

Middle room

green) .8 · .2 = .16

white .05 · .2 = .01

So total prob ~~for~~ for white = ~~.65~~ .8 · .2 · 4 + .05 · .2
= .65

total prob green = .8 · .2 · 1 + .05 · .2 · 4 = .2

P(red in any room) = .05 · P(any room) = 1

.05 · 1

total red = (.05) ~~· 5~~

Or P(red | room 1) = .05 · P(room 1) = .2

.01

then total red = .01 · 5 = .05

(9)

Same by symmetry for other

Check – should add to 1

$.65 + .2 + .05 \cdot 3 = 1$ ✓

.05	.65	.05	.2	.05
black	white	green	red	blue

✓

(I think I was thinking calculations very very simple like other ones)

c) ~~Robot~~ Robot makes an observation → white

$B_0'(s) = P(S_1 = s \mid O_0 = \text{white})$ ~~[illegible]~~

~~now~~ it is getting more complex!

think about conditioning $\dfrac{P(S = s \cap O_0 = \text{white})}{P(O_0 = \text{white})}$

$P(\text{white} \mid s_0) \cdot P(s_0) = P(\text{white}, s_0)$

so for s_0 $\dfrac{.8 \cdot .2}{.65 \leftarrow P(O_0 = \text{white})} = \dfrac{16}{65}$ ✓

$S_1 = \dfrac{.8 \cdot .2}{.65} = \dfrac{16}{65} = S_3 = S_4$

$S_2 = \dfrac{.05 \cdot .2}{.65} = \dfrac{1}{65}$

(10)

d) Told robot goes $\xrightarrow{1}$ $[I_0 = 1]$

$B_1(s) = P(S_1 = s \mid O_0 = \text{white}, I_0 = 1)$

- remember movement is perfect
- so just shift

$0 \quad \frac{16}{65} \quad \frac{16}{65} \quad \frac{1}{65} \quad \frac{32}{65}$ ✓

e) Now what is distribution robot sees. $P(O_1 = 0)$

- it should be the same
- not conditioned on past

.05 ✓ .65 ✗ .05 ✓ .2 ✗ .05 ✓

TA: you found a bug, should be conditioned

Prof: Its implicitly conditioned, not writing out chain each time

So ~~P~~ P(first room = 0, skip

~~P~~ 2nd room (1)

white $P(\text{white} \mid \text{2nd room})\, P(\text{2nd room})$

$.8 \quad \frac{16}{65} = \frac{64}{325}$

green $P(\text{green} \mid \text{2nd room})\, P(\text{2nd room})$

$.05 \cdot \frac{16}{65} = \frac{4}{325}$

(11)

3rd room(2)

	P(white / 3rd room)	P(3rd room)	
white	.05	$\frac{16}{65}$	$= \frac{4}{325}$
green	"		$\frac{64}{325}$

4th room(3)

white	.8 ·	$\frac{1}{65}$	$= \frac{4}{325}$
green	.05 ·	$\frac{1}{65}$	$= \frac{1}{1300}$

5th room(4)

white	.8 ·	$\frac{32}{65}$	$= \frac{128}{325}$
green	.05 ·	$\frac{32}{65}$	$= \frac{8}{325}$

(skipping red, black, blue — all the same)

Total

white	$\frac{64}{325}$	$+ \frac{4}{325}$	$+ \frac{4}{325}$	$+ \frac{128}{325}$	$= \frac{8}{13}$	✓
green	$\frac{4}{325}$	$+ \frac{64}{325}$	$+ \frac{1}{1300}$	$+ \frac{8}{325}$	$= \frac{61}{260}$	✓

red, black, blue ~~.01 + .01 + 2~~ .05

(12)

e) Sees white (O_1 = white)

$B_1'(s) = P(S_1 = s | O_0 = \text{white}, I_0 = 1, O_1 = \text{white})$

- So base on last one

0	$\frac{64}{325}$	$\frac{4}{325}$	$\frac{4}{325}$	$\frac{128}{325}$
✓	✗	✗	✗	✗

or not → needs to add to 1

look back at C

$$\alpha S_1 = \frac{.8 \cdot \frac{16}{65}}{8/13} \leftarrow P(O_1 = \text{white})$$

(P(white|si) P(s))

← oh so just forgot to normalize

0	$\frac{8}{25}$	$\frac{1}{50}$	$\frac{1}{50}$	$\frac{16}{25}$
				✓

g) Robot goes → (perfect transition) $[I_1 = 1]$

remember

0	0	$\frac{8}{25}$	$\frac{1}{50}$	$\frac{33}{50}$
				✓

(13)

h) Now instead of f, g problems it sees green and goes right

- so first just green

$$\frac{P(\text{green}|s)\,P(s)}{P(\text{green})} \leftarrow \text{normalize}$$

0	$\frac{\frac{4}{325}}{\frac{61}{260}}$	$\frac{\frac{64}{325}}{\frac{61}{260}}$	$\frac{\frac{1}{1300}}{\frac{61}{260}}$	$\frac{\frac{8}{325}}{\frac{61}{260}}$
0	$\frac{16}{305}$	$\frac{256}{305}$	$\frac{1}{305}$	$\frac{32}{305}$

- now $\Rightarrow$

0	0	$\frac{16}{305}$	$\frac{256}{305}$	$\frac{33}{305}$ ✓

Done w/ this tutor problem - was 89 points

Part 2 State estimator Implementation

- 3 parts
  - inital state dist
  - state transition model
  - observation model

} did in SwLab 11

Stocastic = probabilistic

- Combine into ssm, Stochastic State Machine

(14)

- defines the 3 distributions
- they give the models on the paper
- transduce method

Tutor 11.2.2

Part 1 Distributions

P(Disease = disease) #

P(Disease) DDist

P(Disease = disease, Test = posTest) #

P(Disease | Test) Proc

P(Disease | Test = posTest) DDist

wow surprised all right

Part 2 SSM

m = instance of a Stocastic State Machine

s, x, y = states

i = input

o = output

(15)

m. start Distribution — DDist ✓

initial Dist, I believe ✓

m. startDistribution. prob(s) # ✓

m. obs Dist — ~~DDist~~ Proc ✓

m. obs Dist (s) — DDist ✓

m. obs Dist (s). prob(o) # ✓

m. obs Dist. prob — error ✓

m. tranitionDist (i) — ~~DDist~~ proc ✓

m. trans Dist (i)(y) — ~~error~~ dist ✓

m. trans Dist (i)(y). prob(x) # ✓

Write python express value $P(O_t \mid S_t = s)$

dist. DDist ({

what the hell?

TA answer from above

So ✓

And the second one $P(S_{t+1} = x \mid S_t = y, I_t = i)$

tutor problem done

(16)

2.2 State Estimation

- "Cool but confusing"
- instance of SM, SM
  - input (o, i)
    - o: observation
    - i: input
  - internal state is the belief state
    - prob dist of possible hidden states
  - output = internal state
- to make pass in model of system
- inside:
  1. Bayesian evidence updating
     - update belief state w/ estimate
  2. Then using resulting intermediate belief do law of total probability to apply transition dist for the input (oh what we just did) Normalize
- use libraries from operating on dists

(17)

So they give State Estimator class
- and get next value
- code very inefficient
- can make more efficient in method shown in lecture readings
~~readings~~
- multiply prob each state s by P(O|s) + normalize

$$\frac{P(s)P(O|s)}{P(O)}$$

- TotalProb also inefficient
- instead make new empty dist over Sprime, iterate over states S, adding to each state Sprime the prob P(being in s) · P(transition to sprime | s)
- it counts calls to DDist, prob
- first eff use my Bayes + TPT

Sample data is now multidimension list - not function

11/21

I do not get what in all world they want us at home to do!

Graphically - how did we do it
- that may be better than trying to convert my old code

(18)

What are we trying to do
- estimate hallway
- like the lab
- look at lab again

- We never did ~~perfect~~ noisy sensor, noisy action - but ok
- starting state = .2 .2 .2 .2 .2
  - ↳ given in model.startDist
  - but is it given in our example?
  - I'm guessing it's

    Good .9 Bad .1

  - yeah we went through this in the reading
- then transition + observation models given
  - like table on lecture 11 pg 6 slide 4
- Step 1 - build Joint table

S	O	perfect	smudged	bleach
	good	.72	.09	.09
	bad	.01	.07	.02

↑ but what do they want us to do
- not hard code table

(19)

Build only the part we need? ←

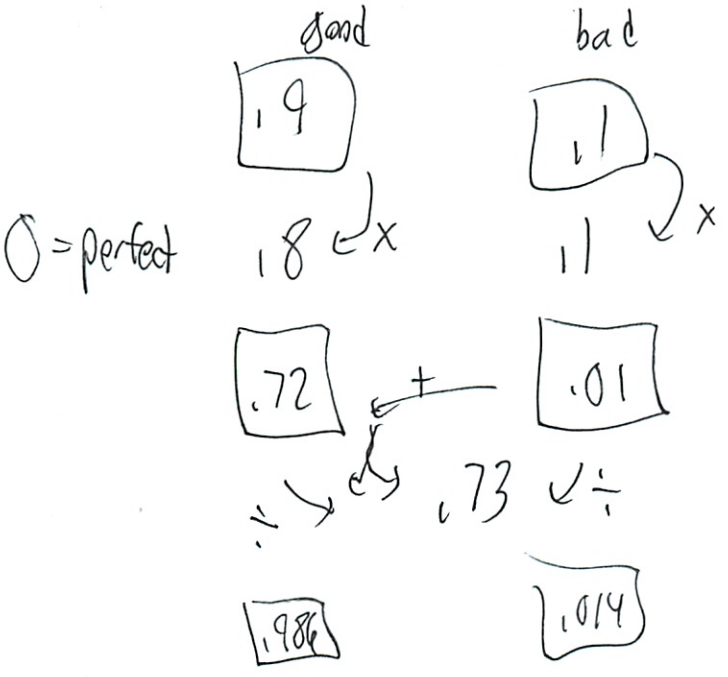
Build + cache

-so taking actual obs o

find $\frac{good \wedge o}{\delta}$ $\frac{bad \wedge o}{0}$

$\delta = good \wedge o + bad \wedge o$

Lets do table



init state {good .9, bad .1}

Observation table { 'good' { perfect .8 }
'bad' { perfect .1 }

~~calculate~~
multiply
add
divide
return

What do we observe - perfect?

- yeah not good, bad

So is state P_{BgA} then - no that is obs?

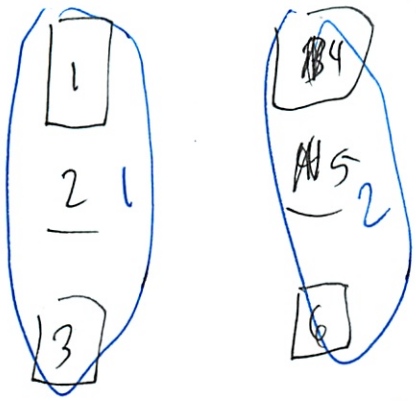
PA = inital b = the actual obs

- so why are they changing parameter order? ?

