Propositional Logic & Reasoning

Vikram Pudi
IIIT Hyderabad

Knowledge Representation

- Knowledge Representation: expressing knowledge explicitly in a computer-tractable way
  - Knowledge Base: set of facts (or sentences) about the domain in which the *agent* finds itself
  - These sentences are expressed in a (formal) language such as logic

Why is it important?

- Reasoning: draw inferences from knowledge
  - answer queries
  - discover facts that follow from the knowledge base
  - decide what to do
  - etc.

Logic in General

- Logics are formal languages for representing information such that conclusions can be drawn
- Syntax: Describes how to make sentences
- Semantics: How sentences relate to reality. The meaning of a sentence is not *intrinsic* to that sentence.
- Proof Theory: A set of rules for drawing conclusions (inferences, deductions).

Logical Arguments

- All humans have 2 eyes.
- Kishore is a human.
  - Therefore Kishore has 2 eyes.
- All humans have 4 eyes.
- Kishore is a human.
  - Therefore Kishore has 4 eyes.
- Both are (logically) valid arguments.
- Which statements are true / false ?

Logical Arguments (contd)

- All humans have 2 eyes.
- Kishore has 2 eyes.
  - Therefore Kishore is a human.
- No human has 4 eyes.
- Kishore has 2 eyes.
  - Therefore Kishore is not human.
- Both are (logically) invalid arguments.
- Which statements are true / false ?

From English to Propositional Formulae

- "it is not the case that the lectures are dull": $\neg D$ (alternatively "the lectures are not dull")
- "the lectures are dull and the text is readable": $D \wedge R$
- "either the lectures are dull or the text is readable": $D \vee R$
- "if the lectures are dull, then the text is not readable": $D \rightarrow R$
- "the lectures are dull if and only if (iif) the text is readable": $D \leftrightarrow R$
- "if the lectures are dull, then if the text is not readable, Kishore will not pass": $D \rightarrow (\neg R \rightarrow \neg P)$

Why *formal* languages?

- Natural languages exhibit ambiguity.
  - Examples:
    - The boy saw a girl with a telescope
    - Our shoes are guaranteed to give you a fit
  - Ambiguity makes reasoning difficult / incomplete
- Formal languages promote rigour and thereby reduce possibility of human error.
- Formal languages help reduce implicit / unstated assumptions by removing *familiarity* with subject matter
- Formal languages help achieve generality due to possibility of finding *alternative interpretations* for sentences and arguments.

Propositional Logic

- Use letters to stand for "basic" propositions
- Complex sentences use operators for not, and, or, implies, iff.
- Brackets ( ) for grouping
  $(P \rightarrow (Q \rightarrow (\neg(R)))$ vs. $P \rightarrow (Q \rightarrow \neg R)$
- Omitting brackets
  - precedence from highest to lowest is: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$
  - Binary operators are left associative
    (so $P \rightarrow Q \rightarrow R$ is $(P \rightarrow Q) \rightarrow R$)
- Questions:
  - Is $(P \vee Q) \vee R$ same as $P \vee (Q \vee R)$?
  - Is $(P \rightarrow Q) \rightarrow R$ same as $P \rightarrow (Q \rightarrow R)$?

Semantics (Truth Tables)

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \rightarrow Q$	$P \leftrightarrow Q$
True	True	False	True	True	True	True
True	False	False	False	True	False	False
False	True	True	False	True	True	False
False	False	True	False	False	True	True

- One row for each possible assignment of True/False to propositional variables
- **Important:** Above P and Q can be any sentence, including complex sentences

Terminology

- A sentence is valid if it is True under all possible assignments of True/False to its propositional variables (e.g. $P \vee \neg P$).
- Valid sentences are also referred to as tautologies
- A sentence is satisfiable if and only if there is *some* assignment of True/False to its propositional variables for which the sentence is True
- A sentence is unsatisfiable if and only if it is not satisfiable (e.g. $P \wedge \neg P$).

