Template for Notes for CIS 5210 Midterm 1

The midterm exam for CIS 5210 will be **open notes**. No other matierals are allowed for the exam - you may not use the textbook, the internet, a Python environment, or communicate with anyone during the exam.

This python notebook is where you should save all of your notes. It provides a structured template that allows you to add summaries for each of the textbook sections that are part of the required reading, plus room for lecture notes and definitions.

Your summaries must be written in your own words. Do not copy and paste from the textbook. Do not share your notes with classmates, or copy any of your classmates' notes.

Instructions

1. Make a copy of this Google Colab file. From the colab menu, pick **File > Save a copy in Drive**. That will save a copy of this notebook in your Google drive.
2. Go through each section of the notebook, and fill in all the summaries for each chapter and its corresponding sections. Be sure to save your work frequently!
3. Read the notes below about how you can format your summaries.
4. Submit your version of this notebook to Gradescope before you start the exam.
5. If you are taking the exam in person, then you must print a copy of your notes for yourself. Here's how to print:

- Run the last cell of this notebook
- Click the "Mirror cell in tab" button (this is located in the upper right corner of the cell, immediately to the right of the trash can). This will open the cell.
- Copy and paste the contents of that cell into Google Docs or Microsoft Words.
- Use your word editor to save as a PDF that you can print.

Formatting Notes

1. The max length for each section's summary is 500 words / 3000 characters.
2. You can also write a 2000 word summary of the lectures for each module.
3. You may also write definitions of concepts for each module. The definitions should be written in your own words (not copy and pasted), and they should be no more than 20 words long for each concept.
4. You can use Markdown formatting for your summaries.
5. You can use Latex formatting for equations. Put two dollar signs before and after your equation like this:

$$\epsilon = \frac{1}{mc^2}$$

6. When you write python code or psuedocode algorithms, you should surround it with triple backticks like this ``` def function(): ... ``` This will nicely format you code like this.

```
def function():
     print('hello world')
```

6. Write \ whenever you want to keep writing on the next line without a newline being inserted.

Example summary

This is a sample summary of Section 2.1.1 of Jurfasky and Martin's Speech and Language Processing's book

```
chapter_2_1_example = """
Regular expressions are an algebraic notation for characterizing \
a set of strings that are useful for searching for a pattern or through a corpus.\
Only the first matching result will be shown. The search string is case sensitive, \
but it can consist of a single character or a sequence of characters. \
While square braces specify a disjunction of characters, the dash can more easily look \
for characters over a range. The caret specifies the non-existence of a character. \
Additionally, the (?) is used to for flexibility on a single character, meaning \
"the preceding character or nothing". More importantly, the Kleene star means zero or \
more occurrences of the immediately previous character or regular expression in square braces.\
...
"""
```

```
length = len(chapter_2_1_example.split(" "))
print("The example summary is {len} words long, so you could write an additional \
{remaining} words.".format(len=length, remaining=(500-length)))

    The example summary is 116 words long, so you could write an additional 384 words.
```

Example summary length

The maximum summary length for each section is 500 words / 3000 characters. Hopefully you'll find that to be quite generous.

```
# This is an example of a 500 word placeholder
placeholder = """ultrices vitae auctor eu augue ut lectus arcu bibendum at varius \
 vel pharetra vel turpis nunc eget lorem dolor sed viverra ipsum nunc aliquet \
 bibendum enim facilisis gravida neque convallis a cras semper auctor neque vitae \
 tempus quam pellentesque nec nam aliquam sem et tortor consequat id porta nibh \
 venenatis cras sed felis eget velit aliquet sagittis id consectetur purus ut \
 faucibus pulvinar elementum integer enim neque volutpat ac tincidunt vitae semper \
 quis lectus nulla at volutpat diam ut venenatis tellus in metus vulputate eu \
 scelerisque felis imperdiet proin fermentum leo vel orci porta non pulvinar neque \
 laoreet suspendisse interdum consectetur libero id faucibus nisl tincidunt eget \
 nullam non nisi est sit amet facilisis magna etiam tempor orci eu lobortis \
 elementum nibh tellus molestie nunc non blandit massa enim nec dui nunc mattis \
 enim ut tellus elementum sagittis vitae et leo duis ut diam quam nulla porttitor \
 massa id neque aliquam vestibulum morbi blandit cursus risus at ultrices mi tempus \
 imperdiet nulla malesuada pellentesque elit eget gravida cum sociis natoque \
 penatibus et magnis dis parturient montes nascetur ridiculus mus mauris vitae \
 ultricies leo integer malesuada nunc vel risus commodo viverra maecenas accumsan \
 lacus vel facilisis volutpat est velit egestas dui id ornare arcu odio ut sem nulla \
 pharetra diam sit amet nisl suscipit adipiscing bibendum est ultricies integer \
 quis auctor elit sed vulputate mi sit amet mauris commodo quis imperdiet massa \
 tincidunt nunc pulvinar sapien et ligula ullamcorper malesuada proin libero nunc \
 consequat interdum varius sit amet mattis vulputate enim nulla aliquet porttitor \
 lacus luctus accumsan tortor posuere ac ut consequat semper viverra nam libero \
 justo laoreet sit amet cursus sit amet dictum sit amet justo donec enim diam \
 vulputate ut pharetra sit amet aliquam id diam maecenas ultricies mi eget mauris \
 pharetra et ultrices neque ornare aenean euismod elementum nisi quis eleifend quam \
 adipiscing vitae proin sagittis nisl rhoncus mattis rhoncus urna neque viverra justo \
 nec ultrices dui sapien eget mi proin sed libero enim sed faucibus turpis in \
 eu mi bibendum neque egestas congue quisque egestas diam in arcu cursus euismod \
 quis viverra nibh cras pulvinar mattis nunc sed blandit libero volutpat sed cras \
 ornare arcu dui vivamus arcu felis bibendum ut tristique et egestas quis ipsum \
 suspendisse ultrices gravida dictum fusce ut placerat orci nulla pellentesque \
 dignissim enim sit amet venenatis urna cursus eget nunc scelerisque viverra mauris \
 in aliquam sem fringilla ut morbi tincidunt augue interdum velit euismod in \
 pellentesque massa placerat duis ultricies lacus sed turpis tincidunt id aliquet \
 risus feugiat in ante metus dictum at tempor commodo ullamcorper a lacus vestibulum \
 sed arcu non odio euismod lacinia at quis risus sed vulputate odio ut enim blandit \
 volutpat maecenas volutpat blandit aliquam etiam erat velit scelerisque in \
 dictum non consectetur a erat nam at lectus urna"""

print("The length in words is", len(placeholder.split(' ')))
print("The length in characters is", len(placeholder))

    The length in words is 498
    The length in characters is 3024
```

Permissions to use your summaries

We'd like to use your summaries to help develop better educational resources like remember.school or a question-answering system about the textbook.

1. Please enter your name.
2. If you grant permission for us to use your summaries under a Creative Commons License, please set `permission = True`. If you do not grant permission, please set it to false. Your answer won't affect your grade.
3. If you grant us permission, and you'd like to remain anonymous, set `anonymous = True`. We'll remove your name before sharing your summaries with anyone.

```
name = ""
permission = False
anonymous = True
```

List of Figures from AIMA

List of all figures and their URLs

You don't need to modify anything in this cell. Just run this cell by pressing the 'play' button to load in a list of URLs for images of the textbook figures.

Show code

Module 1 - Intro to AI

```
# Russell and Norvig, AIMA 4th Edition, Chapter 1 “Introduction”, Section 1.1 “What is AI”

chapter1_1 = """ Chapter 1.1 summary.

```
def my_method(arg_1, arg_2):
  for item in arg_1:
    print(item)

```

$ \sum_{n=0}^{N} n = b^2 $

"""

# Russell and Norvig, AIMA 4th Edition, Chapter 27 “Philosophical Foundations”, Sections 27.1 “The Limits of AI” and 27.2 “Can Mac
chapter27_1 = """
- weak ai: idea that machines could act as if they're intelligent
- strong ai: idea that machines that do so are consciously thinknig
- good old fasion ai (GOFAI): AI pursued in the cult of computationalism cannot produce results
- qualification problem: difficult to capture every contingency in a set of necessary and sufficiant logic rules
- an agent that only knows a dog(x) = mammal(x) is lacking from an agent that actually observes a dog
- embodided cognition approach: makes no sense to consider the brain seperatly
- Godel's incompleteness theorem:
  1.
  2. applys only math and not computers
- ask not if a machine can think but if it passes a Turing Test

"""
chapter27_2 = """ Your summary here """

# Russell and Norvig, AIMA 4th Edition, Chapter 2 “Intelligent Agents” (whole chapter)
chapter2_1 = """
  basically there are agents an precepts that control AI. \
  The agent you be the thing that directly interacts with the environment \
  and it sees percepts. An example would be the vaccum and \
  dirt in multiple dirty/clean rooms
  """


chapter2_2 = """
An agent would need to choose what to do given the performance measure \
  An agent's main goal is to maximize the perfomrance masure with is the expected \
    outcome from an action. An agent would need to analyze its environment in order to \
    find the right action to proceed with.
"""
chapter2_3 = """
  Next step for AI is to figure out the environment that the agent is in. \
    The environment has many different clssifications that are observabale, detemanistic \
      , episodic, sequential, and dynamic.
  """
chapter2_4 = """
- AI job is to design an agent program that implements agnet function
-

"""
```

```
# Lecture notes
module1_lecture_notes = """ Your summary here """


# Definitions of concepts from Module 1.
# You can define any term in the textbook,
# but you most use your own words for the definitions.
# Each definition should be at most 20 words long.
module1_definitions = {
"agent" : "anything that can perceive and act in an environment",
"percept" : "the content the agent percieves",
"agent function" : "maps any given percept sequence to action",
"rational agent" : "does action that maximizes the performance measure given the environment",
"consequentialism" : "evaluate based on consequences",
"task environments" : "problems where rational agents are the solution",
"PEAS" : "performance, environment, snensors, actuators ",
"Fully Observable" : "sensor has all aspects relavent to the choice of action",
"Deterministic" : "next state is determined by current state and action",
"Episodic" : "agent's experience is divided into atomic episodes, previos action doesnt determine next",
"Sequential" : "current decision could affect future ones",
"Dynamic" : "environment changes as agent is in action",
"Agent" : "architecture + program "

}

# You can include explanations for figures in the textbook (except figures with pseduocode).
# The figure will appear in your notes along with your explanation.
module1_figure_explanations = {
"Figure 1.3" : "Blocksworld was an early AI system that allowed users to say commands in natural language in order to have a robot
}
```

Preview of your notes for Module 1

Here's a preview of your how your notes will be displayed. At the end of this Python notebook, we'll aggregate your notes for all modules together into a single display.

