The Context for Search

In order to plan ahead to find a solution, an intelligent agent goes through three distinct phases:

1. Formulate the goal.

   What is desired?

   For elective goals, this is not trivial. And it is something that our programs don't often worry about, because the goals are specified by the programmer!

2. Formulate the problem.

   What needs to be done?

   At this step, the agent defines what a state is, what the operators are, and what the goal test is.

3. Formulate the solution.

   What is the answer?

   Search through possible states for a path to a goal.

Formulating the Problem

Formulating a **problem** for search requires:

1. defining what states look like

   What information about a situation in the world must the agent keep track of in order to think about the effects of its actions?

2. defining what operations are possible (relevant) and how each affects the state of the world

3. defining how to tell when the search is done

   How does the agent know when it is done?

The resulting problem definition consists of:

- the **state space**

    - What states will be considered?
    - What actions will be considered?

    - What is the initial state?

- the **goal test**

    ... may just be “Is this state the goal?”

Formulating the Solution

Formulating a **solution** for search involves applying one of the search techniques we learned about last time.

Systematic search can favor...

• “old” states	*breadth*-first
• “new” states	*depth*-first

The other techniques that Ginsberg discusses in Chapter 3 are all variations of the same systematic search them that we developed in our last session.

- beam search
- depth-limited search
- iterative deepening search

An Exercise: Word Search as Search

```
J V P I A Z N Y I F B U E S S I P C V A
A L T J H Q K B K B V S T E G G U N D X
W W A R R I O R S K N E T V X K O S E D
S M E J A Q C U K U L C Y E M N N R M G
P H H P G I N C C L P K V N K I C R U C
Q W S K G S L L U B V E F T M C S D M O
X H Y A S I Y B B D B J R Y F K O T Z Z
W H M K P E X G L A O P T S C S I R I B
O A Y P G L F X L A H S X I O M H E J G
G O E V Z O H K E C Z S R X B N X H K B
F R L J Q Q J B L F A E R E F P I K O O
S C I H E C M B A Y V A R R I N I C N F
I T S Z B B S F T A E W V S O L Z P S Z
Z R C P Z Y G T M E O A T T R I A W K X
...
```

```
Bucks                Magic
...                  ...
Celtics              Pacers
...                  ...
Lakers               Warriors
```

Work in groups of three or four to **formulate the doing of a word search puzzle as a search problem**. Use the accompanying word search as a concrete example, but don’t make your solution specific to this problem.

Be sure to define the following:

- What is the **initial state** of the problem?
- What **operators** can the agent use to change state?

- How can you recognize **goal states**?

Uninformed Search

BFS and DFS are **uninformed** search strategies.

Uninformed search algorithms explore the states of a problem without knowing anything about the environment.

- BFS puts new states at the back of the line.
- DFS puts new states at the front of the line.

These algorithms work in the same way for every problem, because they do not consider any features that are specific to the problem being solved.

Not too surprisingly, BFS and DFS usually aren't the most intelligent ways for an agent to explore a state space. If the agent knows something about its environment, then it can almost certainly do better.

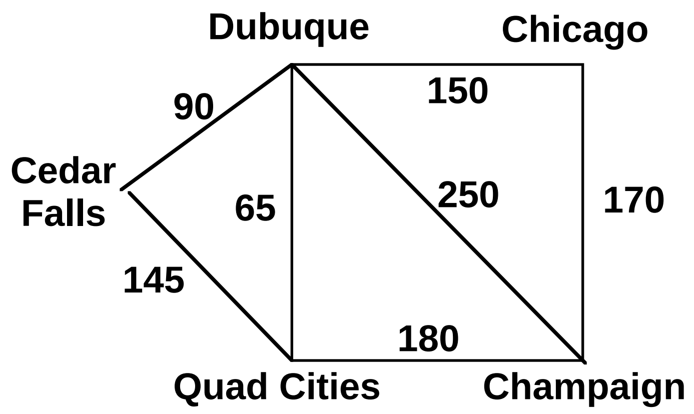
Using Knowledge of the Problem to Improve Search Performance

How can we use knowledge to improve search?

- To change the order in which we consider the states.
- To discard states from the list of states to consider.

One kind of knowledge useful in search is the cost of applying an operator. If my goal is to be in Champaign, Illinois, then there are costs associated with traveling to Cedar Rapids and Waverly. As we apply operators in sequence, we can compute the **path cost** associated with the sequence.