- oh well
- that was very confusing at 1st

29

How to build it in code?



← for each state, support

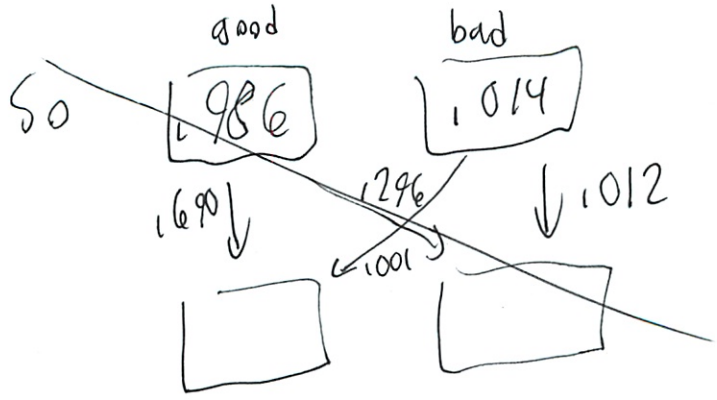
7 ← can do w/ previous

Cool - that is working

Step 2: total prob (new slides p 7)

build this joint dist

		S1	
		good	bad
So	good	.690	.296
	bad	.001	.012

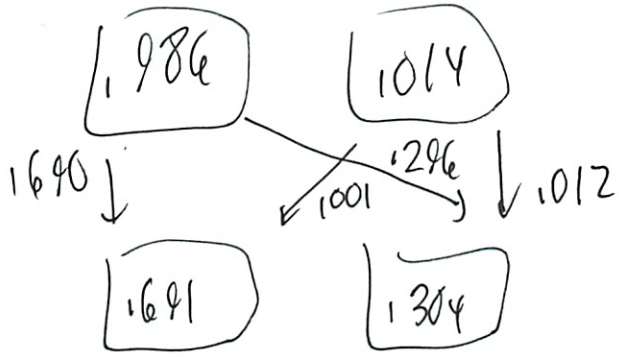


need to marginalize out 1st

(21)

	s_1 good	s_1 bad
s_0 good	.690 ↓+	.296 ↓+
s_0 bad	.001	.012
	.691	.308

No! that was in the table!



add both of the arrows
- return that

but how is the table made?

$P(s_1 | s_0) \; P(s_0 | O_0 = \text{perfet})$

transition model — last ans

So for s_0 good → s_1 good last good

.7 · .98 = .6888 ⟩ + good .691

bad → good · last bad

.1 · .013 = .0013

(22)

Ok - get what doing - now how to build?
- will need all 4

So I think do the above

for ~~each~~ ~~SBB~~ stateItem in sgd.support() (a)
- or do we know it will be two?
- no don't think

for each again (b)

trans model
1b → 1a, proper value) add
2b → 1a

---

transmodel
1b → 2a
2b → 2a, proper value) add

return

---

again parameters flipped

just care about sum

So a = 51
b = 50

(23)

They pre specify the I in transition Model
- only that one used

Based on what the input is

Ah input irrelevant here → always "step"

but it still gives wrong :(

- I commented it part out
- hopefully they don't test when input matters

Now getting dict object not callable

Oh duh [] brackets not ()

Ur why so many answers - should just be 1

Is running 3 times when 1 works

- Oh duh its iterating 3 times as directed
- first time that happened

Check

- Oh last # is the prob call

36 vs 32
66 vs 52
84 vs 72

should be Ok - 4 more than supposed
- guess can cache

(24)

Cache state 0, prob

Meh - did not do anything to ↓ count
(it should have by 2)

Oh well revert + submit

This was fun

Took an hr at home
- really concentrating
- I like these puzzles!

Oh one more tutor problem 11.2.5
- thought handout said optional
- will do later + print the Graphs

Oh here is noisy noisy case

~~Red~~ White

Same hallway white white green white white
0 1 2 3 4

black
white
red
green
blue

possibly colors .8 correct .05 each incorrect

(25)

Action noise

.8 correct
.1 one to left
.1 to right
w/ clipping

S_+, I_+, O_A $\sum_{s \in D} P(S_0 = s) = 1.0$

---

a. Prior belief

$B_0(s) = P(S_0 = s)$

.2 .2 .2 .2 .2 ✓

b. Sees white
- go back + look

$\frac{16}{65}$ $\frac{16}{65}$ $\frac{1}{65}$ $\frac{16}{65}$ $\frac{16}{65}$ ✓

c. Goes I

0 $\frac{16}{65}$ $\frac{16}{65}$ $\frac{1}{65}$ $\frac{32}{65}$

Oh right noisy action!

(26)

So lets look at transition model

- I should run it through script I make earlier

	0	$\frac{16}{65}$	$\frac{16}{65}$	$\frac{1}{65}$	$\frac{32}{65}$
.9 ↓	.1	.1 ↓ .8	.1	↓ .8	etc	↓ .9
+	0	0	$\frac{8}{365}$	$\frac{8}{365}$	$\frac{1}{650}$
	$\frac{8}{325}$	$\frac{64}{325}$	$\frac{64}{325}$	$\frac{4}{325}$	$\frac{144}{325}$
		$\frac{8}{365}$	$\frac{1}{650}$	$\frac{16}{365}$	
Sum	$\frac{8}{325}$	$\frac{72}{325}$	$\frac{29}{130}$	$\frac{28}{325}$	$\frac{289}{650}$ ✓

---

d. Now Robot sees white again

$$B'(s) = P(S_1 = s \mid O_0 = \text{white}, E_0 = 1, O_1 = \text{white})$$

$$\frac{P(\text{white} \mid s)\, P(s)}{P(\text{white})}$$

So for S_0 $\quad \frac{.8 \cdot \frac{8}{325}}{.165} = \frac{128}{4225}$ ✓ excel would help on this

(27)

S_1 $\dfrac{.8 \cdot \frac{72}{325}}{.65} = \dfrac{1152}{4225}$ ✗

S_2 $\dfrac{.2 \cdot \frac{29}{130}}{.65} = \dfrac{58}{845}$ ✗

S_3 $\dfrac{.8 \cdot \frac{28}{325}}{.65} = \dfrac{448}{4225}$ ✓

S_4 $\dfrac{.8 \cdot \frac{289}{~~322~~ 650}}{.65} = \dfrac{2312}{4225}$ ✗

Why 3 wrong? I did the same process on each...
???
Very weird
I went back + checked math
Try in software

.10319 .12871 ✗ .10055 ✗ .10006 .15748 ✗

Ur not close enough

(28)

But what in all world is wrong?

S_2 should have been

$$S_2 \quad \frac{105 \cdot \frac{29}{130}}{165} = \frac{29}{1690}$$ ✓

- or same issue on p12

need to normalizie - add up the above last row

$$\frac{8}{325} + \frac{72}{325} + \frac{29}{130} + \frac{28}{325} + \frac{289}{650} = \frac{939}{650}$$

; should it be 1?

bad idea ✓

; why did the others work then?

$$S_1 \quad \frac{18 \cdot \frac{72}{325}}{\frac{939}{650}} = \frac{192}{1565}$$ ✗

$$S_2 \quad \frac{105 \cdot \frac{29}{130}}{\frac{939}{650}} = \frac{29}{3756}$$ ✗ above S_2 was actually correct

$$S_4 = \frac{18 \frac{289}{650}}{\frac{939}{650}} = \frac{1156}{4685}$$ ✗

29

So that's not it or I added wrong

But denom is right - must be for the others

- the prob of white

Why S_1, S_2. Unless .8 is wrong there - no its white

Or (parenthse error)

S_1 $\frac{1152}{4795}$ ← (denom copy error) (X) nope

S_3 $\frac{448}{4745}$ ← though ran S_3 and denom wrong too (X)

- or was big enough margin

was actually correct w/ $\frac{448}{4225}$

S_4 $\frac{2312}{4225}$ ← here that same denom is back (X)

I don't get it!

Mail it in + do something else

Reply basically said to recalculate P(white)

11/22

Did I do it wrong last time?

- see p11

(30)

$P(\text{white} \mid s)\, P(s)$

So $\Sigma\ .8 \cdot \frac{8}{325} + .8 \cdot \frac{72}{325} + .05 \cdot \frac{24}{130} + .8 \cdot \frac{28}{325}$

$+ .8 \cdot \frac{289}{650}$

$= .632 = \frac{329}{520}$

Why did I not do that before?

- ~~$\frac{939}{650}$~~ was wrong → that should have been 1
- and then I would have seen needs .8, .05 $P(\text{white} \mid s)$
- and close enough for overlap

$S_1 \quad \frac{.8 \cdot \frac{72}{325}}{\frac{324}{520}} = \frac{2304}{8225}$ ✓

$S_4 \quad \frac{.8 \cdot \frac{289}{650}}{\frac{324}{520}} = \frac{4624}{8225}$ ✓

(31)

The others were wrong - but within margin of error - redo for next one

$$S_0 \quad \frac{.8 \cdot \frac{8}{325}}{\frac{939}{650}} = .0311 = \frac{256}{8225}$$ ✓

$$S_2 \quad \frac{.05 \cdot \frac{29}{130}}{\frac{939}{650}} = \frac{29}{1645} = .0176$$ ✓

$$S_3 \quad \frac{.8 \cdot \frac{28}{325}}{\frac{939}{650}} = .1089 = \frac{128}{1175}$$ ✓

Now these should all add to 1

e) Now $\rightarrow$

$$B_2(s) = P(S_2 = s \mid O_0 = \text{white}, t_0 = 1, O_1 = \text{white}, t_1 = 1)$$

33

$\frac{256}{8225}$ $\frac{2304}{8225}$ $\frac{29}{1645}$ $\frac{128}{1175}$ $\frac{4624}{8225}$

↓19 ↓18

+ { 0 / $\frac{1152}{41125}$ / 1152/41125

$\frac{2304}{41125}$ ✗

$\frac{128}{41125}$ ~~1152/41125~~ $\frac{9216}{41125}$ $\frac{29}{16450}$ $\frac{18833}{82250}$ ✗