Preview of your summary

You don't need to modify anything in this cell. Just press play.

Show code

Module 1: Rational Agents and Task Environments

AIMA Chapter 1.1 What is AI?

Chapter 1.1 summary.

```
def my_method(arg_1, arg_2):
  for item in arg_1:
    print(item)
```

$\sum_{n=0}^{N} n = b^2$

AIMA Chapter 27.1 The Limits of AI

- weak ai: idea that machines could act as if they're intelligent
- strong ai: idea that machines that do so are consciously thinknig
- good old fasion ai (GOFAI): AI pursued in the cult of computationalism cannot produce results
- qualification problem: difficult to capture every contingency in a set of necessary and sufficiant logic rules
- an agent that only knows a dog(x) = mammal(x) is lacking from an agent that actually observes a dog
- embodided cognition approach: makes no sense to consider the brain seperatly
- Godel's incompleteness theorem:
    1.
    2. applys only math and not computers
- ask not if a machine can think but if it passes a Turing Test

AIMA Chapter 27.1 Can Machines Really Think?

Your summary here

AIMA 2.1 Agents and Environments

basically there are agents an precepts that control AI. The agent you be the thing that directly interacts with the environment \ and it sees percepts. An example would be the vaccum and \ dirt in multiple dirty/clean rooms

AIMA 2.2 Good Behavior: The Concept of Rationality

An agent would need to choose what to do given the performance measure An agent's main goal is to maximize the perfomrance masure with is the expected outcome from an action. An agent would need to analyze its environment in order to \ find the right action to proceed with.

AIMA 2.3 The Nature of Environments

Next step for AI is to figure out the environment that the agent is in. The environment has many different clssifications that are observabale, detemanistic , episodic, sequential, and dynamic.

AIMA 2.4 The Structure of Agents

- AI job is to design an agent program that implements agnet function
-

Module 1 Lecture Notes

Your summary here

Module 1 Definitions

- **Agent** - anything that can perceive and act in an environment
- **Percept** - the content the agent percieves
- **Agent function** - maps any given percept sequence to action
- **Rational agent** - does action that maximizes the performance measure given the environment
- **Consequentialism** - evaluate based on consequences
- **Task environments** - problems where rational agents are the solution
- **Peas** - performance, environment, snensors, actuators
- **Fully observable** - sensor has all aspects relavent to the choice of action
- **Deterministic** - next state is determined by current state and action
- **Episodic** - agent's experience is divided into atomic episodes, previos action doesnt determine next
- **Sequential** - current decision could affect future ones
- **Dynamic** - environment changes as agent is in action
- **Agent** - architecture + program

Algorithms Learned: Tree Search:

Graph Search:

BFS:

- expand shallowest unexpanded node
- fronteir is FIFO queue
- is complete and will return a solution
- is optimal for unweighted
- use when infinite paths
- use when some solutions have short paths

- Time complexity: O(b^m)
- Space complexity: O(b^d)

DFS:

- expand deepest unexpanded node
- fronteir is LIFO
- put successors at front of frontier list
- complete only for final grpahs and not optimal
- Time: O(b^m)
- Space: O(b*m) -> linear space complexity
- use when space is constrained
- many possible solutions with long paths
- search can be fine tuned

Iterative DFS:

- DFS with a depth limit of l
- no infinite paths problems
- optimal if l = d
- if l < d then incomplete
- if l > d then not optimal
- Time Complexity: O(b^d)
- Space Complexity: O(b*d)
- optimal if step cost is 1

Djikstra:

Uniform Cost Search:

-

Greedy First:

A*:

Minmax/Alpha-Beta Pruning:

1. Start with current poss as MAX
2. Expand game tree a fixed number of ply
3. Apply eval func to leaf pos
4. calculate back up vals bottom up
5. Pick move assigned to MAX at the root
6. Wait for MIN to respond

- start at the bottom of the tree and calc the value of those nodes
- right after pick the max of the 2 children and then the min and keep alternationg
- pruning just cuts off a branch if you know the value won't satisfy the condition (max/min)
- order moves based on how likely they are so that pruning can occur
- alpha and beta are the best numbers they can achieve given the opponents move

AC-3

Optional - Python Notes

```
# No readings for this module.  Instead you can summarize the lecture instead.

# Python notes
python_notes = """ Your summary here """

# Each definition should be at most 20 words long.
python_definitions = {
"mutability" : "your definition here",
}
```

Preview of your summary

```
#@title Preview of your summary
#@markdown You don't need to modify anything in this cell. Just press play.

python_sections = [
  ("Python Notes", python_notes),
]

python_markdown = format_module(1, "Python",
                                python_sections,
                                python_definitions,
                                {})
display(Markdown(python_markdown))
```

You don't need to modify anything in this cell. Just press play.

Module 1: Python

Python Notes

Your summary here

Module 1 Definitions

- **Mutability** - your definition here

Module 2 - Search Problems

```
# Russell and Norvig, AIMA 4th Edition, AIMA Chapter 3 “Problem Solving by Search” (3.1-3.4)
chapter3_1 = """ 3.1 Problem Solving Agents
Goal Formulation: Define the objectives the agent wants to achieve
Problem Formulation: The agent defines an abstract world model; finds the states and actions necessary to reach the goal
Search: The agent simulates the outcomes of taking various states and actions in its model to try and find a sequence of actions t
Execution: The agent follows the sequence to reach the schools
In any fully observable, deterministic, known environment, the solution to any problem is a fixed sequence of actions
Open-loop control: The model does not modify its sequence/world model as it perceives statements
Closed-loop control: The model takes percepts from the environment and modifies its strategy (branching strategy in nondeterminist
Models are abstract mathematical descriptions of a set of states
Valid abstraction – solution in the abstracted space works in the original space
"""
chapter3_2 = """ Grid world: really similar model where agents can move from cell to cell in a grid and perform actions; Vacuum pu
Can have infinite state spaces; Donald Knuth’s 4 game can result in infinitely large numbers if you continue to apply factorials
Touring problems – a set of goal states must be visited; example: the traveling salesman problem (which is NP-hard)
VLSI layout, robot navigation, automatic assembly sequencing, protein design
"""
chapter3_3 = """ Forms a search tree to represent paths through the state space moving towards the solution
The search tree can have multiple paths to any given state (and so multiple nodes for); every node in the tree has a unique path k
At each state node (which will become a parent after expansion), we can expand the node by creating a child node for each resultir
Any state with a node has been reached; the frontier is the set of nodes that have been created but not explored (they exist in th
chapter3_4 = """
Breadth First Search
Complete on infinite search spaces
For each expansion, append unvisited children to the end of a FIFO queue (check if they are goals before pushing, early goal test)
Every intermediate state has an optimal path to it at any given point; this is reliant on the property that each action has equal
Time/Space complexity of O(b^d) – very memory hungry
This is because it has to store all previous states
Dijkstra’s/Uniform Cost Search
Best First Search, where path cost is the evaluation function
Complete and cost optimal
Time complexity of O(b^(1 + floor(C/e)))
C is the optimal cost
e is a lower bound on the cost of any action
This means that UCS is generally much slower than BFS
Depth First Search
Tree-like search (usually, can also be implemented as BestFS(-depth))
This means it does not keep a table of reach states
Non-optimal cost – just returns the first path
Keeps proceeding down to the deepest level it can reach via one path, then pops back up to the next deepest node that is not fully
Complete and efficient in finite tree-like state spaces
Can get stuck in infinite loops, and can go down infinite paths in infinite state spaces;
Incomplete in infinite state spaces
Time complexity is O(n) and Space is O(bm)
Very memory efficient (especially compared to BFS)
Backtracking Search: even more efficient—only store the current path and move back along your maintained stack and update new expl
```

```
 """


# Lecture notes
module2_lecture_notes = """
- if fully observable, determenistic, known environment the solution is a fixed sequence of actions \
- if partially observable, nondeteremnistic solution is based on what percept recieves \
-
Depth Limited DFS:
    Fixed Depth
    DFS but with a parameter l, such that every node at l is treated as if it has no successors
    TC: O(b^l) SC: O(bl); can be incomplete if l is too large/small
    Can use constant time cycle checking + depth constraint to eliminate most cycles
    Diameter of the state space (max distance between any two actions) can be a good choice for l
    Iterative Deepening
    Try the depth-limited search for every value of l up to a limit
    Memory: O(bd) with solution, O(bm) without
    Time: O(b^d) with solution, O(b^m) without
    Complete on any finite space with cycle detection
    Preferred with limited memory and unknown search depth

Bidirectional Search:
    Complete and Optimal
    Search from both the initial and goal state and meet in the middle
    Solution found when both frontiers meet
    Always expand from the node with minimum eval function value (from both sides combined)
    TC: O(b^(d/2)) SC: O(b^(d/2));


"""

# Definitions of concepts from Module 3.
# You can define any bolded term in the textbook,
# but you most use your own words for the definitions.
# Each definition should be at most 20 words long.
module2_definitions = {
"problem-solving agent" : "an agent that can formulate a plan via search",
"reflex-agent" : "chooses action based only on cuurent percept"
}



# You can include explanations for any figure in the textbook.
# The figure will appear in your notes along with your explanation.
module2_figure_explanations = {
"Figure 3.2" : "Your explanation here"
}
```

Preview of your summary

```
#@title Preview of your summary
#@markdown You don't need to modify anything in this cell. Just press play.
module2_sections = [
  ("AIMA Chapter 3.1 Problem-Solving Agents", chapter3_1),
  ("AIMA Chapter 3.2 Example Problems", chapter3_2),
  ("AIMA Chapter 3.3 Search Algorithms", chapter3_3),
  ("AIMA Chapter 3.4 Uninformed Search Strategies", chapter3_4),
  ("Module 2 Lecture Notes", module2_lecture_notes),
]

module2_markdown = format_module(2, "Search Problems",
                                 module2_sections,
                                 module2_definitions,
                                 module2_figure_explanations)
display(Markdown(module2_markdown))
```

You don't need to modify anything in this cell. Just press play.

