Approximate Pathfinding Algorithms Using Ergodic Markov Chains

The Esolangs Code Guessing Round 25

Finding Paths

Pathfinding is a classic problem for which exact algorithms are unreasonably complex. In this Tumblr post, we define a *pathfinding problem* as a 3-tuple (G, e_α, e_ω) where $G \stackrel{\text{def}}{=} (E, V)$ is a directed graph and $e_\alpha, e_\omega \in E$. The graph G is contractually represented using an adjacency matrix $A = \{0,1\}^{|E|\times|E|}$. A *pathfinding solution* to such as problem is a sequence of n edges $S = s_0, \dots, s_n$, such that

- $s_i \in E$ for all s_i,
- $s_0 = e_\alpha$,
- $s_n = e_\omega$, and
- $A_{S_{i-1},S_i} = 1$ for all $0 < i < n$.

A pathfinding algorithm is then morally represented as a function from the set of pathfinding problems to pathfinding solutions. We will try our hardest to construct such a function.

Abolish Borders

In the general case, a pathfinding problem is awfully stressful. We don't even know if there exists a path from e_α to e_ω — how terrifying! In these sorry situations, one typically makes additional assumptions. For example, we could assume the existence of a pathfinding solution. But, following in the footsteps of famous philosophers, we shall assume stricter conditions on the state of the universe. We assume that a solution S exists irrespective of the choice of s_α and s_ω. This will not be motivated.

Markov Chains

We skip some crucial definitions as an extra challenge

Transition

Hey darn would you look at that, the adjacency matrix is almost the same as the transition matrix for a Markov chain. We just need to fix it to fit the definitions above. Figure 1 shows an algorithm for converting an adjacency matrix into a transition matrix.

Figure 1: An algorithm for converting an adjacency matrix into a transition matrix

```
def trans(A):
    return A / A.sum(0)
```

Ergo, Ergodic

Theorem 1. Let P be the transition matrix for a regular Markov chain. Then, the limit

$$\lim_{n\to\infty} P^n = W$$

exists and converges to the limiting matrix

$$W = \begin{bmatrix} \boldsymbol{w} \\ \vdots \\ \boldsymbol{w} \end{bmatrix}$$

consisting of copies of the fixed vector $\boldsymbol{w}$. Furthermore, $\boldsymbol{w}$ is a left eigenvector of P corresponding to the eigenvalue 1, or equivalently, $\boldsymbol{w} \in \operatorname{coker} P - I$. Oh and also, $\dim \operatorname{coker} P - I = 1$. We'll need that later

To prove this, we use the fact that W contains only constant columns. As rows of P are probability vectors, applying P to a probability vector $\boldsymbol{x}$ transforms it under some weighted averages. This transformation monotonically reduces $\max \boldsymbol{x} - \min \boldsymbol{x}$, which can be used to prove the first part of the theorem. The

rest ought to be simple enough with some basic linear algebra, too, so don't shy away.
As a bonus, figure 1 shows an algorithm to find the limiting matrix of a regular Markov chain.

Figure 1: An algorithm to find the limiting matrix of a regular Markov chain

```
def lma(A):
  w= linalg.null_space((A - np.identity(len(A)).T) # TODO: test this code
  return np.tile(w, len(w))
```

Theorem 1. Let P be the transition matrix for an ergodic Markov chain. Then,

$$\frac{1}{2}(I + P)$$

is regular, and

$$\lim_{n\to\infty} P^n = \lim_{n\to\infty} \left(\frac{1}{2}(I + P)\right)^n.$$

Proof: ■

Figure 1 shows an algorithm to convert the transition matrix of an ergodic Markov chain to a regular one, preserving limiting matrices.

Figure 1: An algorithm to convert the transition matrix of an ergodic Markov chain to a regular one, preserving limiting matrices

```
def regular(A):
    return (A + np.identity(len(A))) / 2
```

Mean

Ergodic Markov chains allow one to slide from one state to another rather smoothly. However, in order to be appropriately smooth, one must apply the appropriate amount of gel to their hair. To estimate this, a new metric is required.

An ergodic Markov chain, initially at state s_i, will reach a state s_j in some expected number of steps. We denote this m_{ij}, for the mean passage time. As an unmarked theorem, check out how m_{ij} is bounded by the number of states! Also, we denote r_{ij} by the mean recurrence time, or how long it'll take little ants to retrace their steps. Look, I'm just copying from a textbook at this point, it's not like this really matters.

Let's use a matrix M to hold all the mean times; indexed how you'd expect. Here comes a consequence of that:

Theorem 1: Let's use a matrix M to hold all the mean times for a matrix P; indexed how you'd expect. Then,

$$(I - P)M = \mathbf{1} - diag(r_0, \dots, r_{something}).$$

Wahoo we won't use that I'm just padding the word count are we at 4000 yet

The misadventures of skater Zeb

A cool matrix Z is given by $(I - P + W)^{-1}$. This inverse exists*.

Proof. Let $\mathbf{x}$ be a column vector such that

$$(\mathbf{I} - \mathbf{P} + \mathbf{W})\mathbf{x} = \mathbf{0} .$$

To prove the proposition, it is sufficient to show that $\mathbf{x}$ must be the zero vector. Multiplying this equation by $\mathbf{w}$ and using the fact that $\mathbf{w}(\mathbf{I} - \mathbf{P}) = \mathbf{0}$ and $\mathbf{wW} = \mathbf{w}$, we have

$$\mathbf{w}(\mathbf{I} - \mathbf{P} + \mathbf{W})\mathbf{x} = \mathbf{wx} = \mathbf{0} .$$

Then, the entries m_{ij} to that previous M is given by

$$m_{ij} = \frac{z_{jj} - z_{ij}}{w_j}$$

As you would expect, this is implemented by Figure 1.

Figure 1: this is implemented by

Finally we find a path

A skill solution can be estimated using the mean passage time matrix. For a skill issue from states A to Ω, the right indices in m will tell us how long to wait. Then it is a matter of simply enumerating every single path of length $(\cdot)$ until we find the right one. We mightnot find a path since it's only the average. But that's okay. Approximations are always good, especially if they are slow.

Figure 1: The final countdown

```
    def search(A, l, a, v):
        <haskell do block with [State a] monad/>
```

Sources

https://cg.esolangs.gay/25/
https://en.wikipedia.org/wiki/Absorbing_Markov_chain
https://en.wikipedia.org/wiki/Monte_Carlo_method
https://en.wikipedia.org/wiki/Categorical_theory
https://www.google.com/search?hl=en&q=matrix%20inverse%20calculator
https://alf.nu/SHA1