$\frac{1152}{41125}$ $\frac{116}{8225}$ $\frac{464}{5875}$ $\frac{436}{8225}$ ✗

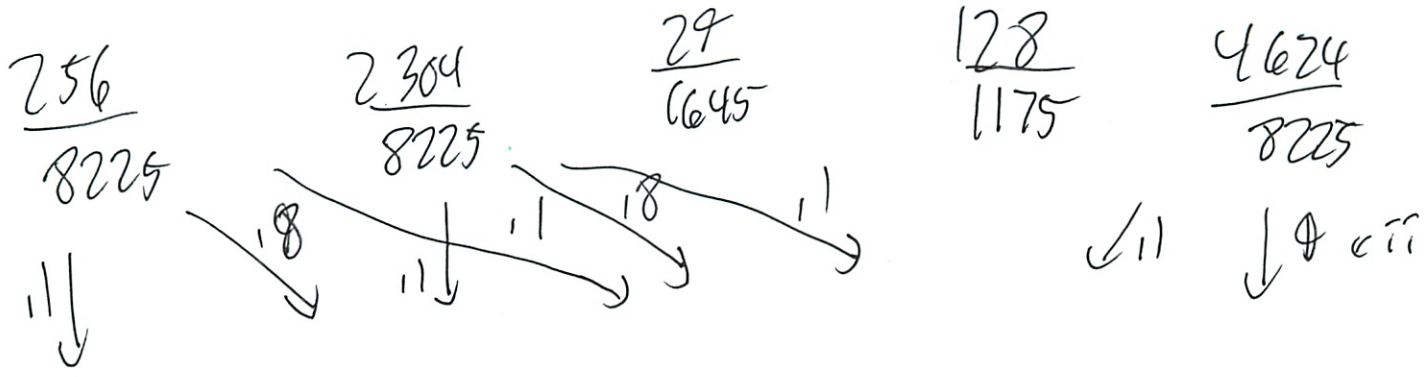
$\frac{29}{16450}$ $\frac{512}{5875}$ $\frac{2312}{41125}$ $\frac{11937}{82250}$ ✗

~~5872~~ 64 / 5875 $\frac{20808}{41125}$ $\frac{21256}{41125}$ ✗

should add to 1

Oh damn → I did for 0 before I moved it over 1st

(33)



128/41125	1024/41125	128/41125	1152/41125	29/16450
	1152/41125	9216/41125	116/8225	64/5875
		29/16450	64/5875	512/5875
			2312/41125	~~4624/8225~~ does not belong on right split only .9
				20808/41125

128/41125 ✓	2176/41125 ✓	18833/82250 ✓	4492/41125 ✓	36013/16450 ✗ ? more than 1

Tihmm or add up rest 1 − that

$$1 - \frac{1297}{3290} = \frac{1993}{3290}$$ ✓

Doe!

(64)

- Checkoff 1

So if a state was 0 → not in state.support()

how to deal w/ it? * consider all possible cases

Plus I removed the (i)

Should have used Tran model → not in code here

↳ returns a procedure

- takes a state
- returns a DDlist

6.01: Introduction to EECS I

Search Algorithms

Week 12 *November 23, 2010*

Reading: 9.1 – 9.4

Uncertainty
- when have different options
- what to do?

→ plan ahead
- if I do these actions, will I get there?

Looking Ahead

- **Reaction:** Use a rule to determine the 'action' to take, as a direct function of the state
  - wall-following
  - proportional controller
- **Planning:** Choose action based on 'looking ahead': exploring alternative sequences of actions

Methods for planning require us to specify an explicit model of the effects of our actions in the world.

need explicit model of world

Using models to choose actions

Assume states and actions are discrete.

discrete
- location
- action

Given
- A state-machine model of the world
- A start state
- A goal test

state machine

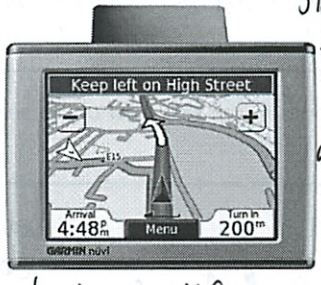
Find a sequence of actions (inputs to the state machine) to reach a goal state from the start state.

- **Primitives:** single actions
- **Combination:** sequencing actions
- **Abstraction:** path
- **Common pattern:** Just describe the goal, and trust an algorithm to automatically compute how to get there.

PCAP

Application: Navigation

What are good definitions of states, actions?



State?
- roads
- intersections

actions
- roads
- turns

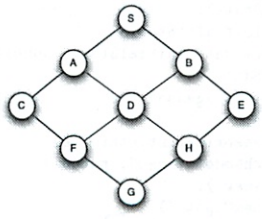
What's the best path?

What makes a path good?

how measure the path?

describe world as set of states

Abstraction: Labeled graph



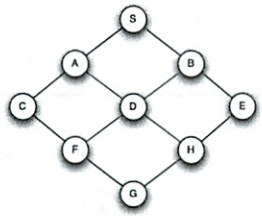
Lots of possible paths! Could enumerate them and evaluate each one. Too hard...

Assume *additive cost*. For now, each segment has a cost of one. We want to find the shortest path.

don't just brute force search every past (British Museum Algorithm)

Formal model

- States: {'S', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'}.
- Starting state is 'S'.
- Goal test: `lambda x: x == 'G'`
- Legal actions and successors:




```
map1 = {'S' : ['A', 'B'],
        'A' : ['S', 'C', 'D'],
        'B' : ['S', 'D', 'E'],
        'C' : ['A', 'F'],
        'D' : ['A', 'B', 'F', 'H'],
        'E' : ['B', 'H'],
        'F' : ['C', 'D', 'G'],
        'H' : ['D', 'E', 'G'],
        'G' : ['F', 'H']}
map1LegalActions = [0, 1, 2, 3]
```

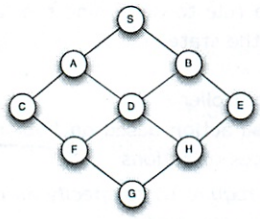
since max 4 options

```
def map1successors(s, a):
    if a < len(map1[s]): return map1[s][a]
    else: return s
```

if action not legal stays in current state

capture states + transitions

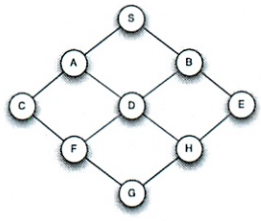
Check yourself



How many "shortest" paths are there from S to G?

6 <7 ≤ 3 ≤ 2

Check yourself



How many "shortest" paths are there from S to G?

6

A numeric example

- States: integers
- Start state: 1
- Legal actions (and successors) in state n: $\{2n, n+1, n-1, n^2, -n\}$
- Goal test: $x = 10$

What are the nodes in the first level of the search tree in this domain?

start n = 1

2, 2, 0, 1, -1

branching factor could be bigger than 2

A numeric example

- States: integers
- Start state: 1
- Legal actions (and successors) in state n: $\{2n, n+1, n-1, n^2, -n\}$
- Goal test: $x = 10$

How long is the longest path in this domain?

∞

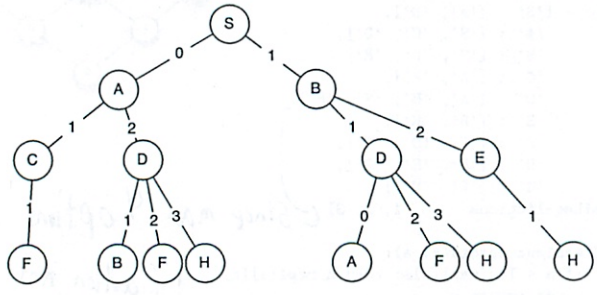
- nothing that says you will stop

going to mix notation

missing code slide → init + path, search seperate paper

Don't be totally stupid!

Pruning Rule 1. Don't consider any path that visits the same state twice.



clips a lot out of tree

Not being totally stupid, in Python

```
def search(initialState, goalTest, actions, successor):
    if goalTest(initialState):
        return [(None, initialState)]
    agenda = [SearchNode(None, initialState, None)]
    while not empty(agenda):
        parent = getElement(agenda)
        for a in actions:
            newS = successor(parent.state, a)
            newN = SearchNode(a, newS, parent)
            if goalTest(newS):
                return newN.path()
            elif parent.inPath(newS):
                pass
            else:
                add(newN, agenda)
    return None
```

implement

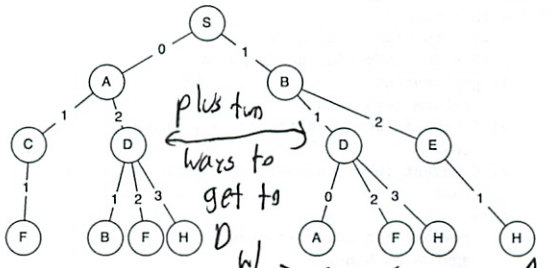
Adding a check for being in path

```
class SearchNode:
    def __init__(self, action, state, parent):
        # as before
    def path(self):
        if self.parent == None:
            return [(self.action, self.state)]
        else:
            return self.parent.path() + [(self.action, self.state)]
    def inPath(self, state):
        if self.state == state:
            return True
        elif self.parent == None:
            return False
        else:
            return self.parent.inPath(state)
```

← recursively walk up tree

Another pruning rule

Pruning Rule 2. If there are multiple actions that lead from a state r to a state s, consider only one of them.



along any path

Stack data structure

Last in, first out

```
>>> s = Stack()
>>> s.push(1)
>>> s.push(9)
>>> s.push(3)
>>> s.pop()
3
>>> s.pop()
9
>>> s.push(-2)
>>> s.pop()
-2
```

LIFO

push - adds element

pop - puts on

build

Stack Class

```
class Stack:
    def __init__(self):
        self.data = []
    def push(self, item):
        self.data.append(item)
    def pop(self):
        return self.data.pop()
    def isEmpty(self):
        return self.data is []
```

w/ a list

Queue data structure

First in, first out

```
>>> q = Queue()
>>> q.push(1)
>>> q.push(9)
>>> q.push(3)
>>> q.pop()
1
>>> q.pop()
9
>>> q.push(-2)
>>> q.pop()
3
```

FIFO

build

Queue Class

```
class Queue:
    def __init__(self):
        self.data = []
    def push(self, item):
        self.data.append(item)
    def pop(self):
        return self.data.pop(0)  #NOTE: different argument
    def isEmpty(self):
        return self.data is []
```

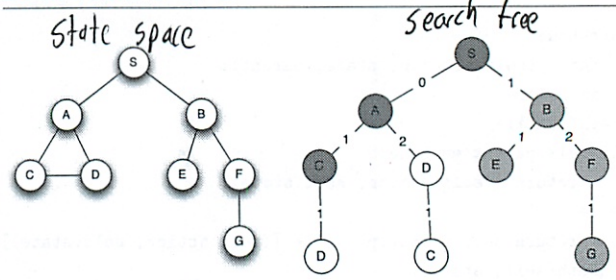
goes long as deep as it can

Depth-First search

```
def depthFirstSearch(initialState, goalTest, actions, successor):
    agenda = Stack()
    if goalTest(initialState):
        return [(None, initialState)]
    agenda.push(SearchNode(None, initialState, None))
    while not agenda.isEmpty()
        parent = agenda.pop()
        newChildStates = []
        for a in actions:
            newS = successor(parent.state, a)
            newN = SearchNode(a, newS, parent)
            if goalTest(newS):
                return newN.path()
            elif newS in newChildStates:  # pruning rule 2
                pass
            elif parent.inPath(newS):     # pruning rule 1
                pass
            else:
                newChildStates.append(newS)
                agenda.push(newN)
    return None
```

← initial

DFS: From S to C



1st agenda S
2nd agenda SA, SB → stack so test SB first
3rd SA, SBE, SBF
4th SA, SBE, SBFG ← don't add anything; back track
5th SA, SBE
6th SA
SAC ✓

DFS properties

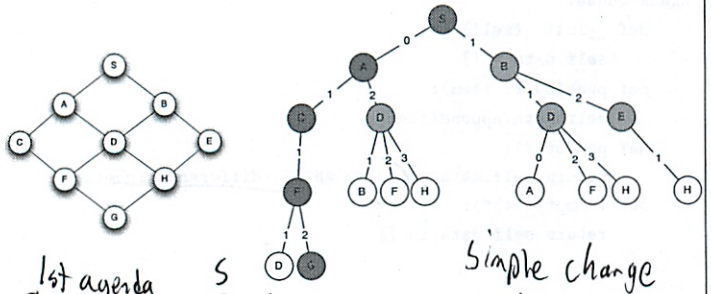
- May run forever if we don't apply pruning rule 1.
- May run forever in an infinite domain.
- Doesn't necessarily find the shortest path.
- Efficient in the amount of space it requires to store the agenda.
  - Only storing frontier

Breadth-First search

```
def breadthFirstSearch(initialState, goalTest, actions, successor):
    agenda = Queue()
    if goalTest(initialState):
        return [(None, initialState)]
    agenda.push(SearchNode(None, initialState, None))
    while not agenda.isEmpty():
        parent = agenda.pop()
        newChildStates = []
        for a in actions:
            newS = successor(parent.state, a)
            newN = SearchNode(a, newS, parent)
            if goalTest(newS):
                return newN.path()
            elif newS in newChildStates:
                pass
            elif parent.inPath(newS):
                pass
            else:
                newChildStates.append(newS)
                agenda.push(newN)
    return None
```

← tiny change in code

BFS: From S to G



1st agenda S
2nd agenda SA, SB
3rd SB, SAC, SAD
4th SAC, SAD, SBD, SBE
...
(gets really long)

Simple change in data structure makes big change!