Module 2: Search Problems

AIMA Chapter 3.1 Problem-Solving Agents

3.1 Problem Solving Agents Goal Formulation: Define the objectives the agent wants to achieve Problem Formulation: The agent defines an abstract world model; finds the states and actions necessary to reach the goal Search: The agent simulates the outcomes of taking various states and actions in its model to try and find a sequence of actions that reaches the goal Execution: The agent follows the sequence to reach the schools In any fully observable, deterministic, known environment, the solution to any problem is a fixed sequence of actions Open-loop control: The model does not modify its sequence/world model as it perceives statements Closed-loop control: The model takes percepts from the environment and modifies its strategy (branching strategy in nondeterministic/partially observable envs Models are abstract mathematical descriptions of a set of states Valid abstraction – solution in the abstracted space works in the original space

AIMA Chapter 3.2 Example Problems

Grid world: really similar model where agents can move from cell to cell in a grid and perform actions; Vacuum puzzle from prev chapter, sliding tile puzzle, and sokoban all can be modeled in grid world; 8-puzzle Can have infinite state spaces; Donald Knuth's 4 game can result in infinitely large numbers if you continue to apply factorials Touring problems – a set of goal states must be visited; example: the traveling salesman problem (which is NP-hard) VLSI layout, robot navigation, automatic assembly sequencing, protein design

AIMA Chapter 3.3 Search Algorithms

Forms a search tree to represent paths through the state space moving towards the solution The search tree can have multiple paths to any given state (and so multiple nodes for); every node in the tree has a unique path back to the root At each state node (which will become a parent after expansion), we can expand the node by creating a child node for each resulting state that can be created by taking an action; Any state with a node has been reached; the frontier is the set of nodes that have been created but not explored (they exist in the queue/stack)e

AIMA Chapter 3.4 Uninformed Search Strategies

Breadth First Search Complete on infinite search spaces For each expansion, append unvisited children to the end of a FIFO queue (check if they are goals before pushing, early goal test) Every intermediate state has an optimal path to it at any given point; this is reliant on the property that each action has equal cost Time/Space complexity of O(b^d) – very memory hungry This is because it has to store all previous states Dijkstra's/Uniform Cost Search Best First Search, where path cost is the evaluation function Complete and cost optimal Time complexity of O(b^(1 + floor(C/e))) C is the optimal cost e is a lower bound on the cost of any action This means that UCS is generally much slower than BFS Depth First Search Tree-like search (usually, can also be implemented as BestFS(-depth)) This means it does not keep a table of reach states Non-optimal cost – just returns the first path Keeps proceeding down to the deepest level it can reach via one path, then pops back up to the next deepest node that is not fully expanded Complete and efficient in finite tree-like state spaces Can get stuck in infinite loops, and can go down infinite paths in infinite state spaces; Incomplete in infinite state spaces Time complexity is O(n) and Space is O(bm) Very memory efficient (especially compared to BFS) Backtracking Search: even more efficient–only store the current path and move back along your maintained stack and update new explorations in-place; O(m) vs O(bm)

Module 2 Lecture Notes

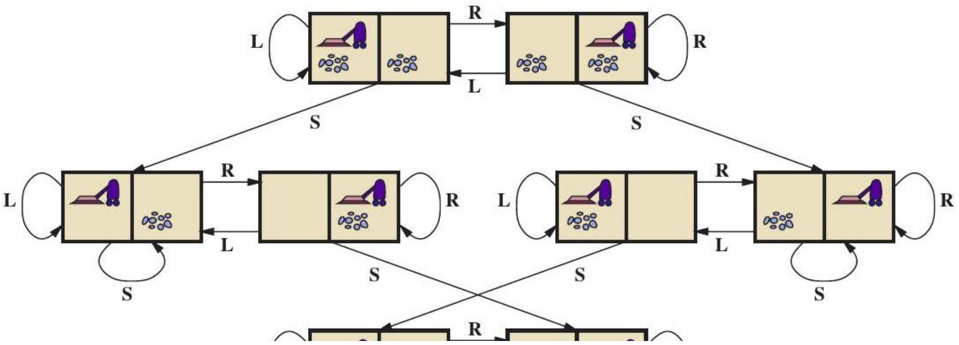
- if fully observable, determenistic, known environment the solution is a fixed sequence of actions - if partially observable, nondeteremnistic solution is based on what percept recieves - Depth Limited DFS: Fixed Depth DFS but with a parameter l, such that every node at l is treated as if it has no successors TC: O(b^l) SC: O(bl); can be incomplete if l is too large/small Can use constant time cycle checking + depth constraint to eliminate most cycles Diameter of the state space (max distance between any two actions) can be a good choice for l Iterative Deepening Try the depth-limited search for every value of l up to a limit Memory: O(bd) with solution, O(bm) without Time: O(b^d) with solution, O(b^m) without Complete on any finite space with cycle detection Preferred with limited memory and unknown search depth

Bidirectional Search: Complete and Optimal Search from both the initial and goal state and meet in the middle Solution found when both frontiers meet Always expand from the node with minimum eval function value (from both sides combined) TC: O(b^(d/2)) SC: O(b^(d/2));

Module 2 Definitions

- **Problem-solving agent** - an agent that can formulate a plan via search
- **Reflex-agent** - chooses action based only on cuurent percept

Module 2 Figures



Module 3 - Informed Search

```
# Russell and Norvig, AIMA Chapter 3 "Problem Solving by Search" (3.5-3.6)
chapter3_5 = """ A* Search
Best first search with eval fn f(n) = g(n) + h(n)
Cost up til now + estimated remaining cost
Complete in finite/infinite spaces
```

```
Optimality is conditional on heuristic properties
admissibility  — never overestimates the cost to the goal
Consistency — stronger version of admissibility; using the triangle inequality h(n)  <= c(n, a, n’) + h(n’)
Surely expanded nodes — If a reachable node is on a path such that every node on the path has f(n) < C*, it will be expanded by A*
Contours — like a topographic map; can circle the changing frontier based on the value of f(n)
A* is optimally efficient  because any other optimal algorithm would have to expand all nodes surely expanded by A*
A* is efficient because it prunes inefficient search paths from the tree
Number of nodes expanded can be exponential in the length of the solution
"""
chapter3_6 = """ Heuristic Quality
Effective Branching factor b* — Given A* generates N nodes, and the solution depth is d, b* is the branching factor for a uniform
Well designed heuristics have b* ~= 1
Effective Depth — find out the constant depth difference between an uninformed search and the informed search for some heuristic
If a heuristic is almost always better than another (by some index) we say that it dominates the other heuristic
Relaxed Problems
Common to get heuristic from a version of the problem with less rules
Because these only add edges, any real solution will be a solution in the relaxed space
ABSOLVER can generate these automatically for some problems
Pattern Databases
Store the cost of specific patterns in the problem to get the cost estimate of a subproblem efficiently
Ideally the subproblems are much cheaper to solve, and the patterns will be repeated for more efficiency add
Combine the multiple pattern heuristics into a single one (addition, max, min)
Landmark Storage
Precomputing the optimal cost between states in the graph can speed up the inference a lot
Take 10-20 “Landmark points” and compute distance from them to every other node
Heuristic will be inadmissible if an optimal path does not go through a landmark
Shortcuts — precomputed shorter paths in the graph
Differential heuristic
h_DH(n) = max (for all L) |C*(n,L) - C*(goal, L)|
Needs landmarks placed fairly well distributed
Meta-learning - works at the meta-level state space; can prevent algorithm from repeating undesirable algorithms
Chapter 5 - Adversarial Search
 """


# Lecture notes
module3_lecture_notes = """
UCS:
- assume step-cost = 1
- g(N) : the path cost function
    - is the depth in search tree
- extension of BFS
- frontier = priroty que using g(n)
- prioritize which nodes to go through

Best-First-Search:
- select node fro expansion w/ minimal evaluation function f(n)
- f(n) is function that includes estimate heuristic

Greedy best-first search:
- expands the node stimated to be closest to goal
- ignores g(n): cost to get to n
- example is Manhattan Dis, Euclidian
- can get stuck in loops, incomplete

A*:
- expand paths that are most promising first
- f(n) = g(n) + h(n)
- frontier is priority queue by increasing f(n)

Heuristic:
- is admissible if never overestimates the cost: optimistic

Dominance:
- if h2(n) >= h1(n): h2 dominates h1
- admissible(optimistic) if 0 <= h(n) <= h*(n)
"""

# Definitions
module3_definitions = {
"State Space" : "set of all states reachable from the initial state by sequence of actions ",
"Path" : "sequence of actions leading state s_j to s_k",
"Frontier" : "states that are available for expanding ",
"Solution" : "path leading from start to goal",
"Strategy" : "order of tree expansion",
"completeness" : "always finds solution",
"Optimality" : "Always fnds best solution",
"Time complexity" : "# of nodes generated",
"Space complexity" : "# of nodes in memory at same time",
"b" : "max branching factor",
"d" : "depth, shallowest goal node",
"m" : "max length of any path",
"heuristic" : "takes state n and returns estimate of dis form n to goal",
"Dominance" : "metric on better heuristics"
}


# You can include explanations for any figure in the textbook.
# The figure will appear in your notes along with your explanation.
module3_figure_explanations = {
"Figure 3.13" : ""
}
```

```
length = len(chapter3_6.split(" "))
print("The example summary is {len} words long, so you could write an additional \
{remaining} words.".format(len=length, remaining=(500-length)))
```

```
    The example summary is 241 words long, so you could write an additional 259 words.
```

Preview of your summary

```
#@title Preview of your summary
#@markdown You don't need to modify anything in this cell. Just press play.
module3_sections = [
  ("AIMA Chapter 3.5 Informed (Heuristic) Search Strategies", chapter3_5),
  ("AIMA Chapter 3.6 Heuristic Functions", chapter3_6),
  ("Module 3 Lecture Notes", module3_lecture_notes),
]

module3_markdown = format_module(3, "Informed Search",
                         module3_sections,
                         module3_definitions,
                         module3_figure_explanations)
display(Markdown(module3_markdown))
```

You don't need to modify anything in this cell. Just press play.