Semantics (Complex Sentences)

R	S	$(R \wedge S) \rightarrow (\neg R \vee S)$
True	True	
True	False	
False	True	
False	False	

Semantics (Complex Sentences)

R	S	$\neg R$	$R \wedge S$	$\neg R \vee S$	$(R \wedge S) \rightarrow (\neg R \vee S)$
True	True	False	True	True	True
True	False	False	False	False	True
False	True	True	False	True	True
False	False	True	False	True	True

Material Implication

- The only time $P \rightarrow Q$ evaluates to False is when P is True and Q is False
- This is known as a conditional statement or material implication
- English usage often suggests a causal connection between antecedent (P) and consequent (Q) – this is not reflected in the truth table
- So $(P \wedge \neg P) \rightarrow$ *anything* is a tautology!

Exercises

Given: A and B are true; X and Y are false, determine truth values of:

- $\neg(A \vee X)$
- $A \vee (X \wedge Y)$
- $A \wedge (X \vee (B \wedge Y))$
- $[(A \wedge X) \vee \neg B] \wedge \neg[(A \wedge X) \vee \neg B]$
- $(P \wedge Q) \wedge (\neg A \vee X)$
- $[(X \wedge Y) \rightarrow A] \rightarrow [X \rightarrow (Y \rightarrow A)]$

Entailment

- $S \Rightarrow P$ — whenever all the formulae in the set S are True, P is True
- This is a *semantic* notion; it concerns the notion of *Truth*
- To determine if $S \Rightarrow P$ construct a truth table for S, P
  - $S \Rightarrow P$ if, in any row of the truth table where all formulae of S are true, P is also true
- A tautology is just the special case when S is the empty set.

Entailment Example

P	$P \rightarrow Q$	Q
True	True	True
True	False	False
False	True	True
False	True	False

Modus Ponens

Therefore, $P, P \rightarrow Q \Rightarrow Q$

Exercises

Use truth tables to determine validity of:

- If it rains, Raju carries an umbrella. Raju is carrying an umbrella, therefore it will rain.
- If the weather is warm and the sky is clear, then either we go swimming or we go boating. It is not the case that if we do not go swimming, then the sky is not clear. Therefore, either the weather is warm or we go boating.

Formal Proofs

- Intend to formally capture the notion of proof that is commonly applied in other fields (e.g. mathematics).
- A proof of a formula from a set of premises is a sequence of steps in which any step of the proof is:
  1. An axiom or premise
  2. A formula deduced from previous steps of the proof using some rule of inference
- The last step of the proof should deduce the formula we wish to prove.
- We say that *S* follows from (premises) *P* to denote that the set of formulae *P* "prove" the formula *S*.

Soundness and Completeness

- A logic is sound if it preserves truth (i.e. if a set of premises are all true, any conclusion drawn from those premises *must* also be true).
- A logic is complete if it is capable of proving *any valid* consequence.
- A logic is decidable if there is a mechanical procedure (computer program) to prove *any* given consequence.

Inference Rules

- Modus Ponens: $P, P \rightarrow Q \Rightarrow Q$
- Modus Tollens: $P \rightarrow Q, \neg Q \Rightarrow \neg P$
- Hypothetical Syllogism: $P \rightarrow Q, Q \rightarrow R \Rightarrow P \rightarrow R$
- And-Elimination: $P_1 \wedge P_2 \wedge \ldots \wedge P_n \Rightarrow P_i$
- And-Introduction: $P_1, P_2, \ldots, P_n \Rightarrow P_1 \wedge P_2 \wedge \ldots \wedge P_n$
- Or-Introduction: $P_i \Rightarrow P_1 \vee P_2 \vee \ldots \vee P_n$
- Double-Negation Elimination: $\neg\neg P \Rightarrow P$
- Unit Resolution: $P \vee Q, \neg Q \Rightarrow P$
- Resolution: $P \vee Q, \neg Q \vee R \Rightarrow P \vee R$