BFS properties

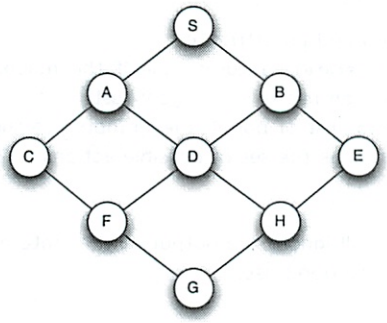
might be multiple paths

- Always returns a shortest path to a goal state, if a goal state exists in the set of states reachable from the start state.
- May run forever in an infinite domain if there is no solution.
- Requires more space than depth-first search.

to store agenda

Dynamic Programming

When happened when we did BFS in this city with goal G?



Visits 16 nodes, but there are only 9 states!!

breadth 1st

Dynamic Programming Principle

The *shortest* path from X to Z that goes through Y is made up of

- the *shortest* path from X to Y and
- the *shortest* path from Y to Z.

additive cost

So, we only need to remember the *shortest* path from the start state to each other state.

DP in breadth-first search

The *first* path that BFS finds from start to X is the *shortest* path from start to X.

So, we only need to remember the *first* path we find from the start state to each other state.

DP as a pruning technique

Pruning Rule 3. Don't consider any path that visits a state that you have already visited via some other path.

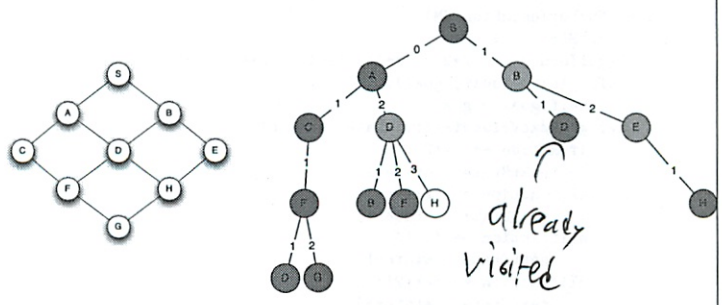
Need to remember the first path we find to each state.

Use dictionary called `visited`

BFS with DP

```
def breadthFirstDP(initialState, goalTest, actions, successor):
    ...
    agenda.push(SearchNode(None, initialState, None))
    visited = {initialState: True}
    while not agenda.isEmpty():
        parent = agenda.pop()
        newChildStates = []
        for a in actions:
            newS = successor(parent.state, a)
            newN = SearchNode(a, newS, parent)
            if goalTest(newS):
                return newN.path()
            elif visited.has_key(newS): # rules 1, 2, 3
                pass
            else:
                visited[newS] = True:
                newChildStates.append(newS)
                agenda.push(newN)
    return None
```

BFS-DP: From S to G



Visits 9 states. Can't visit more than # states

Can never expand more nodes than there are states.

Can be used with DFS as well.

keep track of visited as well

cutoff a lot more of the tree

State machines as world models

- Our search problems all involve sets of states and transitions between them
- Natural to think of using state machine formalism to find paths through state space
  - Given a state machine in its initial state, what sequence of inputs can we feed to it, in order to cause it to reach some goal?
  - Note that we can use search algorithms to find sequence of actions for us

State machines as world models

Use `getNextValues` as successor function in search

Inputs are actions

Add a method and an attribute:

- `done(self, state)`: returns `True` if the machine has terminated; we can use this as a goal test
- `legalInputs`: list of possible legal inputs to the machine; we can use this as the set of possible actions.

For now, we will ignore the output of the state machine. Later we will put it to good use.

Planning in a state machine

Question: Given a state machine in its initial state, what sequence of inputs can we feed to it, in order to cause it to enter a done state?

```
def smSearch(smToSearch, initialState = None, goalTest = None,
             maxNodes = 10000, depthFirst = False, DP = True):
    if initialState == None:
        initialState = smToSearch.startState
    if goalTest == None:
        goalTest = smToSearch.done
    return search(initialState, goalTest,
                  smToSearch.legalInputs,
                  lambda s, a: smToSearch.getNextValues(s,
a)[0],
                  maxNodes = maxNodes,
                  depthFirst=depthFirst, DP=DP)
```

A numeric example

- States: integers
- Start state: 1
- Legal actions (and successors) in state n:
  $$\{2n, n+1, n-1, n^2, -n\}$$
- Goal test: $x = 10$

Connect search to state machine

A numeric example – state machine

```
class NumberTestSM(sm.SM):
    startState = 1
    legalInputs = ['x*2', 'x+1', 'x-1', 'x**2', '-x']
    def __init__(self, goal):
        self.goal = goal
    def getNextValues(self, state, action):
        if action == 'x*2':
            nextState = state*2
        elif action == 'x+1':
            nextState = state+1
        elif action == 'x-1':
            nextState = state-1
        elif action == 'x**2':
            nextState = state**2
        elif action == '-x':
            nextState = -state
        return (nextState, nextState)
    def done(self, state):
        return state == self.goal
```

run it

Numeric – Breadth First

```
>>> smSearch(NumberTestSM(10), initialState = 1,
             depthFirst = False, DP = False)
   expanding:  1
   expanding:  1-x*2->2
   expanding:  1-x-1->0
   expanding:  1--x->-1
   expanding:  1-x*2->2-x*2->4
   expanding:  1-x*2->2-x+1->3
   expanding:  1-x*2->2--x->-2
   expanding:  1-x-1->0-x-1->-1
   expanding:  1--x->-1-x*2->-2
   expanding:  1--x->-1-x+1->0
   expanding:  1-x*2->2-x*2->4-x*2->8
   expanding:  1-x*2->2-x*2->4-x+1->5
33  states visited
[(None, 1), ('x*2', 2), ('x*2', 4), ('x+1', 5), ('x*2', 10)]
```

final path

Numeric – Breadth First with DP

```
>>> smSearch(NumberTestSM(10), initialState = 1,
             depthFirst = False, DP = True)
   expanding:  1
   expanding:  1-x*2->2
   expanding:  1-x-1->0
   expanding:  1--x->-1
   expanding:  1-x*2->2-x*2->4
   expanding:  1-x*2->2-x+1->3
   expanding:  1-x*2->2--x->-2
   expanding:  1-x*2->2-x*2->4-x*2->8
   expanding:  1-x*2->2-x*2->4-x+1->5
17  states visited
[(None, 1), ('x*2', 2), ('x*2', 4), ('x+1', 5), ('x*2', 10)]
```

Numeric – Depth First

Limit numbers to be in range (-20, 20)

```
>>> smSearch(NumberTestFiniteSM(10, 20), initialState = 1,
             depthFirst = True, DP = False)
   expanding:  1
   expanding:  1--x->-1
   expanding:  1--x->-1-x+1->0
   expanding:  1--x->-1-x*2->-2
   expanding:  1--x->-1-x*2->-2--x->2
   expanding:  1--x->-1-x*2->-2--x->2-x+1->3
   expanding:  1--x->-1-x*2->-2--x->2-x+1->3--x->-3
   expanding:  1--x->-1-x*2->-2--x->2-x+1->3--x->-3-x**2->9
20  states visited
[(None, 1), ('-x', -1), ('x*2', -2), ('-x', 2), ('x+1', 3), ('-x',
-3), ('x**2', 9), ('x+1', 10)]
```

Must give it a range - or will go on ∞
gives a longer solution

Computational complexity of algorithm

Let

- b be the *branching factor* of the search tree; that is, the number of successors a node can have.
- d be the *maximum depth* of the search tree; that is, the length of the longest path in the graph.
- l be the *solution depth* of the problem; that is, the length of the shortest path from the start state to the shallowest goal state.
- n be the *state space size*; that is the total number of states in the domain.

There are b^d paths at depth d.

The number of nodes in the tree of depth d is about b^{d+1}.

Without dynamic programming

- Depth first:
  - ⋆ may have to search every path (b^{d+1} nodes), but
  - ⋆ agenda is small (bd)
- Breadth first:
  - ⋆ may have to search to depth l (b^{l+1} nodes),
  - ⋆ agenda may be as large as b^l

With dynamic programming

Visit at most n states!

Sometimes $n << b^l$ (in a road network, for example), sometimes not (small problem in large space).

DP is almost always an improvement in running time.