Module 3: Informed Search

AIMA Chapter 3.5 Informed (Heuristic) Search Strategies

A* Search Best first search with eval fn f(n) = g(n) + h(n) Cost up til now + estimated remaining cost Complete in finite/infinite spaces Optimality is conditional on heuristic properties admissibility – never overestimates the cost to the goal Consistency – stronger version of admissibility; using the triangle inequality h(n) <= c(n, a, n') + h(n') Surely expanded nodes – If a reachable node is on a path such that every node on the path has f(n) < C*, it will be expanded by A*; some nodes on the goal contour (f(n) = C*) may be expanded, but none past this are Contours – like a topographic map; can circle the changing frontier based on the value of f(n) A* is optimally efficient because any other optimal algorithm would have to expand all nodes surely expanded by A* A* is efficient because it prunes inefficient search paths from the tree Number of nodes expanded can be exponential in the length of the solution

AIMA Chapter 3.6 Heuristic Functions

Heuristic Quality Effective Branching factor b* – Given A* generates N nodes, and the solution depth is d, b* is the branching factor for a uniform tree of N+1 nodes with depth d Well designed heuristics have b* ~= 1 Effective Depth – find out the constant depth difference between an uninformed search and the informed search for some heuristic If a heuristic is almost always better than another (by some index) we say that it dominates the other heuristic Relaxed Problems Common to get heuristic from a version of the problem with less rules Because these only add edges, any real solution will be a solution in the relaxed space ABSOLVER can generate these automatically for some problems Pattern Databases Store the cost of specific patterns in the problem to get the cost estimate of a subproblem efficiently Ideally the subproblems are much cheaper to solve, and the patterns will be repeated for more efficiency add Combine the multiple pattern heuristics into a single one (addition, max, min) Landmark Storage Precomputing the optimal cost between states in the graph can speed up the inference a lot Take 10-20 "Landmark points" and compute distance from them to every other node Heuristic will be inadmissible if an optimal path does not go through a landmark Shortcuts – precomputed shorter paths in the graph Differential heuristic h_DH(n) = max (for all L) |C*(n,L) - C*(goal, L)| Needs landmarks placed fairly well distributed Meta-learning - works at the meta-level state space; can prevent algorithm from repeating undesirable algorithms Chapter 5 - Adversarial Search

Module 3 Lecture Notes

UCS:

- assume step-cost = 1
- g(N) : the path cost function
  - is the depth in search tree
- extension of BFS
- frontier = priroty que using g(n)
- prioritize which nodes to go through

Module 4 - Adversarial Search and Games

```
# Russell and Norvig, AIMA chapter 5 "Adversarial Search"”" (5.1-5.3, and 5.5).
chapter5_1 = """ Stances on other agents:
Economy – treat the actions of other agents as an aggregate effect rather than an individual one; works best in scenarios with mar
Part of the environment – misses the nuance of competing goals
Adversarial Game Tree – explicitly model the actions of an adverse agent
Two Player Zero-Sum Games
Define the game as such
Initial State S_0
To-Move(s): the player whose turn it is to move
Actions(s): the set of legal actions at state s
Result(s, a) -> Transition Model – what state do we get taking action a at state s?
Is-Terminal(s) – a test that determines whether the game has ended
Utility(s, p) – a function that determines the payoff to each player based on the terminal state
"""
chapter5_2 = """ MAX wants to win and MIN wants to stop them, so MAX must be able to conditionally branch on MIN’s decisions
For binary outcomes, use AND-OR, general solution is minimax
Ply - one player’s action
Minimax(s) – the utility of that state given both players play optimally from that state
Minimax is not always best if you know that you’re better than your opponent
Minimax Search
Recurse through the tree, and float the minimax values at each level up; then return the action from your action state that moves
O(b^m) TC; SC can be O(bm) or O(m)
Not practical in large search spaces
Multiplayer Games
Store utilities in a vector with length equal to # of players
Utility of node n is equal to the successor state vector which has the highest backed-up value for the player choosing at n
Alliances can implicitly form as a result of the optimal actions of multiple players aligning
 Alliances can be formed explicitly too and have certain rules; breaking alliances can be viewed negatively
In positive-sum games, there are many situations where collaboration is optimal
Alpha-Beta Pruning
Create a bound on the possible values of each node, based on what we know about the previous searches;
 only consider paths that have the possibility of changing the root decision (for example, if less than root value ignore)
Alpha – at least – highest for MAX, Beta – at most – lowest for MIN; these will eventually converge to be equal as more nodes are
A better ordering of actions can result in more effective pruning; up to O(b ^ (m/2) ) is possible with perfect ordering
Can inform move ordering based on what previous moves resulted in possibly better past situations; search for killer moves
Transposition Tables – store the heuristic/utility values of repeated states, to stop repeated searches
Type A - wide but shallow, then estimate utility (Chess)
Type B - follow a few promising paths far (Go, recently Chess)
 """
chapter5_3 = """ Replace terminal test with a cutoff test, using a heuristic function to approximate the ending utility from the c
```

```
Eval should be equal to Utility at terminal states, and between a loss and win at cutoffs;
Should be fast to compute but also be strongly correlated with winning
Ex: Expected value based on proportion of outcomes possible from state
Can use a linear combination with independent features, nonlinear otherwise
For AlphaBeta, iterate depth of search, and have cutoff happen at depth limit; however, we should consider quiescence— only cutofi
Horizon effect — when the agent delays inevitable negative effects past the depth limit, erroneously thinking it has prevented the
Singular Extensions — keep exploring moves that are clearly better at a certain node, past the cutoff point
Forward Pruning
Can sometimes accidentally prune away the best option
Beam Search — take the n best moves (based on heuristic) rather than all possible moves
ProbCut — nondeterministic alpha-beta; prunes if it its likely outside the alpha-beta window
Late move reduction — assume move ordering is good;reduce depth of search for later moves
 """
chapter5_5 = """ Your summary here """

# Russell and Norvig, AIMA Chapter 16 “Making Simple Decisions” (16.1-16.3)
chapter16_1 = """ 16.1 Combining Beliefs and Desires under Uncertainty
P(Result(a) = s`) = Sum_(s) P(s)P(s’|s, a)
EU(a) = Sum_(s`) P(Result(a) = s`)U(s’)
Where EU is expected Utility and U is utility
The agent will work to maximize the utility function, so if the utility function matches the goal, it will be very good
Utility function for next action, performance measure over past actions



 """
chapter16_2 = """ Your summary here """
chapter16_3 = """ Your summary here """

# Lecture notes
module4_lecture_notes = """
- MAX and MIN, MAX moves first
- take turns till game over
- winner gets award

Minimax:
- make best move for MAX
- backed-up val of each node in tree os deteremined by value of children
- max: backed up val is max of the values
- min: backed up value is min of values
 1. Start with current poss as MAX
 2. Expand game tree a fixed number of ply
 3. Apply eval func to leaf pos
 4. calculate back up vals bottom up
 5. Pick move assigned to MAX at the root
 6. Wait for MIN to respond
 def max-value(state):
       if the state is a terminal state:
        return the state’s utility
   initialize v = -∞
   for each successor of state:
     v = max(v, min-value(successor))
   return v
def min-value(state):
       if the state is a terminal state:
        return the state’s utility
   initialize v = +∞
   for each successor of state:
     v = min(v, max-value(successor))
   return v

Alpha-Beta Pruning:
- cut of stems based on if val will be reached
- replace alpha(max): max's best option
- replace beta(min): min's best option

Expectimax Search: compute avg score under optimal play
- have max nodes and chance nodes
    - chance nodes are min nodes but outcome is uncertain
    - calc their expected utils
"""

# Definitions
module4_definitions = {
"concept" : "definition",
}

# Figures with explanlations
module4_figure_explanations = {
```

```
module4_figure_explanations = {
"Figure 5.5" : ""
}
```

Preview of your summary

You don't need to modify anything in this cell. Just press play.

```
#@title Preview of your summary
#@markdown You don't need to modify anything in this cell. Just press play.
module4_sections = [
  ("AIMA Chapter 5.1 Game Theory", chapter5_1),
  ("AIMA Chapter 5.2 Optimal Decisions in Games", chapter5_2),
  ("AIMA Chapter 5.3 Heuristic Alpha—Beta Tree Search", chapter5_3),
  ("AIMA Chapter 5.5 Stochastic Games", chapter5_5),
  ("AIMA Chapter 16.1 Combining Beliefs and Desires under Uncertainty", chapter16_1),
  ("AIMA Chapter 16.2 The Basis of Utility Theory", chapter16_2),
  ("AIMA Chapter 16.3 Utility Functions", chapter16_3),
  ("Module 4 Lecture Notes", module4_lecture_notes),
]

module4_markdown = format_module(4, "Adversarial Search and Games",
                         module4_sections,
                         module4_definitions,
                         module4_figure_explanations)
display(Markdown(module4_markdown))
```

Module 4: Adversarial Search and Games

AIMA Chapter 5.1 Game Theory

Stances on other agents: Economy – treat the actions of other agents as an aggregate effect rather than an individual one; works best in scenarios with many agents Part of the environment – misses the nuance of competing goals Adversarial Game Tree – explicitly model the actions of an adverse agent Two Player Zero-Sum Games Define the game as such Initial State S_0 To-Move(s): the player whose turn it is to move Actions(s): the set of legal actions at state s Result(s, a) -> Transition Model – what state do we get taking action a at state s? Is-Terminal(s) – a test that determines whether the game has ended Utility(s, p) – a function that determines the payoff to each player based on the terminal state

AIMA Chapter 5.2 Optimal Decisions in Games

MAX wants to win and MIN wants to stop them, so MAX must be able to conditionally branch on MIN's decisions For binary outcomes, use AND-OR, general solution is minimax Ply - one player's action Minimax(s) – the utility of that state given both players play optimally from that state Minimax is not always best if you know that you're better than your opponent Minimax Search Recurse through the tree, and float the minimax values at each level up; then return the action from your action state that moves to the maximum (or minimum) value O(b^m) TC; SC can be O(bm) or O(m) Not practical in large search spaces Multiplayer Games Store utilities in a vector with length equal to # of players Utility of node n is equal to the successor state vector which has the highest backed-up value for the player choosing at n Alliances can implicitly form as a result of the optimal actions of multiple players aligning Alliances can be formed explicitly too and have certain rules; breaking alliances can be viewed negatively In positive-sum games, there are many situations where collaboration is optimal Alpha-Beta Pruning Create a bound on the possible values of each node, based on what we know about the previous searches; only consider paths that have the possibility of changing the root decision (for example, if less than root value ignore) Alpha – at least – highest for MAX, Beta – at most – lowest for MIN; these will eventually converge to be equal as more nodes are discovered A better ordering of actions can result in more effective pruning; up to O(b ^ (m/2)) is possible with perfect ordering Can inform move ordering based on what previous moves resulted in possibly better past situations; search for killer moves Transposition Tables – store the heuristic/utility values of repeated states, to stop repeated searches Type A - wide but shallow, then estimate utility (Chess) Type B - follow a few promising paths far (Go, recently Chess)

AIMA Chapter 5.3 Heuristic Alpha–Beta Tree Search

Replace terminal test with a cutoff test, using a heuristic function to approximate the ending utility from the cutoff state Eval should be equal to Utility at terminal states, and between a loss and win at cutoffs; Should be fast to compute but also be strongly correlated with winning Ex: Expected value based on proportion of outcomes possible from state Can use a linear combination with independent features, nonlinear otherwise For AlphaBeta, iterate depth of search, and have cutoff happen at depth limit; however, we should consider quiescence– only cutoff if there are no pending moves (such as captures) otherwise keep looking Horizon effect – when the agent delays inevitable negative effects past the depth limit, erroneously thinking it has prevented the effects Singular Extensions – keep exploring