Example Formal Proof

1. $A \vee (B \rightarrow D)$
2. $\neg C \rightarrow (D \rightarrow E)$
3. $A \rightarrow C$
4. $\neg C \quad / \therefore B \rightarrow E$
5. $\neg A \quad$ *3,4 (Modus Tollens)*
6. $B \rightarrow D \quad$ *1,5 (Unit Resolution)*
7. $D \rightarrow E \quad$ *2,4 (Modus Ponens)*
8. $B \rightarrow E \quad$ *6,7 (Hypothetical Syllogism)*

Exercises

Construct formal proof of validity for:

- If the investigation continues, then new evidence is brought to light. If new evidence is brought to light, then several leading citizens are implicated. If several leading citizens are implicated, then the newspapers stop publicizing the case. If continuation of the investigation implies that the newspapers stop publicizing the case, then the bringing to light of new evidence implies that the investigation continues. The investigation does not continue. Therefore, new evidence is not brought to light.
- *C*: The investigation continues. *N*: New evidence is brought to light. *I*: Several leading citizens are implicated. *S*: The newspapers stop publicizing the case.

Machine, Data and Learning

Machine Learning

- Scientific study of algorithms and statistical models that computer systems use
  - To perform a specific task effectively without using explicit instructions
  - Rely on patterns and inference instead.
- Involves
  - Building a **mathematical model** based on sample data, known as "training data“ to make predictions or decisions
  - No explicit programming done to perform the task

Machine Learning

- Term coined around 1960
- Why learn ? Why not just hire enough programmers and code in rules ?
  - Lots of patterns for an activity/event
  - Events can be dynamic
  - **Data** is increasing exponentially
  - **Data** is also in various formats [Text, Audio, Video]
  - Higher quality **data** due to cheaper storage
- Can be broadly classified into three categories
  - Unsupervised, Supervised and Reinforcement learning

Unsupervised Learning

- Takes a set of data that contains only inputs and finds structure in data E.g., Grouping or Clustering of data points

- **Marketing:** Finding groups of customers with similar behavior given a large database of customer data containing their properties and past buying records.
- **Biology:** Classification of plants and animals given their features.
- **Earthquake studies:** Clustering observed earthquake epicenters to identify dangerous zones.
- **World Wide Web:** Clustering weblog data to discover groups of similar access patterns.

Supervised Learning

- Builds mathematical model using data set that has both inputs and desired outputs E.g., Classification and Regression tasks

User ID	Gender	Age	Salary	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	1
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	1
15728773	Male	27	58000	1
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	1
15727311	Female	35	65000	0
15570769	Female	26	80000	1
15606274	Female	26	52000	0
15746139	Male	20	86000	1
15704987	Male	32	18000	0
15628972	Male	18	82000	0
15697686	Male	29	80000	0
15733883	Male	47	25000	1