This Week

Software lab: search

Design lab: None! Thanksgiving

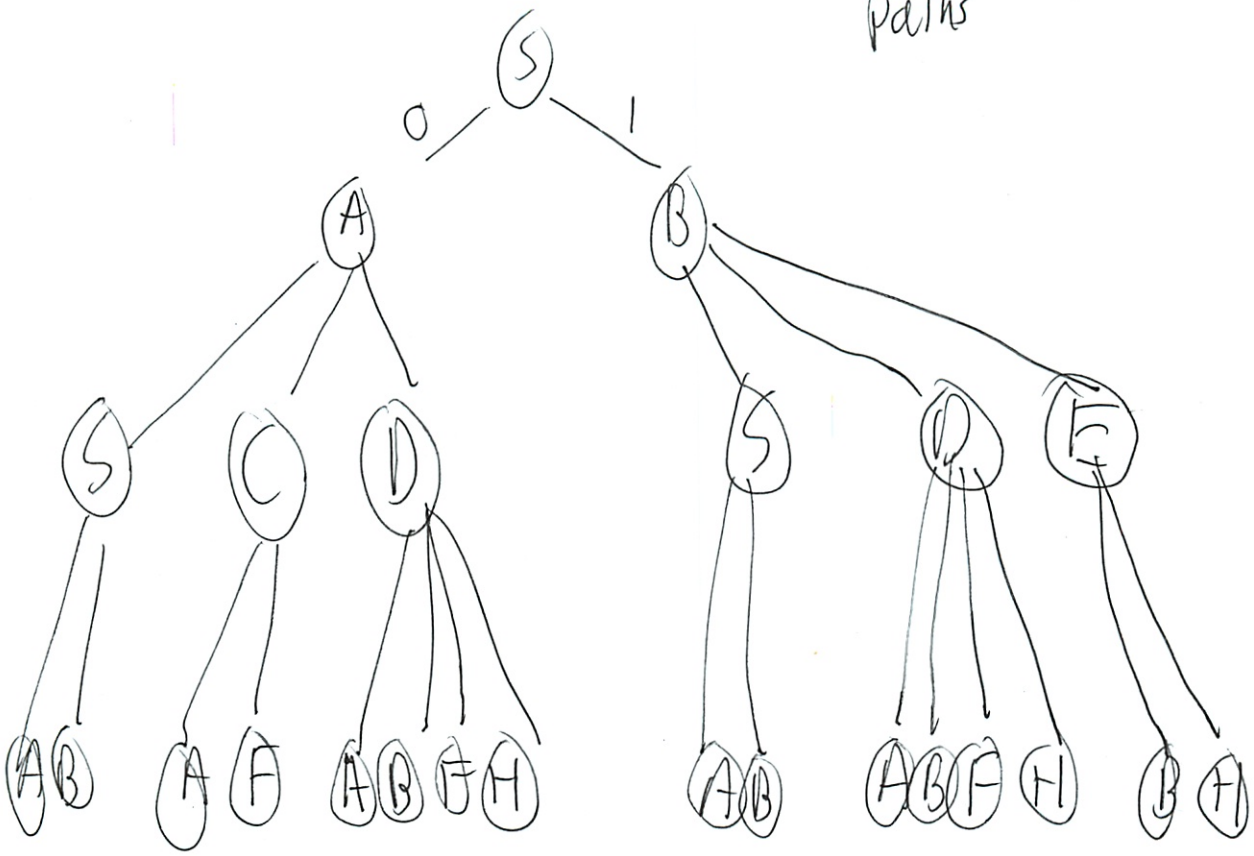
Nanoquiz Make-up: Wednesday, December 1: 4PM - 9PM in 34-501

Be sure to fill in the tutor problem to select which NQs you are going to make up:

- Everyone can make up NQ 1
- Everyone can chose any two additional two NQs to make up
- If you have excuses from S^3 for missed NQs, you can make those up as well.

If you choose to make up a NQ, the new score will replace the old score, even if it's lower.

Search Tree encodes all possible paths



only up to lengnt 3

need to know seq of actions you took
- so can walk back

Make agendai
- list of nodes we found
- but have not looked at children
- then visit its children
- and add their children to agenda

We want to search as few nodes as possible

Return path [(None, S_0), (A_1, S_1) ...]

```
def search (inital state, goal Test, actions, sucessor):
    if goal Test (inital state):
        return [(None, inital State)]
    agenda = [SearchNode (None, inital State, None)]
    while not empty (agenda):
        parent = get Element (agenda)
        for a in actions:
            new S = sucessor (parent.state, a)
            newN = SearchNode (a, new S, parent)
            if goal Test (newS):
                return newN.path()
            else:
                add (newN, agenda)
    return None
```

← if goal, return path

← otherwise add to agenda

swLab 12

11/23

- no lab handout, only tutor problems
- 12.2.1 Farmer et al. Machine
  - Farmer, Goat, Wolf, Cabbage
  - Farmer owns goat, wolf, cabbage
  - Come to left bank of river - need to get to other side (right)
  - Boat fits at most 2 of them
    - farmer must be 1
  - Farmer can't leave goat + cabbage alone
  - " " " wolf + goat "

Use a (SM) → sm Search to solve

State: (farmerLoc, goatLoc, wolfLoc, cabbageLoc)

(s state either L, R

(boat always w/ farmer)

actions
- take None
- take Goat
- " Wolf
- " Cabbage

(2)

Create SM

Sm = FarmerGoatWolfCabbage()

getNextValues returns (nextStat, nextState)

inital state (L, L, L, L)

if try an illegal state - machine does nothing

output: list w/ each time step

a) sm.transduce(['takeGoat'])

- starting at inital state

(R, R, L, L) ✓

b) sm.transduce(['takeNone', 'takeGoat'])

(R, L, L, L)

(R, L, L, L) impossible, so some ✗

- is it first state impossible → goat + cabbage alone

(L, L, L, L)

(R, R, L, L) ✓

(3)

c) Sm.traverse(takeGoat, takeNone, takeNone)

(R, R, L, L)

(L, R, L, L)

(R, R, L, L) <that was stupid, but legal> ✓

12.2.2 Farmer et al. Search

- Same rules of the game
- FarmerGoatWolfCabbage class SM
  - needs GetNextValues
  - legal Inputs → serves as successors function actions list
- startState
- done() method
- prevent illegal actions
- use order of state tuple
  state[wolf] → state[2]
- states must be tuples, not lists so can be keys in dictionary search
- write in sol1Work.py

(4)

So where to get started?

Need to do whole search thing

Code missing from lecture

- Oh they include in Search module?
  - So I don't have to copy
  - did we have an implementation example?
  - the numeric example

- what are legal inputs
  - the actions
  - list

- Then GetNextValues
  - need to check for illegal move
  - and flip state w/o affecting others
    - write flip method
  - turn state to a list

- Ok implemented list w/o checking
- first check that they are both on same side
  - make procedure

(5)

- such a stupid procedure
  - but PCAP!
- don't need to do if false
  - will auto fall back to doing nothing
- now how to check - if goat + cabbage
  goat + wolf
  - where to check
  - at end, if illegal state, return original
  - Cool - I like how I coded this
- now need done when (A, B, A, A)
  - better type write or will never finish!
  - note the very cool return state == (A, A, A, A)
    True if true
    False if False
- try running
  - "search failed after visiting 1 state"
- ~~Add~~ TA: add in print state, action to debug

(6)

Fails because every input (all 4) return (L,L,L,L)

- what is doing that?
- my check if they are on same side always true
  - oh not use new State.L
- now fails after 3 states
- how can I put divider line b/w stages/states its trying?
- when it takes to goat → its not flipping
- oh it is flipping → I'm printing input, not output
- when L, R, L, L can't figure out what to do next
- i all are back where before
  - did takeGoat RRLL
    takeNone LRLL

    then can't figure out next

    takeNone - been there before

    takeGoat - illegal

I messed up code to make it easy to debug

5)

If the farmer is there

Can have goat + cabbage
goat + wolf

After class 11/23

So need to change to goat + cabage + no farmer
this time function of just new state
- no, bad for debugging
- just do diff side

Ok runs now and finishes!
9 states visited!
Check tutor ✓
Ha they have pictures!
My FSM still does not debug!

That was fast - lab over

A bunch of hw due 12/1
- might as well do now

Week 12 HW

11/23

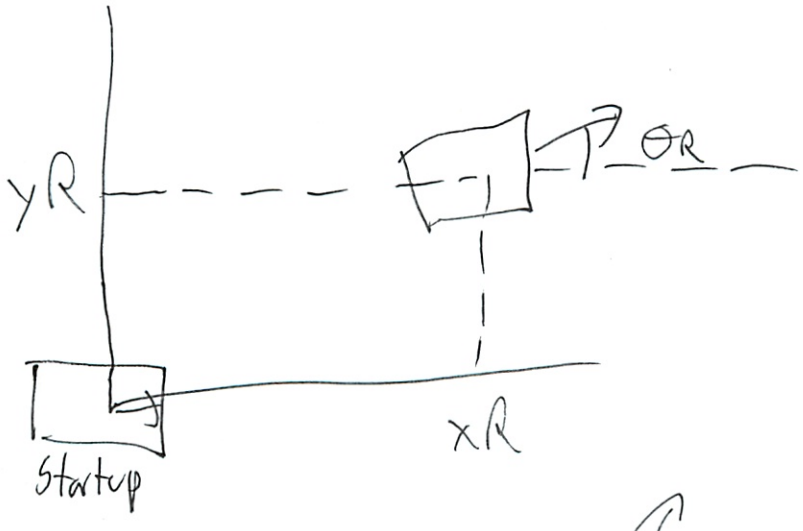
12.3.1 Sonar Hit

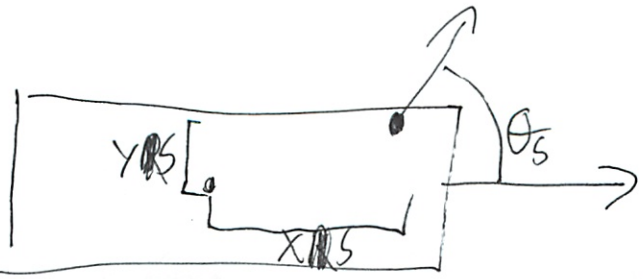
- interpreting sonar sensor on robot
- returns distance measurement as a point
- interpret soner in the robot's odometry frame
- need to know robots location + orientation "pose"

Sonar Pose(xS, yS, thetaS)

↳ center of sensor relative to center of robot
angle from robots nose to beam

Robot Pose(xR, yR, thetaR)





(2)

So essentially two coordinate frames A, B

origin of B is (xB, yB) relative to A

X-axis of B rotated by thetaB relative to x-axis of A ...

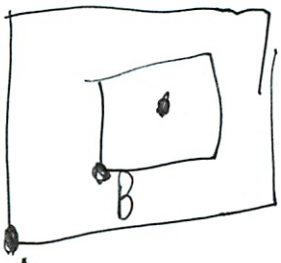
$$aX = xB + \cos(thetaB) * bx - \sin(thetaB) * by$$
$$aY = yB + \sin(thetaB) * bx + \cos(thetaB) * by$$

- So basically when thetaB = 0

$$aX = xB + bx$$
$$aY = yB + by$$

- this is done by Pose.transformPoint in util
- draw pictures to help understand
- write SonarHit → given a distance measurement from one sonar, sonar's pose on robot + robots pose on world → returns Point in world frame

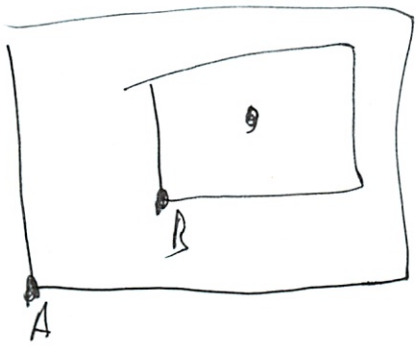
11/25
At home, home

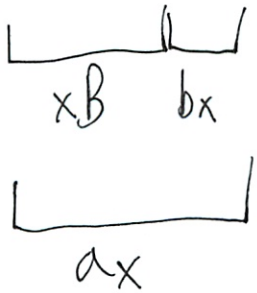


? what is significance of
x_B vs a_x
why are capital and second?
" other lower " first?