▾ Module 5 - "Constraint Satisfaction Problems"

```
# Russell and Norvig, AIMA, Chapter 6 “Constraint Satisfaction Problems” (6.1-6.5)
chapter6_1 = """ Your summary here """
chapter6_2 = """ Your summary here """
chapter6_3 = """ Your summary here """
chapter6_4 = """ Your summary here """
chapter6_5 = """ Your summary here """


# Lecture notes
module5_lecture_notes = """
What is Search For:
    Assumptions about the worlds:
    Single Agent
    Deterministic actions
    Observed state
    Discrete state space
    Planning: sequence of actions
    Path to the goal is important
Paths have various costs & depths
Heuristics give problem specific guidance (help speed up search
Identification:
Goal is important, not the path
All paths have same depth
Constraint satisfaction problems are specialized for identification

Big Idea:
    Represent the constraints that solutions must satisfy in uniform declarative language
    Find solutions by general purpose search algos with no change from problem to problem
    No hand-built transition functions
    No hand built heuristics
    Just specify the problem in a formal declarative language and the general-purpose algo does the rest

CSP Definition:
Consists of
Finite set of variables X1, X2, … Xn
non empty domain of all possible values for each variable D1, … Dn where Di  = {v1, …., vk
Finite set of constraints
Each constraint has a limit ofr the values variables can take
A state is the assignment of values to some variables
A consistent assignment doesn’t violate the constraints like in Sudoku
Assignment is complement when all variables have a value
```

```
Solution is complete and consistent
Solutions can be found w/ complete general purpose algorithms

Benefits of CSP
Clean specification of many problems, generic goal, successor function & heuristics
CSP “knows” which variables violate a constraint
Knows where to focus search
Automatically prune off all branches that violate the constraints

Representations:
    Constraint graph
    Nodes are variables
    Arcs are binary constraints
    Standard representation pratten:
    Variables with values
    Constraint graph simplifies search
    This problem is binary CSP

Varieties:
    Discrete variables
    Finite domains
    N variables, domain size d -> O(d^n) complete assignments
    Not all gonna be solutions
    Infinite domains
    Integers, strings
    E.g Job scheduling

Idea 1: CSP as a search problem:
Can be expressed as a search problem,
Initial State: the empty assignment {}
Successor function: Assign value to any unassigned variable provided that there is not constraint conflict
Goal test: current assignment is complete
Path cost: constant cost for each step
Solution is always found at depth n for n variable
DFS can be used
Search and branching factor
N variables of domain size d
Branching factor at root is n*d
Branching factor at next level is (n - 1) * d
Tree has n! * d^n
Variable assignments are commutative
Only need to consider assignments to a single variable at each node

Searching and Backtracking:
DFS for CSPs with variable assignments is called backtracking search
Go down a path and if you reach a wong end, go back to previous move and continue from there
Backtracking search is basic uninformed algo for CSP

Improving backtracking efficiency:
General purpose methods & general-purpose heuristics can give huge gains in speed
Heuristics:
Q: which value should be assigned next:
1 most constrained variable
If ties: most constraining variable
Which order should variable values be tried:
Least constraining value
Can we detect inevitable failure early:
Forward checking

Towards Constraint Propagation:
Forward checking propagates info from assigned to unassigned variables but does provide detection for all failures
Constraint propagation goes beyond forward checking & repeatedly enforces constraints locally

Idea 3(big idea): inference in CSP:
CSP combines search and inference
Search
Constraint propagation (inference)
Eliminates possible values for a variable if the value would violate local consistency
Can do inference first, intertwine with search
Local Consistency:
Node consistency: satisfies unary constraints
Is trivial
Arc consistency: satisfies binary constraints
Xi is arc-consistent with respect to xj  if for every value v in Dj there is some value w in Dj that satisfies the binary constrai
Convert an edge into an arc from i -> j and j -> i
X -> Y is consistent if for every value x of x there is some allowed y
```

```
"""

# Definitions
module5_definitions = {
"concept" : "definition",
}

# Figures with explanlations
module5_figure_explanations = {
"Figure 6.4" : ""
}
```

Preview of your summary

You don't need to modify anything in this cell. Just press play.

Show code

Module 5: Constraint Satisfaction Problems

▾ Module 6 - Logical Agents

```
# Russell and Norvig, AIMA Chapter 7 “Logical Agents” (Sections 7.1-7.5)

chapter7_1 = """
- knowledge based agents use reasoning over internal representation to choose next action
- can accept new tasks in the form of described goals
- achieve competence by being told about the enviornment and adapt to changes
- knowledge base: a sentence
- knowledge language representation: each sentence is representing an aspect of the world
- Axiom: sentence that is being taken as given w/out being derived
- adding to knowedge base is called Tell and Ask
- inference: deriving new sentences from old
- usually operates in 3 steps:
    1. tells knowledge base what it percieves
    2. Asks what action it should perform
    3. Tells the knowledge base what action it performed
- declarative: start with empty KB and Tell it sentences until it percieves its enviornment
- procedural: encodes desired behaviors directly
"""
chapter7_2 = """ Your summary here """
chapter7_3 = """
- syntax: the way sentences of a language are expressed
- semantics: meaning of a language with respect to a possible world
- entailment: logic that a sentence follows logically from another sentence
*** ADD MATHEMATICAL REPRESENTATIONS ***
- truth/sound-preserving: inference album that derives only entitled sentences
- completeness: complete when can derive any sentence that is entitled
- grounding: connection between logical reasoning processes and real enviornment

"""
chapter7_4 = """ Your summary here """
chapter7_5 = """ Your summary here """

# Lecture notes
module6_lecture_notes = """

Knowledge-based Agents: use a process of reasoning over internal representation of knowledge to figure out next action/
So far the representation of states has been atomic

Knowledge Base:
A sentence written in a knowledge representation language
Sentence contains an assertion of the world
Axiom: sentence that is given
Derived sentence: a new sentence derived from others
Inference: process of deriving new sentences from old
KB initially contains background information and KB agane t adds info through observations of the world

Hunt the Wumpas/Wampas:
KB agent: R2D2 that explores a cave consisting of different rooms
Environment: 4x4 grid of rooms, agent starts at [1, 1]
Performance measure:
+1000 for rescuing Luke
-1000 for falling into a pit
-1 for each action
-10 for using blaster fire
Actuators:
R2 can move forward, left, right
Grab to pick up gold
Shoot to fire bolt
Sensors:
Sense of smell, has a stench if they’re cells next to the wampa
Pit will have a breeze that R2 can sense.
Wampa World
Partially observable: cannot see whole world just based of stench and breeze
Sequential
Deterministic, discrete, static, single-agent
Logic:
Can serve as general clas of representations for KB agents
KB consists of sentences in representation language
Representation language has a syntax that specifies sentences
A logic defines semantics
```

```
Semantics defines the truth w/ respect of possible world

Possible worlds & Models:
Model: mathematical abstractions that have fixed set of truth values which are {true, false} for each sentence
Of sentence a is true in model we say
M satisfies a or m is a model of a and use notation M(a) as set of all models of a
Logical Entailment:
Perform logical reasoning once we know whats true
Entailment is the idea thAt a sentence follows logically from another sentence
To mean a entails B we do a |= B if and only if M(a) is in M(B)
Means that a is more specific, or stronger than B.

Logical Inference: entailment applied to derive conclusions
Design inference algorithms to enumerate these sentences
Atomic Sentence: represented w/ single propositional symbol
Propositional symbol: stand for statement that can be t/f
Complex sentences: are constructed from simpler ones using parentheses and logical connectives
Conjunction: true if both of the sentences are true
Conditional:
If you assume P is true can you then get that Q is true

Part 2:
Sound: the inference algorithm should only derive entailed sentences
Complete: an inference algo’s is complete if it can derive all sentences that are entailed

Theorem Proving:
Logically equivalent: they are true in the same set of models
Sentence a and B are logically equivalent if and only if they entail each other

Symbols:
- ¬ (not) : ¬W1,3 is the negation of W1,3
- ∧ (and): W1,3 ∧ P3,1 is called a conjunction
- ∨ (or): W1,3 ∨ P3,1 is called a disjunction
- ⟹ (implies): W1,3 ⟹ S1,2 is called an implication. W1,3 is its premise or antecede and S1,2 is its conclusion or consequence
- ⇔ (if and only if): W1,3⇔¬W3,4 is called an biconditional

Logical Equivalence
- (α ⋀ β) ≡ (β ⋀ α): Commutativity of ∧
- (α⋁β) ≡ (β⋁α): communitivity of ∨
- ((α⋀β)⋀ɣ) ≡ (α⋀(β⋀ɣ)) : Associativity of ^
- ((α⋁β)⋁ɣ) ≡ (α⋁(β⋁ɣ)) : Associativity of v
- ¬(¬α) ≡ α : double negative association
- (α⟹β) ≡ (¬β ⟹ ¬α): Contraposition
- (α⟹β) ≡ (¬α⋁β): Implication elimination
- (α⇔β) ≡ ((α⟹β)⋀(β⟹α)) : Biconditional elimination
- ¬(α⋀β) ≡ (¬α ⋁ ¬β): De Morgan
- ¬(α⋁β) ≡ (¬α ⋀ ¬β): De Morgan
- (α⋀(β⋁ɣ)) ≡ ((α⋀β)⋁(α⋀ɣ))): Distributiviy of ^ over v
- (α⋁(β⋀ɣ)) ≡ ((α⋁β)⋀(α⋁ɣ))): Distributivity of ∨ over ∧

Monoticity implies soundness:
- set of entailed sentence can only increase as information is assesd to KB

"""

# Definitions
module6_definitions = {
"concept" : "definition",
}


# Figures with explanlations
module6_figure_explanations = {
"Figure 7.8" : ""
}
```

▸ Preview of your summary

You don't need to modify anything in this cell. Just press play.