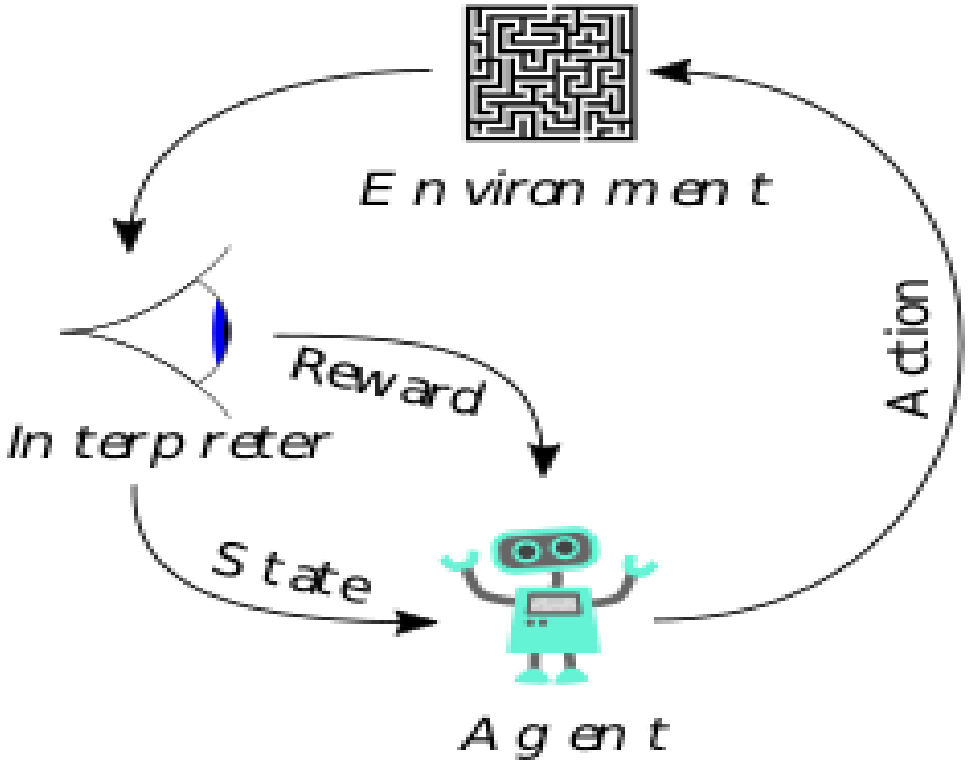
Figure A: CLASSIFICATION

Temperature	Pressure	Relative Humidity	Wind Direction	Wind Speed
10.69261758	986.882019	54.19337313	195.7150879	3.278597116
13.59184184	987.8729248	48.0648859	189.2951202	2.909167767
17.70494885	988.1119385	39.11965597	192.9273834	2.973036289
20.95430404	987.8500366	30.66273218	202.0752869	2.965289593
22.9278274	987.2833862	26.06723423	210.6589203	2.798230886
24.04233986	986.2907104	23.46918024	221.1188507	2.627005816
24.41475295	985.2338867	22.25082295	233.7911987	2.448749781
23.93361956	984.8914795	22.35178837	244.3504333	2.454271793
22.68800023	984.8461304	23.7538641	253.0864716	2.418341875
20.56425726	984.8380737	27.07867944	264.5071106	2.318677425
17.76400389	985.4262085	33.54900114	280.7827454	2.343950987
11.25680746	988.9386597	53.74139903	68.15406036	1.650191426
14.37810685	989.6819458	40.70884681	72.62069702	1.553469896
18.45114201	990.2960205	30.85038484	71.70604706	1.005017161
22.54895853	989.9562988	22.81738811	44.66042709	0.264133632
24.23155922	988.796875	19.74790765	318.3214111	0.329656571

Figure B: REGRESSION

Reinforcement Learning

- Concerned with how software agents should take actions in an environment to maximize cumulative reward E.g. Autonomous vehicles, Computer games



Some Applications

- Search engines
- Information retrieval
- Recommendation systems
- Credit card fraud detection
- Disease diagnosis
- Election prediction
- Image processing
- Speech translation
- …

AlphaGo

- First computer Go program to defeat a 9-dan professional player
- Uses Monte Carlo Tree search algorithm based on knowledge learned by a deep learning method
- Beat World No. 1 ranked player in 2017
  - Retired after this match
- https://deepmind.google/technologies/alphago/
- https://www.youtube.com/watch?v=WXuK6gekU1Y

- AlphaGo Zero – Version without human data and stronger than AlphaGo [defeated 100-0]

AlphaZero & MuZero

- AlphaZero, a generalized version of AlphaGo Zero Took 4 hours to learn Chess and defeat reigning world computer chess champion 28 to 0 in 100 matches
- https://www.youtube.com/watch?time_continue=7&v=tXlM99xPQC8