$a_x = x_B + (\cos(\text{theta}_B) \cdot b_x - \sin(\text{theta } B) \cdot b_y$

- Oh x_B is coords to start of B
b_x is X coord in B frame

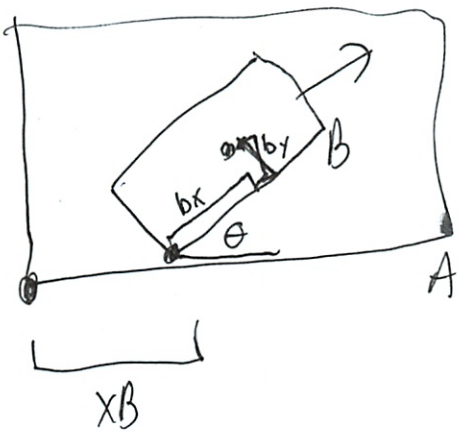




? So why all the complex w/ theata?
- Something w/ angle

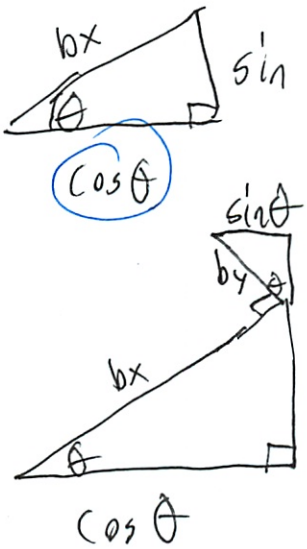
9

- oh if coord frame rotated

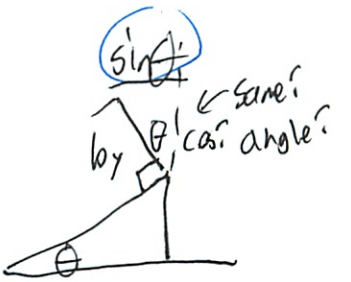


$a_x = b_x \cos(\theta) - b_y \sin(\theta)$

(but why do we care about vertical offset



- oh



(it seems to me it would be − minus

Oh it is I copied formula wrong

ok I confirmed that ✓

Now what to do for problem?

- so return a_x, a_y, given the values
  - actually pretty easy - they did the math

(5)

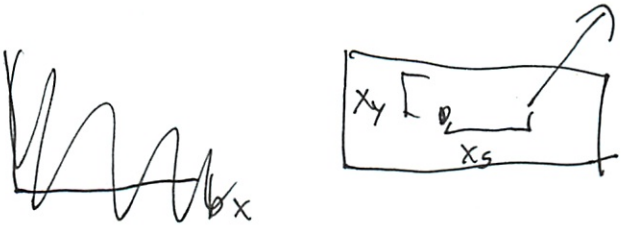
returns Point

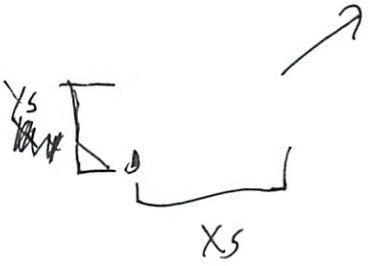
↳ X, Y

$$ax = xR + \cos(\theta R) \cdot \underline{bx} - \sin(\theta R)\ \underline{by}$$

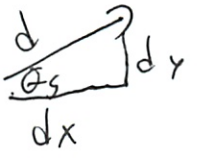
↑ angle ~~P~~ coord ↑ robot rotated from world

↑ for these need to do simple trig





; so want to decompose distance



$\sin\theta_s = \frac{dy}{d}$ $\cos\theta_s = \frac{dx}{d}$

$dy = d\sin\theta_s$ $dx = d\cos\theta_s$

$$bx = x_s + d\cos\theta_s$$

$$by = y_s + d\sin\theta_s$$

6

✗ instance method object is unsubscriptable - on tuple

Oh need to do sonarPose.xyt Tuple()[0]

? forgot before

✗ spelling error

✗ conflicts over math.

And y is off on first one

- but don't see anything wrong!

✗ Now it says cos undefined - oh θ sign

- well what is it?

✓ finally got it

12.3.2 Ideal Sonar Readings

- now need to figure out where robot is) localization
- given known map
- like when we were doing state estimation
- have a model of how robot interacts w/ world → ssm.StochasticSM
- feed into state estimator
  - constantly updates prob dist

6

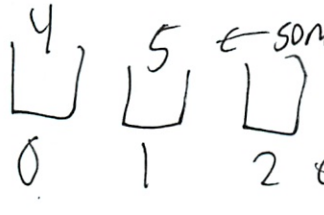
- will do in des Lab 12
  - Using Sonar 0 to gather info
- have a sensor read error model
  - Similar to like colors in hallway
- here will have perfect sensor readings
  - wallSegs - list of line segs representing walls
    - Util.LineSeg class
  - robotPoses - list of poses (over time?)
- SonarHit - what we hit before
- SonarPose0 - w/ respect to robot
  - for
- SonarMax - limit that sonar can read
- # Obs that have passed

Task: Implement the following procedures

1. discreteSonar - takes sonar reading as input
   - generates integer bin index as output

(8)

- range #Obs # of bins
- any sonar reading goes into last bin



? but confused how all reading ≥ sonar max should go in last bin

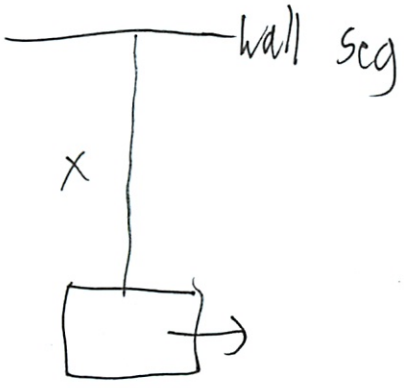
- Then they say interval 0 → 1 divided into 3 bins

- use round + int

2. IdealReadings
   - takes list of wall segments
   - takes list of robot poses
   - returns list of discretized sonar readings for Sonar 0 for each state

↑ so tells you what the sonar <u>should</u> be reading

(9)



- look from live seg to sonar max
- look for any intersections w/ live seg
  - intersection() method in util.LineSeg
    - ah thats nice of them
- for each state
- give test case

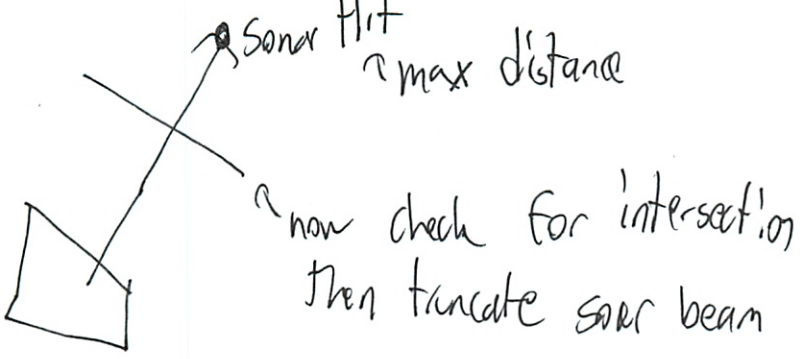
Go lets start on part 2 - ~~see~~ I actually understand what want to do

ans = []

for each state

- implement broader too (if θ != 0)
  - relatate to world, w/ function from last problem
    - Sonar Hit
      - don't we no know where hits?
        - well that is max dist

(10)



; will only 1 intersect

No must check for min

- intersection reports what -> a point
- then need to find length
  - make a line to test lenght
  - but line seg does not have dist :(
  - Can I extend class?
    - yes!

so $\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$

Worked

- can have object as array index
- Use ;
- now list assignment out of range
  - ***
  - ;;;
  - just append

(11) then don't need i

Now unsupported bool , instance in My Line

- oh I know → returns false if it never intersects
- oh False
  ↑must have cap F

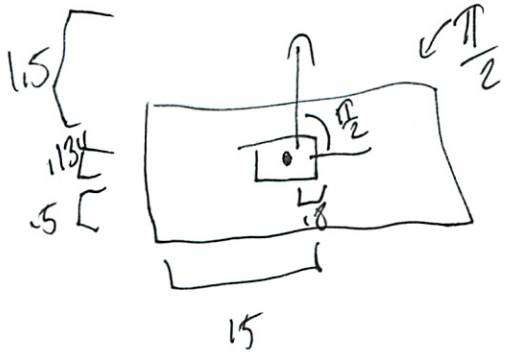
∴ Now need to check for smallest

- we did this before I think
- just rewrite
- opps I did max Value

- ok think got it

(X) Major wrong

- rethink → correct ans 9, 4, 5, 4
  "but tracks longer than max sensor!



π/2 angle

is right .58

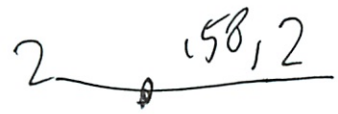
~~beam~~ ~~light~~ 2.134

well end of the beam

- yeah sounds right

(12)

(rosses at ~~[illegible]~~ line (0, 2) (8, 2)

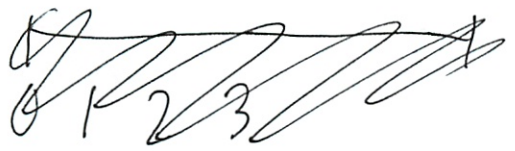


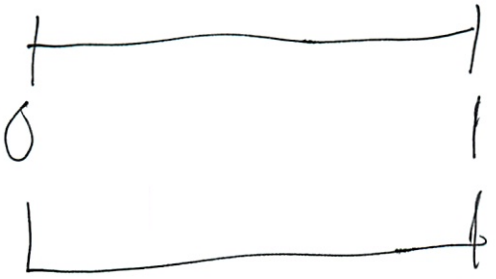
- or is ~~[illegible]~~ lenght cant total offset

- what am I canting as lenght
- start ~~(that is beam start)~~; → that point
- what is start .5, .5
  - oh should be .58, .634
- oh and the beam end is sonar hit ~~[illegible]~~ ???
  - yeah
- ok ans are much nicer decimals now (since lenght of ⌐ line)
- but the ans is 9 ;(
  - what qs do they even want us to answer??
- and they want us to discretize
  - just cand
- oh return sonar Max if none
  - did not change anything
- so ~~[illegible]~~ my ans the 2nd and 4th are same - so off by a factor

(13)

Oh! Is decrease mean split it





reduce

0 .15 .30 .45 .6 .75 .9 1.05 1.20 1.35 1.5

.6 .86 1.36

4 5 9

✓

thats' it - so that is what they meant!
- clever detective work

go back to first one to do list

but then bins not by numObs, its by sonarmax
numObs has nothing has to do w/ this!
or that is # of bins to divide into

(14)

So box width = $\frac{\text{numObs} - 1}{\text{~~maxObs~~ sonarMax}}$

then how to split it?

for loop

!=0

if ~~i~~ > i * width and < (i+1) * width

return i

end > return numObs - 1

(i could put 1st to optimize - but don't care about here)