Show code

Module 6: Logical Agents

AIMA Chapter 7.1 Knowledge-Based Agents

- knowledge based agents use reasoning over internal representation to choose next action
- can accept new tasks in the form of described goals
- achieve competence by being told about the enviornment and adapt to changes
- knowledge base: a sentence
- knowledge language representation: each sentence is representing an aspect of the world
- Axiom: sentence that is being taken as given w/out being derived
- adding to knowedge base is called Tell and Ask
- inference: deriving new sentences from old
- usually operates in 3 steps:
    1. tells knowledge base what it percieves
    2. Asks what action it should perform
    3. Tells the knowledge base what action it performed
- declarative: start with empty KB and Tell it sentences until it percieves its enviornment
- procedural: encodes desired behaviors directly

AIMA Chapter 7.2 The Wumpus World

Your summary here

AIMA Chapter 7.3 Logic

- syntax: the way sentences of a language are expressed
- semantics: meaning of a language with respect to a possible world
- entailment: logic that a sentence follows logically from another sentence
- ** ADD MATHEMATICAL REPRESENTATIONS ***
- truth/sound-preserving: inference album that derives only entitled sentences
- completeness: complete when can derive any sentence that is entitled
- grounding: connection between logical reasoning processes and real enviornment

AIMA Chapter 7.4 Propositional Logic: A Very Simple Logic

Your summary here

AIMA 7.5 Propositional Theorem Proving

Your summary here

Module 6 Lecture Notes

Knowledge-based Agents: use a process of reasoning over internal representation of knowledge to figure out next action/ So far the representation of states has been atomic

Knowledge Base: A sentence written in a knowledge representation language Sentence contains an assertion of the world Axiom: sentence that is given Derived sentence: a new sentence derived from others Inference: process of deriving new sentences from old KB initially contains background information and KB agane t adds info through observations of the world

Hunt the Wumpas/Wampas: KB agent: R2D2 that explores a cave consisting of different rooms Environment: 4x4 grid of rooms, agent starts at [1, 1] Performance measure: +1000 for rescuing Luke -1000 for falling into a pit -1 for each action -10 for using blaster fire Actuators: R2 can move forward, left, right Grab to pick up gold Shoot to fire bolt Sensors: Sense of smell, has a stench if they're cells next to the wampa Pit will have a breeze that R2 can sense. Wampa World Partially observable: cannot see whole world just based of stench and breeze Sequential Deterministic, discrete, static, single-agent Logic: Can serve as general clas of representations for KB agents KB consists of sentences in representation language Representation language has a syntax that specifies sentences A logic defines semantics Semantics defines the truth w/ respect of possible world

Possible worlds & Models: Model: mathematical abstractions that have fixed set of truth values which are {true, false} for each sentence Of sentence a is true in model we say M satisfies a or m is a model of a and use notation M(a) as set of all models of a Logical Entailment: Perform logical reasoning once we know whats true Entailment is the idea thAt a sentence follows logically from another sentence To mean a entails B we do a I= B if and only if M(a) is in M(B) Means that a is more specific, or stronger than B.

Logical Inference: entailment applied to derive conclusions Design inference algorithms to enumerate these sentences Atomic Sentence: represented w/ single propositional symbol Propositional symbol: stand for statement that can be t/f Complex sentences: are constructed from simpler ones using parentheses and logical connectives Conjunction: true if both of the sentences are true Conditional: If you assume P is true can you then get that Q is true

Part 2: Sound: the inference algorithm should only derive entailed sentences Complete: an inference algo's is complete if it can derive all sentences that are entailed

Theorem Proving: Logically equivalent: they are true in the same set of models Sentence a and B are logically equivalent if and only if they entail each other

Symbols:

- ¬ (not) : ¬W1,3 is the negation of W1,3
- ∧ (and): W1,3 ∧ P3,1 is called a conjunction
- ∨ (or): W1,3 ∨ P3,1 is called a disjunction
- ⟹ (implies): W1,3 ⟹ S1,2 is called an implication. W1,3 is its premise or antecede and S1,2 is its conclusion or consequence
- ⇔ (if and only if): W1,3⇔¬W3,4 is called an biconditional

Logical Equivalence

- $(\alpha \bigwedge \beta) \equiv (\beta \bigwedge \alpha)$: Commutativity of ∧
- $(\alpha \bigvee \beta) \equiv (\beta \bigvee \alpha)$: communitivity of ∨
- $((\alpha \bigwedge \beta) \bigwedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$: Associativity of ^
- $((\alpha \bigvee \beta) \bigvee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$: Associativity of v
- $\neg(\neg\alpha) \equiv \alpha$: double negative association
- $(\alpha \Longrightarrow \beta) \equiv (\neg\beta \Longrightarrow \neg\alpha)$: Contraposition
- $(\alpha \Longrightarrow \beta) \equiv (\neg\alpha \bigvee \beta)$: Implication elimination
- $(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Longrightarrow \beta) \bigwedge (\beta \Longrightarrow \alpha))$: Biconditional elimination
- $\neg(\alpha \bigwedge \beta) \equiv (\neg\alpha \bigvee \neg\beta)$: De Morgan

- ¬(α⋀β) ≡ (¬α ⋁ ¬β): De Morgan
- ¬(α⋁β) ≡ (¬α ⋀ ¬β): De Morgan
- (α⋀(β⋁ɣ)) ≡ ((α∧β)∨(α∧ɣ))): Distributiviy of ^ over v
- (α⋁(β⋀ɣ)) ≡ ((α∨β)∧(α∨ɣ))): Distributivity of ∨ over ∧

Monoticity implies soundness:

- set of entailed sentence can only increase as information is assesd to KB

Your Exam Notes

Here are your full set of exam notes that you can use on your midterm. Choose **Runtime > Run all**, then your notes will be displayed.

If you are taking the exam in person, then you must print a copy of your notes for yourself. Here's how to print:

- Run the cell below.
- Click its "Mirror cell in tab" button (this is located in the upper right corner of the cell, immediately to the left of the trash can icon). This will open the cell.
- In Chrome, you can then print your notes using the **File > Print** option from the browser's menu.
- In the print options, change the layout from Landscape to Portrait.
- Clicking the Save, will save a PDF that you can then print.
- If you're having trouble printing directly from Colab, try copying and pasting the formatted output (below) into a Google doc and printing from there.

```
# This will generate the full set of notes that you can use for your exam.
full_markdown = "\n\n".join([module1_markdown,
                             python_markdown,
                             module2_markdown,
                             module2_markdown,
                             module3_markdown,
                             module4_markdown,
                             module5_markdown,
                             module6_markdown,]
                       )
display(Markdown(full_markdown))
```

Module 1: Rational Agents and Task Environments

AIMA Chapter 1.1 What is AI?

Chapter 1.1 summary.

```
def my_method(arg_1, arg_2):
  for item in arg_1:
    print(item)
```

$\sum_{n=0}^{N} n = b^2$

AIMA Chapter 27.1 The Limits of AI

- weak ai: idea that machines could act as if they're intelligent
- strong ai: idea that machines that do so are consciously thinknig
- good old fasion ai (GOFAI): AI pursued in the cult of computationalism cannot produce results
- qualification problem: difficult to capture every contingency in a set of necessary and sufficiant logic rules
- an agent that only knows a dog(x) = mammal(x) is lacking from an agent that actually observes a dog
- embodided cognition approach: makes no sense to consider the brain seperatly
- Godel's incompleteness theorem:
    1.
    2. applys only math and not computers
- ask not if a machine can think but if it passes a Turing Test

AIMA Chapter 27.1 Can Machines Really Think?

Your summary here

AIMA 2.1 Agents and Environments

basically there are agents an precepts that control AI. The agent you be the thing that directly interacts with the environment \ and it sees percepts. An example would be the vaccum and \ dirt in multiple dirty/clean rooms

AIMA 2.2 Good Behavior: The Concept of Rationality

An agent would need to choose what to do given the performance measure An agent's main goal is to maximize the perfomrance masure with is the expected outcome from an action. An agent would need to analyze its environment in order to \ find the right action to proceed with.

AIMA 2.3 The Nature of Environments

Next step for AI is to figure out the environment that the agent is in. The environment has many different clssifications that are observabale, detemanistic , episodic, sequential, and dynamic.

AIMA 2.4 The Structure of Agents

- AI job is to design an agent program that implements agnet function
-

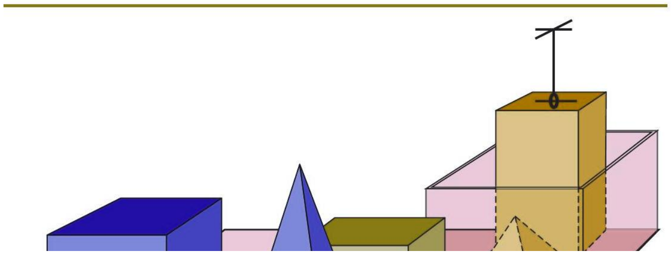
Module 1 Lecture Notes

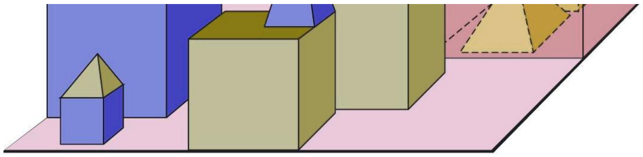
Your summary here

Module 1 Definitions

- **Agent** - anything that can perceive and act in an environment
- **Percept** - the content the agent percieves
- **Agent function** - maps any given percept sequence to action
- **Rational agent** - does action that maximizes the performance measure given the environment
- **Consequentialism** - evaluate based on consequences
- **Task environments** - problems where rational agents are the solution
- **Peas** - performance, environment, snensors, actuators
- **Fully observable** - sensor has all aspects relavent to the choice of action
- **Deterministic** - next state is determined by current state and action
- **Episodic** - agent's experience is divided into atomic episodes, previos action doesnt determine next
- **Sequential** - current decision could affect future ones
- **Dynamic** - environment changes as agent is in action
- **Agent** - architecture + program

Module 1 Figures





- **Figure 1.3** - Blocksworld was an early AI system that allowed users to say commands in natural language in order to have a robotic arm move blocks in a computer simulation. To work successfully the AI must understand human language in a somewhat similar way to Turing's test of computer intelligence.

Module 1: Python

Python Notes

Your summary here

Module 1 Definitions

- **Mutability** - your definition here

Module 2: Search Problems

AIMA Chapter 3.1 Problem-Solving Agents

3.1 Problem Solving Agents Goal Formulation: Define the objectives the agent wants to achieve Problem Formulation: The agent defines an abstract world model; finds the states and actions necessary to reach the goal Search: The agent simulates the outcomes of taking various states and actions in its model to try and find a sequence of actions that reaches the goal Execution: The agent follows the sequence to reach the schools In any fully observable, deterministic, known environment, the solution to any problem is a fixed sequence of actions Open-loop control: The model does not modify its sequence/world model as it perceives statements Closed-loop control: The model takes percepts from the environment and modifies its strategy (branching strategy in nondeterministic/partially observable envs Models are abstract mathematical descriptions of a set of states Valid abstraction – solution in the abstracted space works in the original space

AIMA Chapter 3.2 Example Problems

Grid world: really similar model where agents can move from cell to cell in a grid and perform actions; Vacuum puzzle from prev chapter, sliding tile puzzle, and sokoban all can be modeled in grid world; 8-puzzle Can have infinite state spaces; Donald Knuth's 4 game can result in infinitely large numbers if you continue to apply factorials Touring problems – a set of goal states must be visited; example: the traveling salesman problem (which is NP-hard) VLSI layout, robot navigation, automatic assembly sequencing, protein design

AIMA Chapter 3.3 Search Algorithms

Forms a search tree to represent paths through the state space moving towards the solution The search tree can have multiple paths to any given state (and so multiple nodes for); every node in the tree has a unique path back to the root At each state node (which will become a parent after expansion), we can expand the node by creating a child node for each resulting state that can be created by taking an action; Any state with a node has been reached; the frontier is the set of nodes that have been created but not explored (they exist in the queue/stack)e