Consider our simple map from last time:

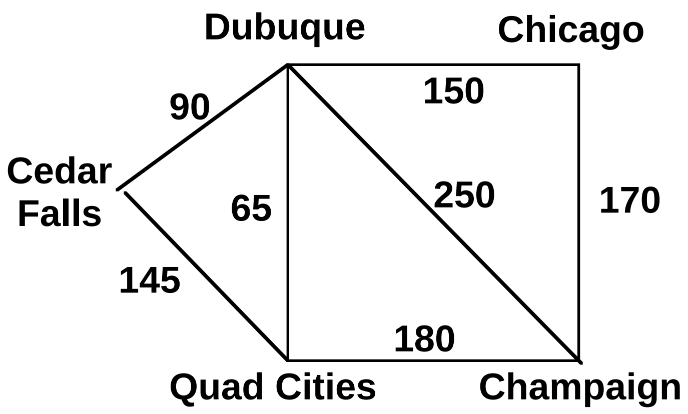


Knowledge of operator cost is problem-specific, because the operators are problem-specific. Any search algorithm that uses such costs is called *informed*.

Using Path Cost to Improve Search Performance

How can we use knowledge of path cost to improve search?

In **Uniform Cost Search** (UCS), we use a strategy that sorts the list in ascending order on path cost. The path cost of a node n is usually called `g(n)`.



Uniform Cost Search

UCS implements the same idea behind BFS (explore nodes close to the start state first to ensure completeness), but it uses a more practical definition of “close”.

Newly-expanded states will usually enter the list of states to consider near the back, with older nodes nearer the front. But a path of low real cost, even if it requires more steps, can leap-frog ahead of a path with fewer operators.

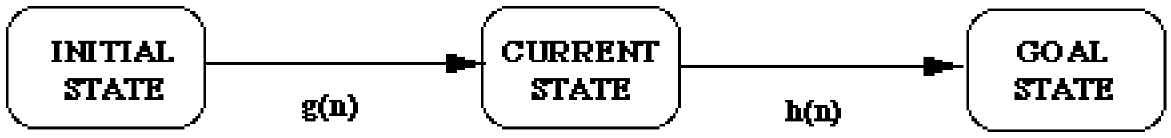
How does UCS stack up against our evaluation criteria?

completeness	guaranteed to find a solution if one exists
time	generally expensive for the same reasons as BFS, but often an improvement
space	generally expensive for the same reasons as BFS, but often an improvement
optimality	only if the “cost” of each operator > 0

The problem with UCS is that it is too much like BFS. We aren’t using much domain knowledge, and its effect on ordering the states is small.

Where to look next?

Other Ways to Use Path Cost



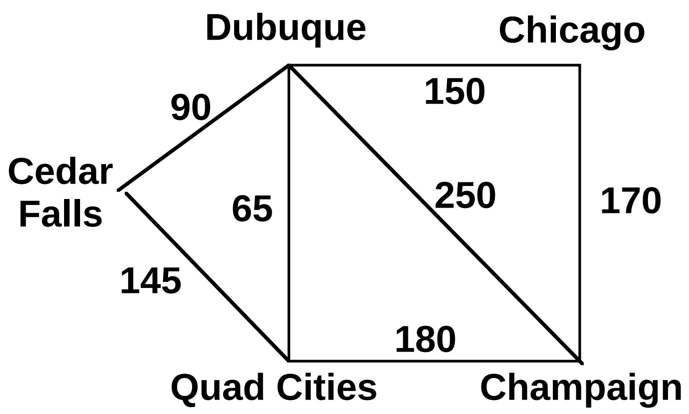
It turns out that, if my goal is to be in Champaign, Illinois, then there are **two costs** associated with traveling to Cedar Rapids, and **two costs** associated with traveling to Waverly. Cedar Rapids is farther away from Cedar Falls than is Waverly, *but its closer to Allerton Park.*

UCS uses g(n) to guide its search. g(n) measures the cost expended in getting from the initial state to the current state, the sunk cost of taking the path.

What about looking at what seems to be a more important measure: what will it cost to get from the current state to the goal?

Using Future Path Cost to Improve Search Performance

Greedy search orders the unexplored states in ascending order of their *expected future path cost*


Dubuque
Chicago
90
150
Cedar Falls
65
250
170
145
180
Quad Cities
Champaign