- MuZero: Master games without knowing rules
- Uses approach similar to AlphaZero, developed in 2019
- Trained via self-play and play against AlphaZero with no access to rules, opening books or endgame tables
- Viewed as significant advancement over AlphaZero

AlphaFold: solution to a 50 year old grand challenge in biology

- https://www.youtube.com/watch?v=KpedmJdrTpY
- https://deepmind.google/discover/blog/alphafold-a-solution-to-a-50-year-old-grand-challenge-in-biology/
- Figuring out what shapes proteins fold into is known as the “protein folding problem” - grand challenge in biology for the past 50 years
- Focus of intensive scientific research for many years, using a variety of experimental techniques such as nuclear magnetic resonance and X-ray crystallography.

AlphaFold

- Number of ways a protein could theoretically fold before settling into its final 3D structure is astronomical.
- Cyrus Levinthal estimated 10^300 possible conformations for a typical protein.
- Estimated would take longer than the age of universe to enumerate all possible configurations. Yet in nature, proteins fold spontaneously, some within milliseconds - referred to as Levinthal's paradox.

Year 2023 in review for AI

- Article titled 2023: The Crazy AI Year by Nisha Arya, KDnuggets
- Many media sources claim year 2023 can be considered the year of AI
- Jan:
  - With huge buzz around ChatGPT Microsoft announced $10 billion funding in OpenAI
- Feb:
  - Google came up with BARD. Microsoft came up with its Bing chatbot

Year 2023 in review for AI

- Mar:
  - Access to Bard was given to a limited number of people to kickstart the Google GenAI journey.
  - Initiated a domino effect with Adobe introducing Firefly and Canva introducing their virtual design assistant.
  - OpenAI also launched APIs for ChatGPT, as well as their text-to-speech model called Whisper. On the 14th of March, OpenAI released its most advanced model GPT-4.

Year 2023 in review for AI

- Apr:
  - Announcement of Google DeepMind - a combination of Google Research and DeepMind.
  - Russia's Sberbank released ChatGPT rival GigaChat
  - HuggingFace also entering the market with the release of an AI chatbot to rival ChatGPT called HuggingChat
- May:
  - Google announced the Bard chatbot to the public - added some fuel to the GenAI fire with Microsoft revealing its debut AI assistant for Windows 11.

Year 2023 in review for AI

  - Market capitalization of NVIDIA topped $1 trillion for the first time, holding its status as the AI chip leader.
  - Elon Musk's new brain implant startup, called Neuralink, in which the company aims to create and implant AI-powered chips in people’s brains. This was approved by the FDA for human trials.
- Jun:
  - Apple’s Vision Pro, the AI-powered augmented reality headset was developed to take immersive experiences to the next level.
  - European Parliament made some negotiations about the EU AI Act, with 499 votes in favor, 28 against, and 93 abstentions.

Year 2023 in review for AI

  - McKinsey predicted that GenAI has the potential to add up to $4.4 trillion in value to the global economy.
- July:
  - Meta introduced Llama 2, an open-source Large Language Model (LLM) which was trained on a mix of publicly available data, and designed to drive applications such as OpenAI's ChatGPT, Bing Chat, and other modern chatbots.
  - Anthropic also released Claude 2, which dethroned ChatGPT and has it shaking in its boots.
  - The safety around AI is becoming a popular topic as LLMs are becoming a part of our day-to-day lives.

Year 2023 in review for AI

- Microsoft announced that it will charge customers $30 per month to use Microsoft 365 Copilot.

- Aug:
  - Google said that it would also be charging $30 per month for users to make use of their GenAI tools in their Duet AI for Workspace.
  - OpenAI introduced custom instructions to get the most out of ChatGPT. Poe a chatbot service that allows you to use state-of-the-art models such as Claude +, GPT-3.5-Turbo, and GPT-4.