AIMA Chapter 3.4 Uninformed Search Strategies

Breadth First Search Complete on infinite search spaces For each expansion, append unvisited children to the end of a FIFO queue (check if they are goals before pushing, early goal test) Every intermediate state has an optimal path to it at any given point; this is reliant on the property that each action has equal cost Time/Space complexity of O(b^d) – very memory hungry This is because it has to store all previous states Dijkstra's/Uniform Cost Search Best First Search, where path cost is the evaluation function Complete and cost optimal Time complexity of O(b^(1 + floor(C/e))) C is the optimal cost e is a lower bound on the cost of any action This means that UCS is generally much slower than BFS Depth First Search Tree-like search (usually, can also be implemented as BestFS(-depth)) This means it does not keep a table of reach states Non-optimal cost – just returns the first path Keeps proceeding down to the deepest level it can reach via one path, then pops back up to the next deepest node that is not fully expanded Complete and efficient in finite tree-like state spaces Can get stuck in infinite loops, and can go down infinite paths in infinite state spaces; Incomplete in infinite state spaces Time complexity is O(n) and Space is O(bm) Very memory efficient (especially compared to BFS) Backtracking Search: even more efficient–only store the current path and move back along your maintained stack and update new explorations in-place; O(m) vs O(bm)

Module 2 Lecture Notes

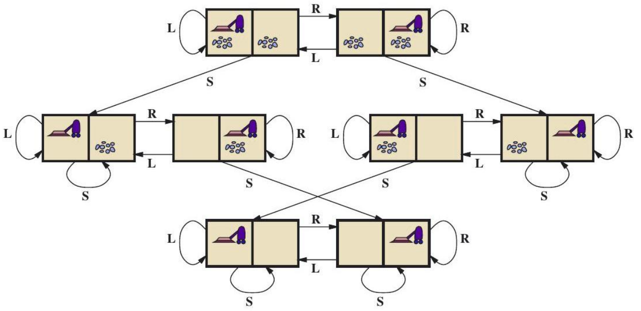
- if fully observable, determenistic, known environment the solution is a fixed sequence of actions - if partially observable, nondeteremnistic solution is based on what percept recieves - Depth Limited DFS: Fixed Depth DFS but with a parameter l, such that every node at l is treated as if it has no successors TC: O(b^l) SC: O(bl); can be incomplete if l is too large/small Can use constant time cycle checking + depth constraint to eliminate most cycles Diameter of the state space (max distance between any two actions) can be a good choice for l Iterative Deepening Try the depth-limited search for every value of l up to a limit Memory: O(bd) with solution, O(bm) without Time: O(b^d) with solution, O(b^m) without Complete on any finite space with cycle detection Preferred with limited memory and unknown search depth

Bidirectional Search: Complete and Optimal Search from both the initial and goal state and meet in the middle Solution found when both frontiers meet Always expand from the node with minimum eval function value (from both sides combined) TC: O(b^(d/2)) SC: O(b^(d/2));

Module 2 Definitions

- **Problem-solving agent** - an agent that can formulate a plan via search
- **Reflex-agent** - chooses action based only on cuurent percept

Module 2 Figures



- **Figure 3.2** - Your explanation here

Module 2: Search Problems

AIMA Chapter 3.1 Problem-Solving Agents

3.1 Problem Solving Agents Goal Formulation: Define the objectives the agent wants to achieve Problem Formulation: The agent defines an abstract world model; finds the states and actions necessary to reach the goal Search: The agent simulates the outcomes of taking various states and actions in its model to try and find a sequence of actions that reaches the goal Execution: The agent follows the sequence to reach the schools In any fully observable, deterministic, known environment, the solution to any problem is a fixed sequence of actions Open-loop control: The model does not modify its sequence/world model as it perceives statements Closed-loop control: The model takes percepts from the environment and modifies its strategy (branching strategy in nondeterministic/partially observable envs Models are abstract mathematical descriptions of a set of states Valid abstraction – solution in the abstracted space works in the original space

AIMA Chapter 3.2 Example Problems

Grid world: really similar model where agents can move from cell to cell in a grid and perform actions; Vacuum puzzle from prev chapter, sliding tile puzzle, and sokoban all can be modeled in grid world; 8-puzzle Can have infinite state spaces; Donald Knuth's 4 game can result in infinitely large numbers if you continue to apply factorials Touring problems – a set of goal states must be visited; example: the traveling salesman problem (which is NP-hard) VLSI layout, robot navigation, automatic assembly sequencing, protein design

AIMA Chapter 3.3 Search Algorithms

Forms a search tree to represent paths through the state space moving towards the solution The search tree can have multiple paths to any given state (and so multiple nodes for); every node in the tree has a unique path back to the root At each state node (which will become a parent after expansion), we can expand the node by creating a child node for each resulting state that can be created by taking an action; Any state with a node has been reached; the frontier is the set of nodes that have been created but not explored (they exist in the queue/stack)e

AIMA Chapter 3.4 Uninformed Search Strategies

Breadth First Search Complete on infinite search spaces For each expansion, append unvisited children to the end of a FIFO queue (check if they are goals before pushing, early goal test) Every intermediate state has an optimal path to it at any given point; this is reliant on the property that each action has equal cost Time/Space complexity of O(b^d) – very memory hungry This is because it has to store all previous states Dijkstra's/Uniform Cost Search Best First Search, where path cost is the evaluation function Complete and cost optimal Time complexity of O(b^(1 + floor(C/e))) C is the optimal cost e is a lower bound on the cost of any action This means that UCS is generally much slower than BFS Depth First Search Tree-like search (usually, can also be implemented as BestFS(-depth)) This means it does not keep a table of reach states Non-optimal cost – just returns the first path Keeps proceeding down to the deepest level it can reach via one path, then pops back up to the next deepest node that is not fully expanded Complete and efficient in finite tree-like state spaces Can get stuck in infinite loops, and can go down infinite paths in infinite state spaces; Incomplete in infinite state spaces Time complexity is O(n) and Space is O(bm) Very memory efficient (especially compared to BFS) Backtracking Search: even more efficient–only store the current path and move back along your maintained stack and update new explorations in-place; O(m) vs O(bm)

Module 2 Lecture Notes

- if fully observable, determenistic, known environment the solution is a fixed sequence of actions - if partially observable, nondeteremnistic solution is based on what percept recieves - Depth Limited DFS: Fixed Depth DFS but with a parameter l, such that every node at l is treated as if it has no successors TC: O(b^l) SC: O(bl); can be incomplete if l is too large/small Can use constant time cycle checking + depth constraint to eliminate most cycles Diameter of the state space (max distance between any two actions) can be a good choice for l Iterative Deepening Try the depth-limited search for every value of l up to a limit Memory: O(bd) with solution, O(bm) without Time: O(b^d) with solution, O(b^m) without Complete on any finite space with cycle detection Preferred with limited memory and unknown search depth

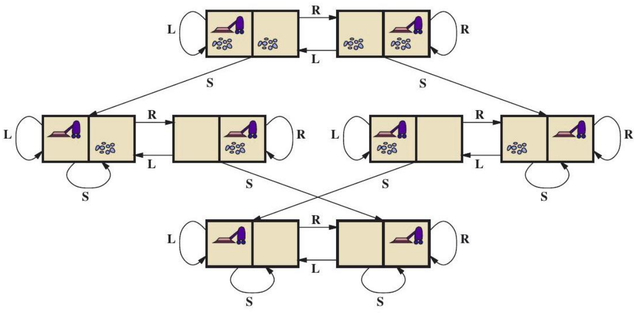
- memory and unknown search depth

Bidirectional Search: Complete and Optimal Search from both the initial and goal state and meet in the middle Solution found when both frontiers meet Always expand from the node with minimum eval function value (from both sides combined) TC: O(b^(d/2)) SC: O(b^(d/2));

Module 2 Definitions

- **Problem-solving agent** - an agent that can formulate a plan via search
- **Reflex-agent** - chooses action based only on cuurent percept

Module 2 Figures



- **Figure 3.2** - Your explanation here

Module 3: Informed Search

AIMA Chapter 3.5 Informed (Heuristic) Search Strategies

A* Search Best first search with eval fn f(n) = g(n) + h(n) Cost up til now + estimated remaining cost Complete in finite/infinite spaces Optimality is conditional on heuristic properties admissibility – never overestimates the cost to the goal Consistency – stronger version of admissibility; using the triangle inequality h(n) <= c(n, a, n') + h(n') Surely expanded nodes – If a reachable node is on a path such that every node on the path has f(n) < C*, it will be expanded by A*; some nodes on the goal contour (f(n) = C*) may be expanded, but none past this are Contours – like a topographic map; can circle the changing frontier based on the value of f(n) A* is optimally efficient because any other optimal algorithm would have to expand all nodes surely expanded by A* A* is efficient because it prunes inefficient search paths from the tree Number of nodes expanded can be exponential in the length of the solution

AIMA Chapter 3.6 Heuristic Functions

Heuristic Quality Effective Branching factor b* – Given A* generates N nodes, and the solution depth is d, b* is the branching factor for a uniform tree of N+1 nodes with depth d Well designed heuristics have b* ~= 1 Effective Depth – find out the constant depth difference between an uninformed search and the informed search for some heuristic If a heuristic is almost always better than another (by some index) we say that it dominates the other heuristic Relaxed Problems Common to get heuristic from a version of the problem with less rules Because these only add edges, any real solution will be a solution in the relaxed space ABSOLVER can generate these automatically for some problems Pattern Databases Store the cost of specific patterns in the problem to get the cost estimate of a subproblem efficiently Ideally the subproblems are much cheaper to solve, and the patterns will be repeated for more efficiency add Combine the multiple pattern heuristics into a single one (addition, max, min) Landmark Storage Precomputing the optimal cost between states in the graph can speed up the inference a lot Take 10-20 "Landmark points" and compute distance from them to every other node Heuristic will be inadmissible if an optimal path does not go through a landmark Shortcuts – precomputed shorter paths in the graph Differential heuristic h_DH(n) = max (for all L) IC*(n,L) - *C(goal, L)I Needs landmarks placed fairly well distributed Meta-learning - works at the meta-level state space; can prevent algorithm from repeating undesirable algorithms Chapter 5 - Adversarial Search

Module 3 Lecture Notes

UCS:

- assume step-cost = 1
- g(N) : the path cost function
  - is the depth in search tree
- extension of BFS
- frontier = priroty que using g(n)
- prioritize which nodes to go through

Best-First-Search:

- select node fro expansion w/ minimal evaluation function f(n)
- f(n) is function that includes estimate heuristic

