STATE SPACES

① Consider the "word ladder" problem. In this problem, we're given a start word and a target word, and we want to derive the target word from the start word by changing one letter at a time, such that the intermediate words are also English words.

TRAIN ← start word

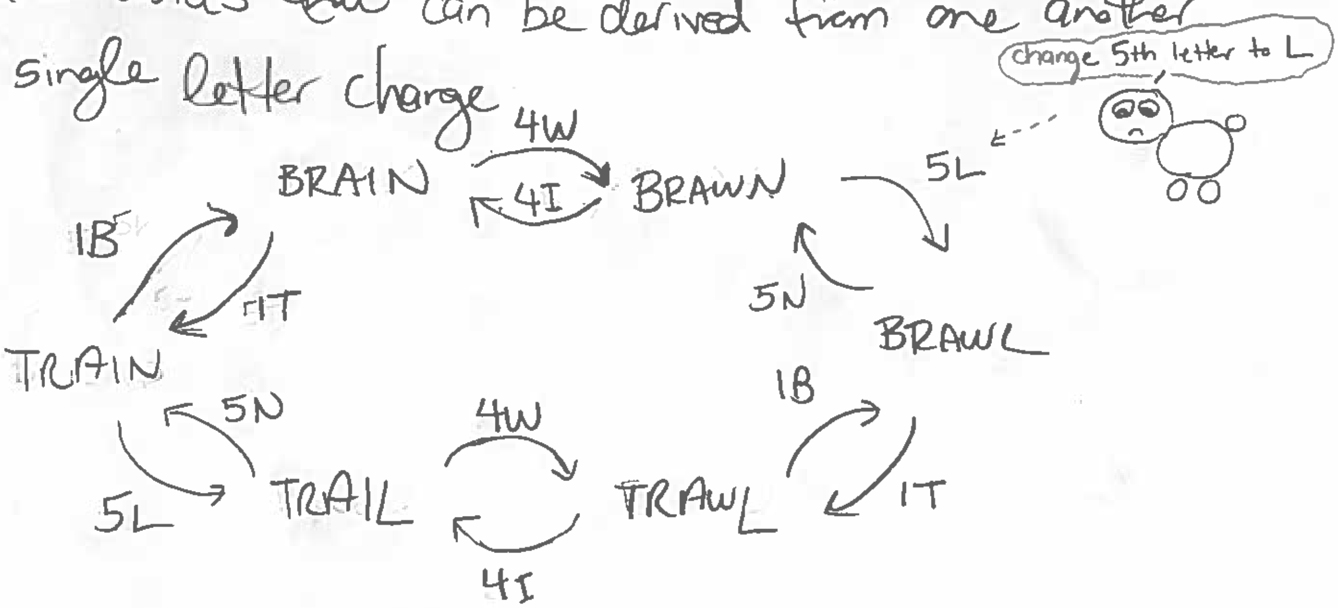
BRAIN

BRAWN

PRAWN ← target word

---

② We can represent the problem as a graph where the nodes are 5-letter words, and the (directed) edges are words that can be derived from one another using a single letter change



STATE SPACES

(3) This graph is an instance of a <u>state machine</u>, defined as a tuple $(Q, \Sigma, \Delta, q_0, F)$, where:

- Q is a set of <u>states</u>
- Σ is a set of <u>actions</u>
- $\Delta \subseteq Q \times \Sigma \times Q$ is a set of permitted <u>transitions</u>
- $q_0 \in Q$ is the <u>initial state</u>
- F is a set of <u>final (or goal) states</u>

unlike FSMs, we don't require Q to be finite

for example:

$Q = \{$ TRAIN, BRAIN, BRAWN, BRAWL, $\ldots \}$

$\Sigma = \{ 1A, 1B, 1C, \ldots, 5X, 5Y, 5Z \}$

(TRAIN, 1D, DRAIN) $\in \Delta$

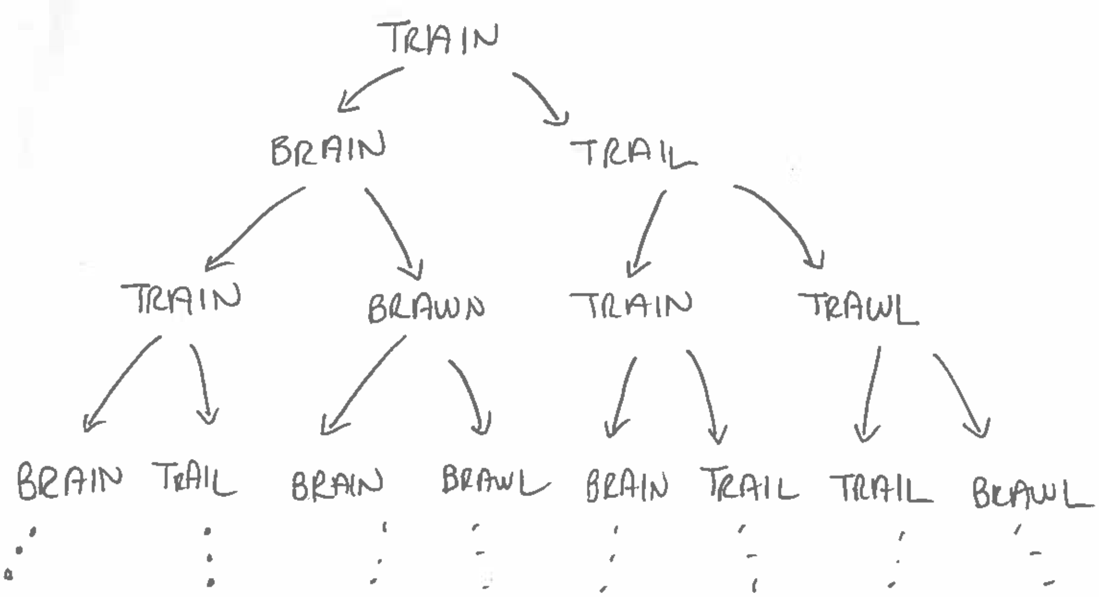
(BRAWL, 5N, BRAWN) $\in \Delta$

etc.

$q_0 =$ TRAIN

$F = \{$PRAWN$\}$

(4) We can expand a state machine into a <u>search tree</u>, where the root is q_0 and the children of a node q are the states q' such that $(q, \sigma, q') \in \Delta$ for some $\sigma \in \Sigma$.



State Spaces

(5) We will generally be interested in <u>weighted state machines</u>, which is a tuple $(Q, \Sigma, \Delta, q_0, F, w)$ where:

- $(Q, \Sigma, \Delta, q_0, F)$ is a state machine
- $w : \Delta \to \mathbb{R}$ assigns a real-valued weight (or cost) to each transition.

---

(6) In the word ladder problem, we usually assume that each transition costs 1, and so our goal of reaching the target word in a minimum number of steps can be expressed as follows.

Define a <u>search path</u> as a sequence $\langle d_0, \ldots, d_k \rangle$ of transitions $d_i \in \Delta$ where $d_i = (q_i, \sigma_i, q_{i+1})$ for $\sigma_i \in \Sigma$.

For instance, the solution in (1) corresponds to the path

$\langle$(TRAIN, 1B, BRAIN), (BRAIN, 4W, BRAWN), (BRAWN, 1P, PRAWN)$\rangle$

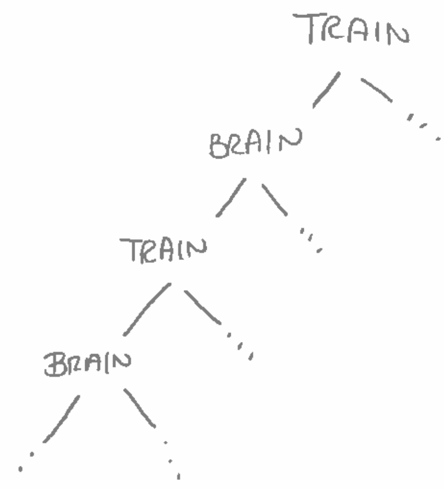
---

(7) The <u>cost</u> of a search path is the sum of the weights of its transitions. So if we set $w(\delta) = 1$ for all $\delta \in \Delta$, then the cost of a word ladder search path is just its length (i.e. the number of transitions).

STATE SPACES – M. Hopkins

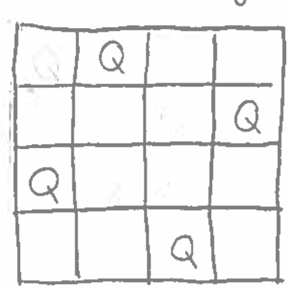
---

(8) If a state machine has cycles (as in our word ladder formulation), then the search tree will have infinite depth, e.g.