- not working well
- oh opps flip $\frac{\text{sonar Max}}{\text{numObs} - 1}$
- now kinda close - weird
- and added = - what are the cases it is failing

(15)

in	my	correct	
0	9	0	✗
.14	0	0	✓
.16	0	1	✗
1.34	8	9	✗
1.[illegible]36	8	9	✗
1.[illegible]49	8	9	✗
1.5	8	9	✗
5	9	9	✓

← need = to for < min
└ actually don't need min case
- will exit 1st

← oh print i+1

So now all right except the 0
add the min back in
- now I wrong
- how I am starting at bucket 1
- remove i+1 to i

(16)

It is judging .16 as correct b/c it's .16667 bins

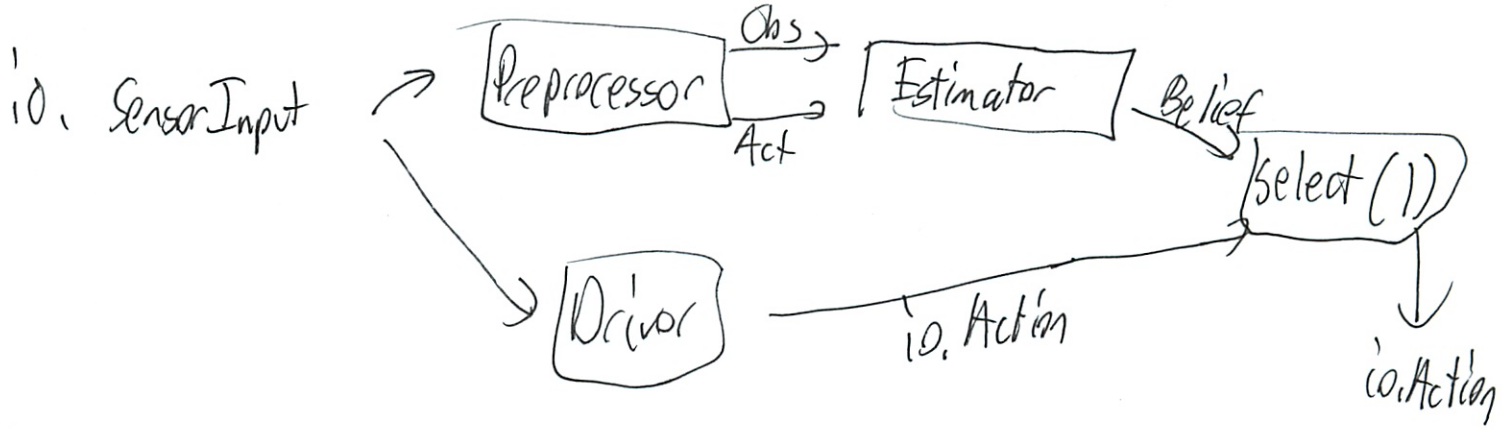
So if divide by numObs not numObs-1 bins it works

right on target

now add descritize to 2nd function

12.3.3 Localization

- in last problem computing ideal sensor readings for a world
- now general structure of localization system



Preprocessor - takes input io.SensorInput
- generates (output, action)

Estimator - instance of seGraphics.StateEstimator
- from last sw lab

Driver - instance of move.MoveToPose class

(17)

- moves robot in a straight line

Parameters

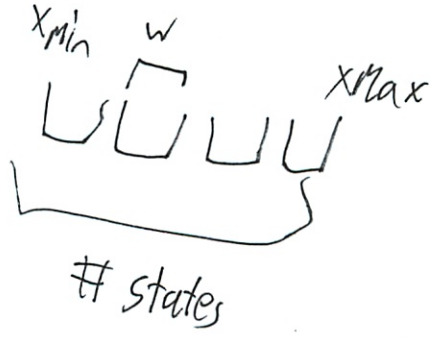
numObs = 10
- for descritizing

num States = 10
- possible values of robot

xMin, xMax - bounds of robot

y = fixed y of robot

Sonar Max



---

we want to know Robot x's coord based on seq of observations + actions.

$w = \frac{xMax - xMin}{\#\ states}$ ∴ or am I flipping it again

Preprocessor

Input → 10. Sensor Input

Output → (obs, act) pair
- each integer index

(18)

- will infer actions from observing robot odometry
  - gives us relative displacements
- last movement = $X_t - X_{t-1}$
- but discretized → $\inf(\text{round}(\frac{(X_t - X_{t-1})}{w}))$
  - w: width of state interval
- must include output
  - obs $t-1$
  - act $t-1 \to t$
- both None when starts
  - make no state update
- (so basically a descritizer)

Estimator

Input → (obs, act) from above

Output → a belief state DDist of location of robot in the world

- using a ssm. Stocastic SM
  - start w/ uniform
- Then do Bayes + TPT.

Observation Model

- is a conditional distribution
- will assume a discretized ideal/perfect reading

Transition Model

- proc that takes action + returns a proc which takes start state as input returns a distribution
- assume perfect transition model
- ~~give action A, state s~~

So now the qv

Sonar readings = 5, 1, 1, 5, 1, 1, 1, 5, 1, 5

- give value of pre processor + estimator

1. Preprocessor at time 0

input sonars (.8, 1, ...)
odometry (1, .5, 0)

(20)

Output (obs, act)???

- ? so what is obs and act again?
- ? is it None, None since first started ✓

2. Preprocessor time 1

input sonar (.25, 1.2, ...)
odometry (2.4, .5, 0)

output (obs, act) ??

- So now need to actually ans
- Sonars, just want first .25
- using code from last problem?
- So obs = 1 ← goes in bin 1
- but what is action?
  - given int(rand($\frac{X_t - X_{t-1}}{w}$))

$\frac{2.4 - 1}{1} \leftarrow \frac{10-0}{10} = 1.4 \xrightarrow{\text{rand}} 1$

(x) (y)
so (1, 1)

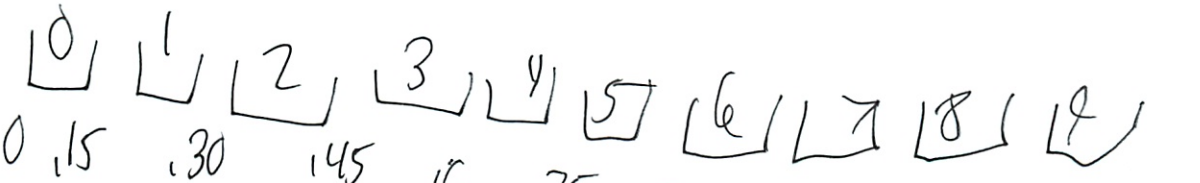
(21)

Also only have 10 checks - must be careful

action right, but not obs

Obs is just descritized sensor reading, right?

- yeah $\frac{1.5-0}{10} = .15$ intervals



0 .15 .30 .45 .6 .75 .9 1.05 1.20 1.35 1.5

↑

but why is it not 1?

- Check in. sensor Input

- yeah is list of sonars

- Or is it obs from t-1 so that was [illegible] .8 → 5 ✓

before check will do next ↑ why?? sees silly

- prob something w/ timing

b/c it uses old obs + current action

but obs is ↑ sonar

and movement is ↻ wheels

can't just add

- but order of our state estimate

3. Preprocessor time 2

In Sonars (.16, .2, ...)

Pose (7.3, .5, 0)

22

is look then move

Obs → this time $t-1$ so .25 → A ✓

act → $\frac{7.3-2.4}{1} = 4.9$ ✗

Oh duh rand 5 ✓

4. Estimator (time 0)
   - So this got input None
   - using ssm
   - just starting state
   - which is uniform

   .1 .1 .1 .1 .1 .1 .1 .1 .1 .1 ✓

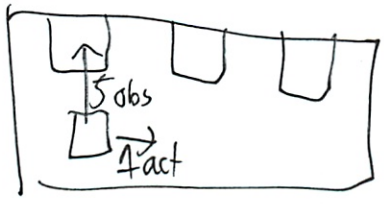
5. Estimator (time 1)
   - Inp (5, 1)
   - So this is w/ Bayes + Total
   - Should I try to program or do by hand?
   - Obs model is is perfect, so are in state 5
   - Then action model perfect, so state 6 ???

(23)

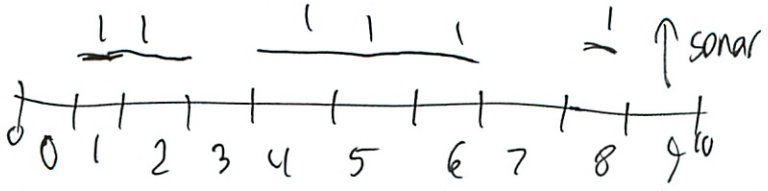
Can you even do that ???

- Since as I said it's two diff things

The 5 sensor state helps we know the lines



- oh right did this in last problem
- But ~~see~~ they did not give the wall pos in this problem
- ~~estimater says~~
- last problem says [4, 7, 5]
  - So these are x, pos where see this
  - no these are the ideal sonar readings for a given pose, knowing walls
  - in localization you do have map of hallway
  - oh this is what ideal [5, 1, 1, 5, 1, 1, 1, 5, 1, 5 is]



(24)

So an [illegible] sonar reading of 5 means was in state 0, 3, 7, 9 then moved → so can be in states 1, 4, 8, 9

So then prob dist (guessing - not doing "right way")

0 .25 0 0 .25 0 0 0 .25 .25 ✓

[illegible] Woot thats it!.

6. Estimator (time 2)

Input (1, 5)

- using past history? yeah I guess (I see how this is coming together - this is difficult!)
- So we know we are seeing a 1
- So can't be at 0, 3, 7, 9 so

0 .333 0 0 .333 0 0 0 .333 0

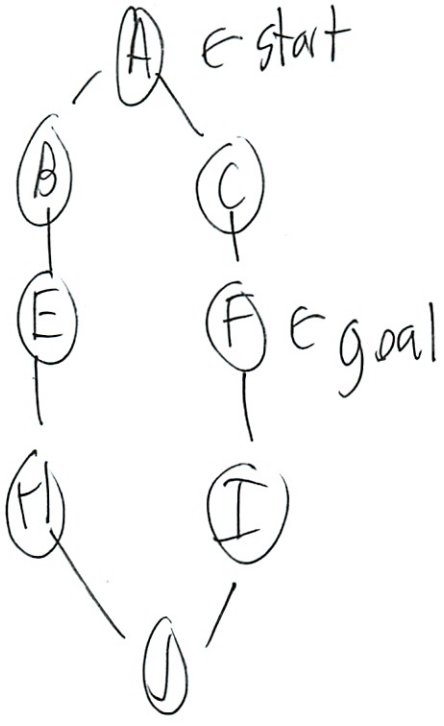
- Now a movement 5 →

0 0 0 0 0 0 .333 0 0 .1666 ✓

25

12, 3.4 Compare Search

- ok diff topic



children pushed onto agenda in reverse order

???

not revising

no dynamic programming

1. What path w/ ~~breath~~ breath 1st?

   A B C E F ✗

   - ? do they want full path - or output

2. Depth A B E H J I F ✓

   - but what is this about reverse alpha order

what are they looking for exactly ??