Greedy best-first search:

- expands the node stimated to be closest to goal
- ignores g(n): cost to get to n
- example is Manhattan Dis, Euclidian
- can get stuck in loops, incomplete

A*:

- expand paths that are most promising first
- f(n) = g(n) + h(n)
- frontier is priority queue by increasing f(n)

- frontier is priority queue by increasing f(n)

Heuristic:

- is admissible if never overestimates the cost: optimistic

Dominance:

- if h2(n) >= h1(n): h2 dominates h1
- admissible(optimistic) if 0 <= h(n) <= h*(n)

Module 3 Definitions

- **State space** - set of all states reachable from the initial state by sequence of actions
- **Path** - sequence of actions leading state s_j to s_k
- **Frontier** - states that are available for expanding
- **Solution** - path leading from start to goal
- **Strategy** - order of tree expansion
- **Completeness** - always finds solution
- **Optimality** - Always fnds best solution
- **Time complexity** - # of nodes generated
- **Space complexity** - # of nodes in memory at same time
- **B** - max branching factor
- **D** - depth, shallowest goal node
- **M** - max length of any path
- **Heuristic** - takes state n and returns estimate of dis form n to goal
- **Dominance** - metric on better heuristics

Module 4: Adversarial Search and Games

AIMA Chapter 5.1 Game Theory

Stances on other agents: Economy – treat the actions of other agents as an aggregate effect rather than an individual one; works best in scenarios with many agents Part of the environment – misses the nuance of competing goals Adversarial Game Tree – explicitly model the actions of an adverse agent Two Player Zero-Sum Games Define the game as such Initial State S_0 To-Move(s): the player whose turn it is to move Actions(s): the set of legal actions at state s Result(s, a) -> Transition Model – what state do we get taking action a at state s? Is-Terminal(s) – a test that determines whether the game has ended Utility(s, p) – a function that determines the payoff to each player based on the terminal state

AIMA Chapter 5.2 Optimal Decisions in Games

MAX wants to win and MIN wants to stop them, so MAX must be able to conditionally branch on MIN's decisions For binary outcomes, use AND-OR, general solution is minimax Ply - one player's action Minimax(s) – the utility of that state given both players play optimally from that state Minimax is not always best if you know that you're better than your opponent Minimax Search Recurse through the tree, and float the minimax values at each level up; then return the action from your action state that moves to the maximum (or minimum) value O(b^m) TC; SC can be O(bm) or O(m) Not practical in large search spaces Multiplayer Games Store utilities in a vector with length equal to # of players Utility of node n is equal to the successor state vector which has the highest backed-up value for the player choosing at n Alliances can implicitly form as a result of the optimal actions of multiple players aligning Alliances can be formed explicitly too and have certain rules; breaking alliances can be viewed negatively In positive-sum games, there are many situations where collaboration is optimal Alpha-Beta Pruning Create a bound on the possible values of each node, based on what we know about the previous searches; only consider paths that have the possibility of changing the root decision (for example, if less than root value ignore) Alpha – at least – highest for MAX, Beta – at most – lowest for MIN; these will eventually converge to be equal as more nodes are discovered A better ordering of actions can result in more effective pruning; up to O(b ^ (m/2)) is possible with perfect ordering Can inform move ordering based on what previous moves resulted in possibly better past situations; search for killer moves Transposition Tables – store the heuristic/utility values of repeated states, to stop repeated searches Type A - wide but shallow, then estimate utility (Chess) Type B - follow a few promising paths far (Go, recently Chess)

AIMA Chapter 5.3 Heuristic Alpha–Beta Tree Search

Replace terminal test with a cutoff test, using a heuristic function to approximate the ending utility from the cutoff state Eval should be equal to Utility at terminal states, and between a loss and win at cutoffs; Should be fast to compute but also be strongly correlated with winning Ex: Expected value based on proportion of outcomes possible from state Can use a linear combination with independent features, nonlinear otherwise For AlphaBeta, iterate depth of search, and have cutoff happen at depth limit; however, we should consider quiescence– only cutoff if there are no pending moves (such as captures) otherwise keep looking Horizon effect – when the agent delays inevitable negative effects past the depth limit, erroneously thinking it has prevented the effects Singular Extensions – keep exploring moves that are clearly better at a certain node, past the cutoff point
Forward Pruning Can sometimes accidentally prune away the best option Beam Search – take the n best moves (based on heuristic) rather than all possible moves ProbCut – nondeterministic alpha-beta; prunes if it its likely outside the alpha-beta window Late move reduction – assume move ordering is good;reduce depth of search for later moves

AIMA Chapter 5.5 Stochastic Games

Your summary here

AIMA Chapter 16.1 Combining Beliefs and Desires under Uncertainty

16.1 Combining Beliefs and Desires under Uncertainty P(Result(a) = s) = Sum_(s) P(s)P(s'|s, a) EU(a) = Sum_(s) P(Result(a) = s`)U(s') Where EU is expected Utility and U is utility The agent will work to maximize the utility function, so if the utility function matches the goal, it will be very good Utility function for next action, performance measure over past actions

AIMA Chapter 16.2 The Basis of Utility Theory

Your summary here

AIMA Chapter 16.3 Utility Functions

Your summary here

Module 4 Lecture Notes

- MAX and MIN, MAX moves first
- take turns till game over
- winner gets award

Minimax:

- make best move for MAX
- backed-up val of each node in tree os deteremined by value of children
- max: backed up val is max of the values
- min: backed up value is min of values
  1. Start with current poss as MAX
  2. Expand game tree a fixed number of ply
  3. Apply eval func to leaf pos
  4. calculate back up vals bottom up
  5. Pick move assigned to MAX at the root
  6. Wait for MIN to respond def max-value(state): if the state is a terminal state: return the state’s utility initialize v = -∞ for each successor of state: v = max(v, min-value(successor)) return v def min-value(state): if the state is a terminal state: return the state’s utility initialize v = +∞ for each successor of state: v = min(v, max-value(successor)) return v

Alpha-Beta Pruning:

- cut of stems based on if val will be reached
- replace alpha(max): max's best option
- replace beta(min): min's best option

Expectimax Search: compute avg score under optimal play

- have max nodes and chance nodes
  - chance nodes are min nodes but outcome is uncertain
  - calc their expected utils

Module 4 Definitions

- **Concept** - definition

Module 5: Constraint Satisfaction Problems

AIMA Chapter 6.1 Defining Constraint Satisfaction Problems

Your summary here

AIMA Chapter 6.2 Constraint Propagation: Inference in CSPs

Your summary here

AIMA Chapter 6.3 Backtracking Search for CSPs

Your summary here

AIMA Chapter 6.4 Local Search for CSPs

Your summary here

AIMA Chapter 6.5 The Structure of Problems

Your summary here

Module 5 Lecture Notes

What is Search For: Assumptions about the worlds: Single Agent Deterministic actions Observed state Discrete state space Planning: sequence of actions Path to the goal is important Paths have various costs & depths Heuristics give problem specific guidance (help speed up search Identification: Goal is important, not the path All paths have same depth Constraint satisfaction problems are specialized for identification

Big Idea: Represent the constraints that solutions must satisfy in uniform declarative language Find solutions by general purpose search algos with no change from problem to problem No hand-built transition functions No hand built heuristics Just specify the problem in a formal declarative language and the general-purpose algo does the rest

CSP Definition: Consists of Finite set of variables X1, X2, … Xn non empty domain of all possible values for each variable D1, … Dn where Di = {v1, …., vk Finite set of constraints Each constraint has a limit ofr the values variables can take A state is the assignment of values to some variables A consistent assignment doesn’t violate the constraints like in Sudoku Assignment is complement when all variables have a value Solution is complete and consistent Solutions can be found w/ complete general purpose algorithms

Benefits of CSP Clean specification of many problems, generic goal, successor function & heuristics CSP “knows” which variables violate a constraint Knows where to focus search Automatically prune off all branches that violate the constraints

Representations: Constraint graph Nodes are variables Arcs are binary constraints Standard representation pratten: Variables with values Constraint graph simplifies search This problem is binary CSP

Varieties: Discrete variables Finite domains N variables, domain size d -> O(d^n) complete assignments Not all gonna be solutions Infinite domains Integers, strings E.g Job scheduling

Idea 1: CSP as a search problem: Can be expressed as a search problem, Initial State: the empty assignment {} Successor function: Assign value to any unassigned variable provided that there is not constraint conflict Goal test: current assignment is complete Path cost: constant cost for each step Solution is always found at depth n for n variable DFS can be used Search and branching factor N variables of domain size d Branching factor at root is n*d Branching factor at next level is (n - 1) * d Tree has n! * d^n Variable assignments are commutative Only need to consider assignments to a single variable at each node

Searching and Backtracking: DFS for CSPs with variable assignments is called backtracking search Go down a path and if you reach a wong end, go back to previous move and continue from there Backtracking search is basic uninformed algo for CSP

Improving backtracking efficiency: General purpose methods & general-purpose heuristics can give huge gains in speed Heuristics: Q: which value should be assigned next: 1 most constrained variable If ties: most constraining variable Which order should variable values be tried: Least constraining value Can we detect inevitable failure early: Forward checking

Towards Constraint Propagation: Forward checking propagates info from assigned to unassigned variables but does provide detection for all failures Constraint propagation goes beyond forward checking & repeatedly enforces constraints locally

Idea 3(big idea): inference in CSP: CSP combines search and inference Search Constraint propagation (inference) Eliminates possible values for a variable if the value would violate local consistency Can do inference first, intertwine with search Local Consistency: Node consistency: satisfies unary constraints Is trivial Arc consistency: satisfies binary constraints Xi is arc-consistent with respect to xj if for every value v in Dj there is some value w in Dj that satisfies the binary constraint on the arc between xi and xj Convert an edge into an arc from i -> j and j -> i X -> Y is consistent if for every value x of x there is some allowed y

Module 5 Definitions

- **Concept** - definition

Module 6: Logical Agents

AIMA Chapter 7.1 Knowledge-Based Agents

- knowledge based agents use reasoning over internal representation to choose next action
- can accept new tasks in the form of described goals
- achieve competence by being told about the enviornment and adapt to changes
- knowledge base: a sentence
- knowledge language representation: each sentence is representing an aspect of the world
- Axiom: sentence that is being taken as given w/out being derived
- adding to knowedge base is called Tell and Ask
- inference: deriving new sentences from old
- usually operates in 3 steps:
    1. tells knowledge base what it percieves
    2. Asks what action it should perform
    3. Tells the knowledge base what action it performed
- declarative: start with empty KB and Tell it sentences until it percieves its enviornment
- procedural: encodes desired behaviors directly

AIMA Chapter 7.2 The Wumpus World

Your summary here

AIMA Chapter 7.3 Logic