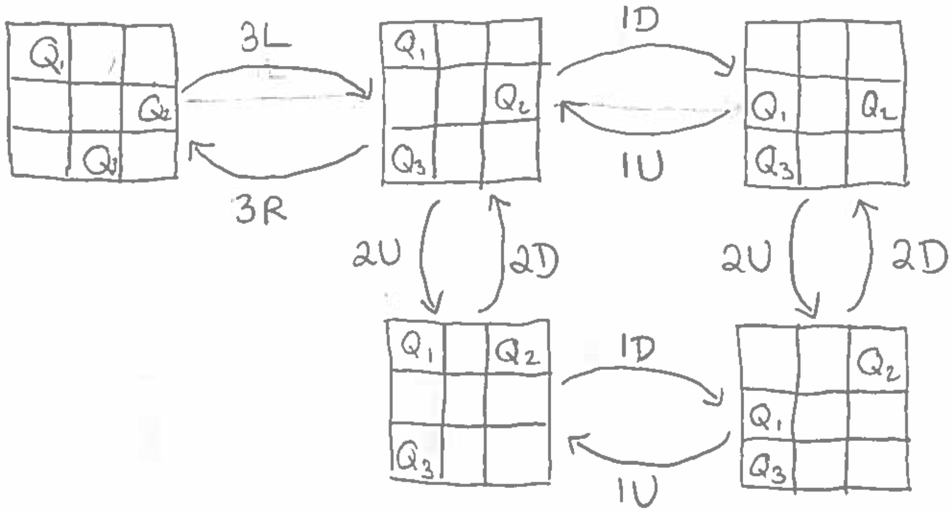
---

(9) Some search problems have state machine formulations that are acyclic. For instance, consider the n-queens problem, which asks us to place n queens on an n×n board so none can attack another, e.g.

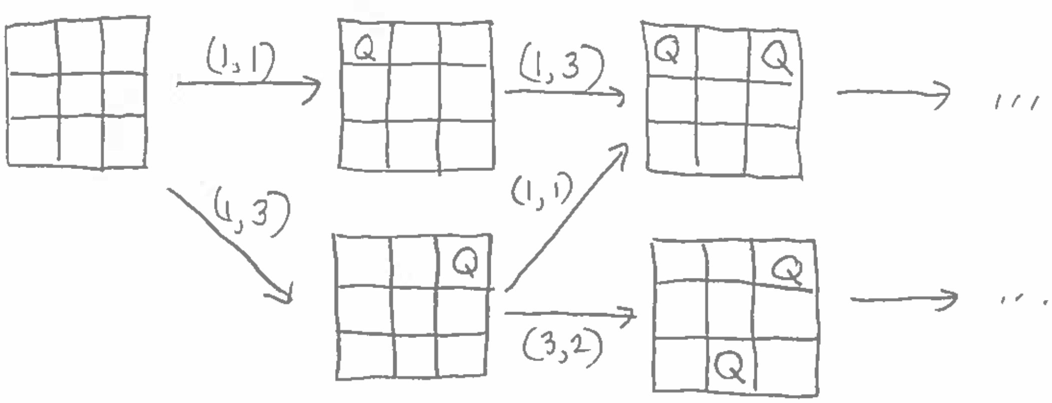


STATE SPACES – M. Hopkins

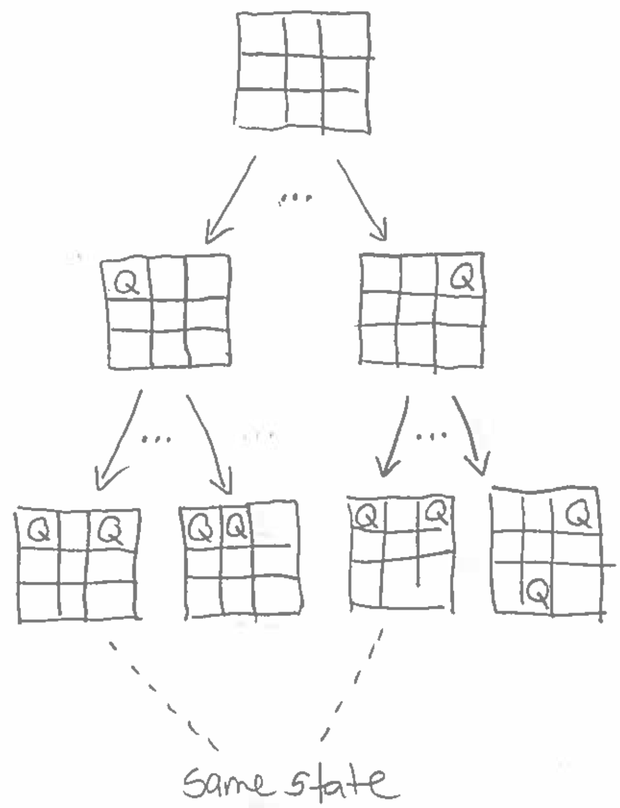
(10) One way to define a state machine for this problem is to let each state be a board configuration with n queens on it. Each action moves one of the queens one square:



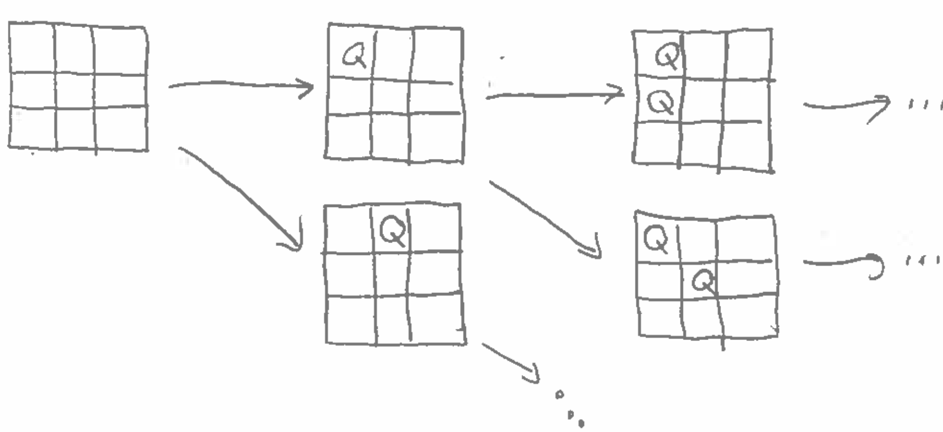
(11) This is cyclic, thus expands into an infinite search tree. Another state machine for this problem is to let each state be a board configuration with zero to n queens. Each action adds a queen to the board:



(12) This machine is acyclic, thus expands into a finite-depth search tree (of depth n). Note that this does <u>not</u> mean there are no repeated states, e.g.



(13) We can make either machine more efficient by only defining actions that don't immediately create a bad state. For instance, we can narrow the action to "add a queen to the topmost row without a queen."



STATE SPACES

(14) Now the machine has many fewer states, and even if we expand it to a search tree, there are no repeated states.

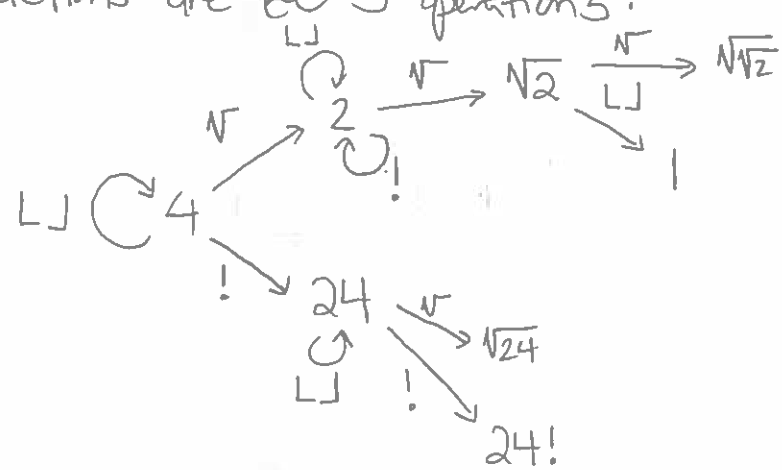
Fewer states is generally better, because there are less states to search through!

(15) So far we have only discussed finite state machines, but some search problems deserve state machines with an infinite number of states.