So for 4 is that about reverse alph order

agenda is a queue

(26)

And added to queue in reverse alphabet order?

1. A C B F (X)

I wish they had the animation online

Look at course notes

~~choose~~ choose oldest path from agenda (queue)

So A C B ~~F~~ F

↑ added first ↑ F finds - don't get why wrong

~~A B C~~

does A

↳ new agenda C B

↑ reverse alpha order

do C

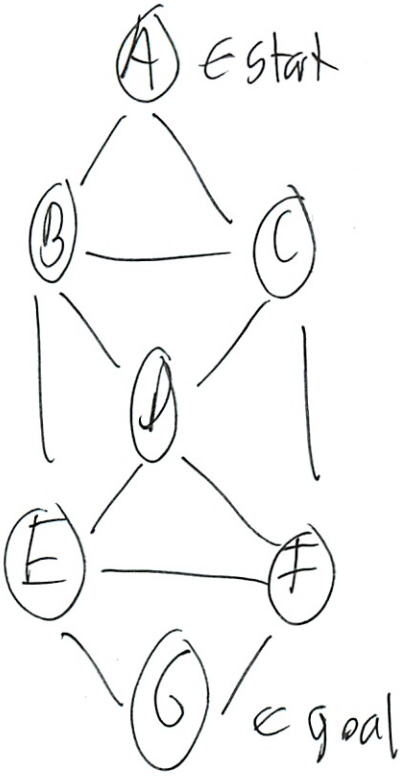
↳ agenda B ~~F~~ (F) ← see F so stop?

do B

↳ agenda ~~F~~ F ~~E~~ E

So A C F (✓)

12.3.5 Compare Searches (more complex)



alphapetical order
won't revisit same state within path

1. Breath first no DP seq of ~~agenda~~ paths pushed to agenda
   - ~~seen~~
     1. A
     2. A B
     3. A C

   } given

- ? what do they mean seq of paths pushed to agenda
- or seems like seq of paths checked - giving full seq each time

28

Check A

↳ new agenda B C

Check B

↳ new agenda C D F ~~F~~ can't get there from B

Check C

↳ new agenda ~~D~~ D E F D E F

missing some — see next pg

can repeat on diff path

Check ~~A~~ ~~B~~ B

↳ new agenda E F ~~D~~ E F E F

Check ~~A B C D E~~ E

↳ new agenda ~~F~~ F ~~D~~ E F E F G

Bingo

~~plus~~ what I checked already

so enter this

(X) so the final ABE G was right

and #9 was A C F

So what in all world did I do wrong??

29

Did I miss ~~1st~~ horizontal line?

A
↳ A A
B C

A
B
↳ A AB AB AB
C C D E

A
C AB AB AB AC AC AC
↳ C D E B D F

AB
C AB AB AC AC AC ~~ABC~~ ABC ABC
↳ D E B D F ~~B~~ D F

won't do same path

AB
D AB AC AC AC ABC ABC ABD ABD
↳ E B D F D F E F ← btw how do you represent this ~~order~~ in code?

AB
E AC AC AC ABC ABC ABD ABD ABE ABE
↳ B D F D F E F F 6
✓ ✓ ✓ ✓ ✓ ✓ ✗ ✗ ✗ ✓ bingo

Enter like this ↑ so what went wrong here go back up ABD add in? ✓

(30)

The last 4 are

ABD C ABD E ABD F final ABE G (4)

↳ never mention ABE F

"does it never get added to the queue?"

Now w/ DP (Breadth 1st)

- what is DP again?
- don't consider a state where already visited on diff path (pruning rule 3)

So just go through with it

A
↳ AB, AC

AB
↳ AC, ~~AB~~ C (not this time! DP), ABD, ABE, ~~ABF~~ no!

(what I was doing 1st time I ~~thought~~ think)

(31)

A
C L) AB AB AC AC
D E D F
already visit on other path?

AB
D L) AB AC ABD ABD
E F E F
DBP DP

AB
E L) AC A BE
F 6

Enter ✓

12.3.6 Paths in the Map

Look in tutor12work.py
Call search() to find path b/w nodes in map1
using the 4 search methods DF, BF, DFOP, BFOP
w/ verbose to see it

- goal is 6 ;;

depthFirst('5', lambda x: x == "6", map1, map1)
↑ or list of legal actions

57

['S','A','B','C','D','E','F','H','G']

(X) Dict object not callable

Oh need ~~def map~~ successor

- so instead map1 above (last param) map1successors

```
def map1successors(s,a)
    return map1[s][a]
```

- all these examples in course notes anyway

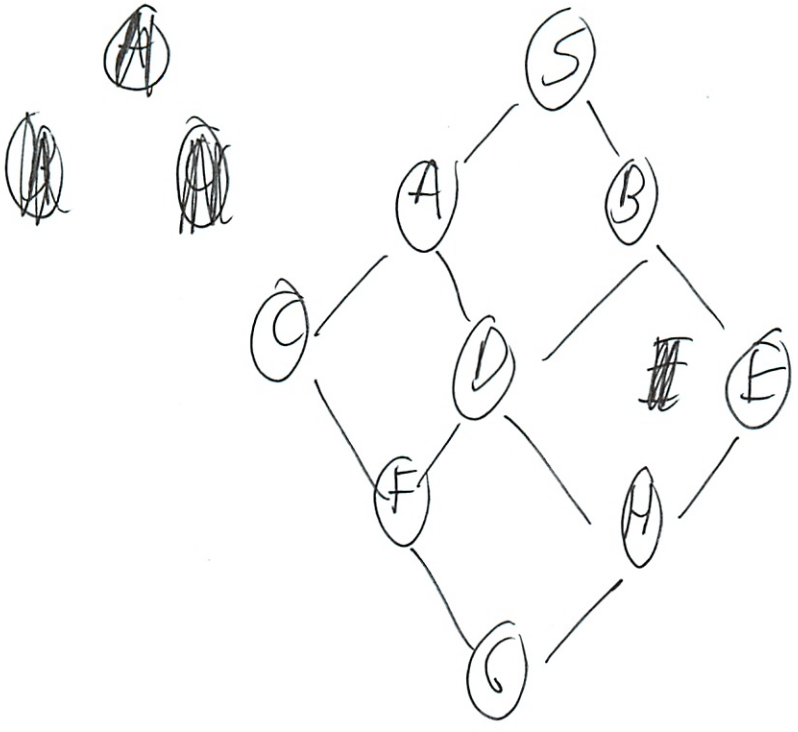
(X) List indicies must be ints, not strs

This might be the legal action thing

- should be #
- course notes says 0, 1, ..., n-1 where n is max # of successors in each state
- from lecture notes just the max # of actions
- then change map1successors to return current state if that option not available
- thought it looked familiar - just copy from lecture notes

(✓) fixed

(33)



its this one

depth first

S → B, E, H, G

Breth first

S → A → C → F → G

depth 1st DP S → B → E → H → G

Breath 1st DP S → A → C → F → G

Now questions

1. BF no DP → A → C → F → G ✓

- start at A →

12 states visited ✓

6 nodes visited (count lines) ✓

(39)

4. BF w/ DP A → G

A → C → F → G

8 states visited

6 " expanded

7. Name of states visited more than once

" How am I supposed to know

just look at agend over time w/o DP

A S C D B F B F H

B D E

makes sense to have 4

So B D F (X)

" wrong format "

B F B D (X) just pick 1 ✓

B F D (X)

8. The path w/ w/o DP should be same for BF?

- True best path will be same ✓
- it just discards bad paths earlier

9. Enter max # states that can be visited w/ any BF-DP

35

(First state does not count as visited)

So $n-1$ where n = # of states

There 8 ✓

10. ~~BFS~~ DF, no DP G→C

G → H → E → B → D → F → C ✓

(they pick diff states so hard to do on paper)

10 states visited ✓

16 " expanded ✓

11. DF DP G→C

G → H → E → B → S → A → C ✓

↑ diff ✓

8 states visited

6 states expanded ✓

16. Name of states visited more than once w/o DP

G F H D E B ~~S~~ 2 S D A A

DF ✗ makes sense to have

what the hell do they want here?

(36)

They say a state

- "i so pick 1" ✓

17. Enter max # of states that can be visited w/ any DFDP

- same as before:

n-1

8 here ✓

18. Path should be generally same DFDP/not?

- No
- we saw here it has not ✓

12.3.7 Robots on a Grid

- write a program that uses search to plan paths on a grid
- states are tuples (i, j) (or (x, y))
- robot can go up, down, left, right

Make a SM definition of this domain

- have a done when at (3,4)

- This should be easy

(37)

What does done do?
- return True when done?

(1) Got it to work

12.3.8 Obstacles (optional)

Consider the following functions for ~~dea~~ searching maps ↙ map test
- like in course notes chap 9

- - wait do they have a mistake - defining functions inside other function definitions.
  - email in
  - response: read 3.4 in course notes. See excesizes 3.12 + 3.13
  - so basically define inside procedure env
    - so only accessible from the proc env
  - that's my guess

Want to generalize map searching so some actions + states are forbidden
↑ arcs paths

So create proc map Obst Test
- 2 arguments

38

- badStates - list of bad state pairs
- bad Arcs - list of state pairs
  - directional / direction matters

if it is bad state → stay at current state.

It should be a lot like map test

Hint: Python in operator

- So need to test actions + states
  - seperatly

11/29
MIT Dorm

do state first
- state completly on blacklist
- new state is map[s][a]

Transition
- current s
- new is map[s][a]

⊗ Search failed after visiting 1 state

① fixed it up

(39) Knight paths on a chessboard 12.3.9

- write a search path for knights move
- states → tuple (i, j) ie (x, y)
- knight can move

8 moves possible

- board 8×8 (0, 7)
- fill out the sqn
- just have states be offset tuples
- do it
- then ~~check~~ check if off board - if is, ~~just~~ just keep current state
- ; so just like 12.~~A~~3.7 ;;

(X) Unsubscriptable item NoneType
- where am I returning that ;

Oh I never returned getNextValue

(X) One works - other returns wrong answer
(V) Oh was bad returning of state

12.3.10 2 Robots on Grid

- ∞ grid
- same left, right, up, down
- want robots to meet in same spot in min steps
- have 2 instances of Robot Moves
  - w/ State machine combinators m
- Copy over Robot Moves class
  - ch new goal
- ¿ Parallel SMs ?
- How does goal FN get called ?
- and why only 1 s ?

(X) errors w/ startState

Oh input split

↳ So parallel 2 machine

returned something now

~~Oh well 2 ish~~

Now need to implement goal Fn

- printed to see what it is
- do a try catch ¿ for None ?

(11)

✓ 1st test

✓ 2nd test

✗ 3rd test fails - gets a none

- oh I see where found goalFn
- but why can't it move past none?
- it never runs getNextValue ever!
- goalFn returns none and I return false
  - and it gives up!
- But it should be the initial state! ((3,0),(0,3))
- emailed in

reply back

- had right ans, difficult formatting
- needed to ~~enter~~ enter small subset of what I had ✓

done hw