A conjecture by the Stanford professor Donald Knuth claims that any positive integer can be obtained by some sequence of factorial, square root, and floor operations applied to the number 4. For example,

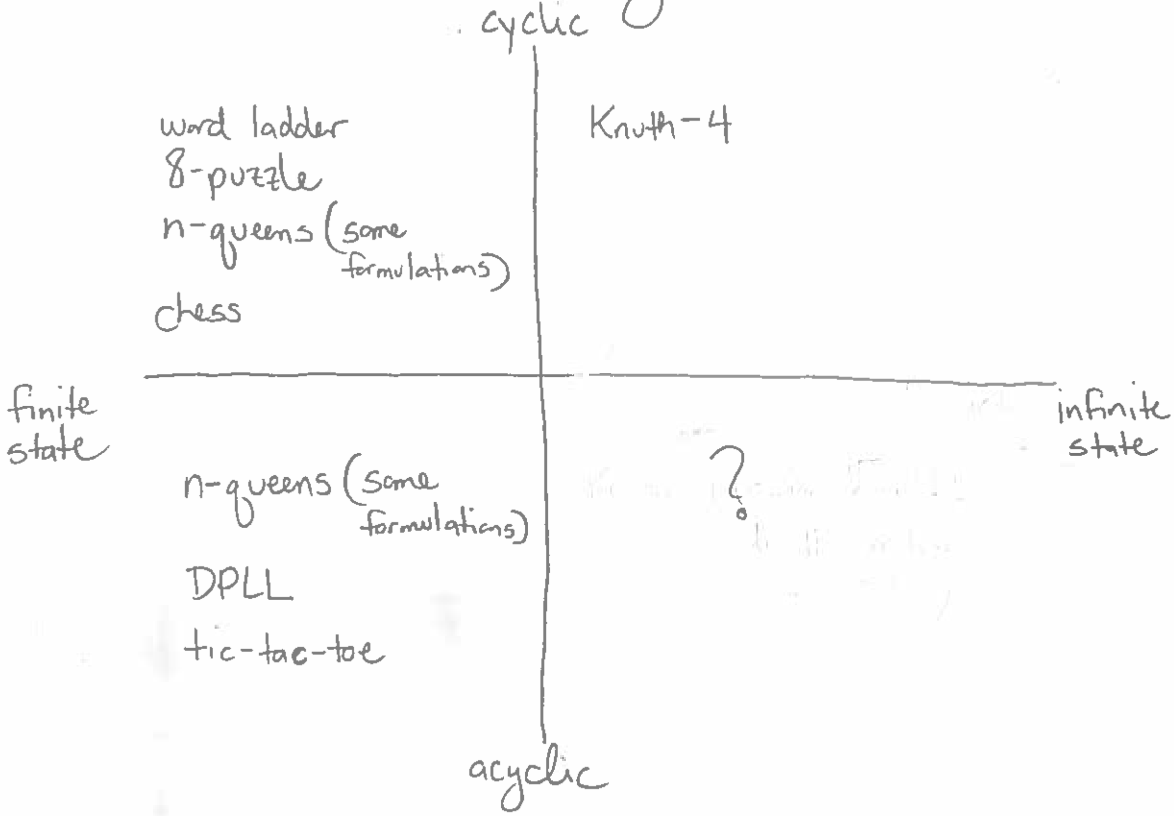
$$\left\lfloor \sqrt{\sqrt{\sqrt{\sqrt{\sqrt{4!!}}}}} \right\rfloor = 5$$

We can construct a state machine for this problem where the states are positive (real) numbers and the actions are the 3 operations:



STATE SPACES

(16) We can categorize search machines in terms of whether they are cyclic and whether they are finite-state:



State Spaces

Exercise (AIMA 3.5): Define a state machine for n-queens where the states are board configurations with k queens ($0 \le k \le n$) and the actions are "add a queen to any square in the leftmost empty column such that it is not attacked by another queen." Show that there are at least $\sqrt[3]{n!}$ reachable states from the start state (no queens on the board).

Consider the k^{th} column. When we place a queen there, $k-1$ queens have been already placed. Since each placed queen can only attack 3 (at most) column positions, at least $n-3(k-1)$ squares must be open. So the branching factor at depth k of the search tree is at least $n-3(k-1)$. Thus, the search tree up to depth $\frac{n}{3}+1$ has at least $n(n-3)(n-6)\cdots 1$ nodes

$$
\begin{aligned}
n(n-3)(n-6)\cdots &= \sqrt[3]{(n(n-3)(n-6)\cdots)^3} \\
&= \sqrt[3]{n^3(n-3)^3(n-6)^3\cdots} \\
&\geq \sqrt[3]{n(n-1)(n-2)(n-3)(n-4)(n-5)(n-6)(n-7)(n-8)\cdots 1} \\
&= \sqrt[3]{n!}
\end{aligned}
$$