Year 2023 in review for AI

- Sept:
  - Amazon announced a $4 billion investment in OpenAI competitor Anthropic.
  - OpenAI continues with its quest to visualize content with a Canva plugin for ChatGPT.
- Oct:
  - We experienced the Executive Order on the Safe, Secure, and Trustworthy Development and Use of Artificial Intelligence. This also was shaking up the AI world, with CEOs, leaders, and others having contradicting opinions about the implementation of AI systems into society.

Year 2023 in review for AI

- Nov:
  - Elon Musk's AI startup, xAI, unveiled the AI chatbot "Grok", AWS with the release of Amazon Q, and Pika 1.0 from StabilityAI.
  - OpenAI also held its first developer event in November, where it delved into GPT-4 Turbo and the GPT Store.
  - OpenAIs CEO Sam Altman getting fired by the board out of nowhere. He was immediately offered a job by Microsoft with OpenAI employees threatening to resign if Sam Altman did not come back and claim his position as CEO. So now he is back, with some new board members as well as a new "observer" role for Microsoft.

Year 2023 in review for AI

- Dec:
  - Google came to shake the market again with their 3 variant family of large language models and ChatGPT's new rival: Gemini.
  - We already know that OpenAI is looking into GPT 5, 6, and 7. So let’s see what 2024 January has to bring.

Machine, Data and Learning

Selected slides for lectures on ML Topic

Generalization & Goodness of Fit

- Based on Chapter 1 of Python Machine Learning by Example by Yuxi Liu

- **Generalization** refers to how well the concepts learned by a ML model generalizes to specific examples or data not yet seen by the model.
  - …
- **Goodness of fit** describes how well a model fits for a set of observations.
  - Overfitting and Underfitting

Overfitting

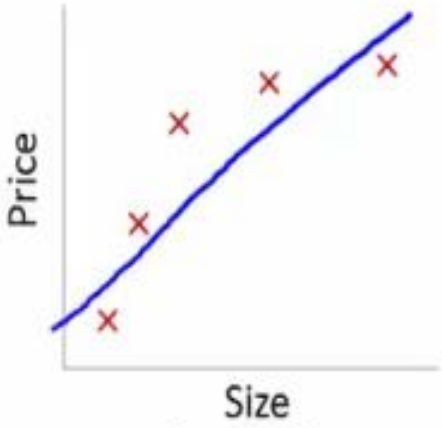
- Phenomenon of extracting too much information from training sets or memorization can cause overfitting
  - Makes ML model work well with training data called **low bias**
  - Bias refers to error due to incorrect assumptions in learning algorithm
  - However, does not generalize well or derive patterns, performs poorly on test datasets called **high variance**
  - Variance measures error due to small fluctuations in training set

Underfitting

- Model is underfit if it does not perform well on training sets and will not do so on test sets
- Occurs when we are not using enough data to train or if we try to fit wrong model to the data
  - E.g., if you do not read enough material for exam or if you prepare wrong syllabus
- Called **high bias** in ML although **variance is low** [i.e. consistent but in a bad way]
- May need to increase number of features since it expands the hypothesis space.

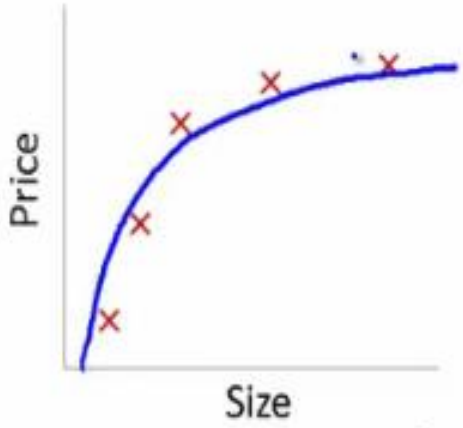
Goodness of fit

- For same data:



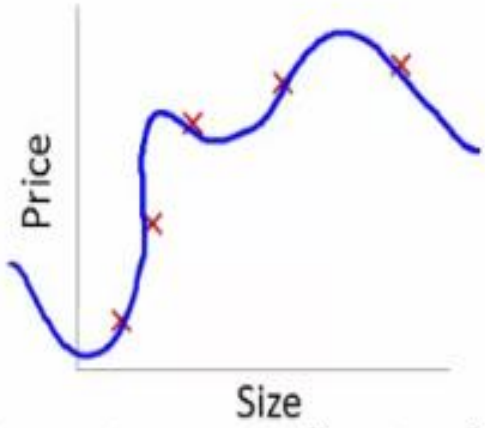
$\theta_0 + \theta_1 x$

High bias
(underfit)



$\theta_0 + \theta_1 x + \theta_2 x^2$

"Just right"



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

High variance
(overfit)

Bias-Variance Tradeoff

- If the model is too simple and has very few parameters then it may have high bias and low variance
- If the model has large number of parameters it may have high variance and low bias
- We need to find a right/good **balance** without overfitting or underfitting the data
- As more parameters are added to a model
  - Complexity of the model rises
  - Variance becomes primary concern while bias falls steadily.

Bias-Variance Tradeoff

- Suppose a training set consists of points x1, …, xn and real values yi associated with each point xi
- We assume there is a function y = f(x) + Ɛ, where the noise Ɛ has zero mean and variance σ^2
- Find $\hat{f}(x)$, tht approximates f(x) as well as possible
- To measure how well the approximation was performed, we minimize the mean square error $(y - \hat{f}(x))^2$
- A number of algorithms exist to find $\hat{f}(x)$, that generalizes to points outside of our training set

Bias-Variance Tradeoff

- Variance measures how far a set of (random) numbers are spread out from their average value.
- Measured as expectation of the squared deviation of a random variable from its mean.

$$Var(X) = E[(x-\mu)^2]$$

$$Var(X) = E[(x-E[x])^2]$$

$$= E[x^2 - 2xE[x] + E[x]^2)$$

$$= E[x^2] - 2E[x]E[x] + E[x]^2$$

$$= E[x^2] - E[x]^2$$

Bias-Variance Tradeoff

- Turns out expected (mean squared) error of $\hat{f}$ on an unseen sample in general can be decomposed as:

$$E\left[\left(y - \hat{f}(x)\right)^2\right] = (Bias[\hat{f}(x)])^2 + Var[\hat{f}(x)] + \sigma^2$$

where,

$$Bias\left(\hat{f}(x)\right) = E[\hat{f}(x) - f(x)]$$

$$= E[\hat{f}(x)] - E[f(x)] = E[\hat{f}(x)] - f(x)$$

$$Since\ f\ is\ deterministic, E[f] = f$$

and

$$Var[\hat{f}(x)] = E[\hat{f}(x)^2] - E[\hat{f}(x)]^2$$

Note that all three terms are positive

Bias-Variance Tradeoff

- Notations:

$$Var[x] = E[x^2] - (E[x])^2$$

$$E[X^2] = Var(X) + (E[x])^2$$

$$Given\ y = f + \varepsilon\ and\ E[\varepsilon] = 0, E[y] = E[f + \varepsilon] = E[f] = f$$

$$Since\ Var[\varepsilon] = \sigma^2, Var[y] = E[(y - E[y])^2] = E[(y - f)^2]$$

$$= E[(f + \varepsilon - f)^2] = E[\varepsilon^2] = Var[\varepsilon] + (E[\varepsilon])^2 = \sigma^2$$

Bias-Variance Tradeoff

- The expected error on an unseen sample x can be decomposed as:

$$
\begin{aligned}
E[(y-\hat{f})^2] &= E[(f+\varepsilon-\hat{f})^2] \\
&= E[(f+\varepsilon-\hat{f}+E[\hat{f}]-E[\hat{f}])^2] \\
&= E\left[(f-E[\hat{f}])^2\right] + E[\varepsilon^2] + E\left[(E(\hat{f})-\hat{f})^2\right] \\
&+ 2E[(f-E[\hat{f}])\varepsilon] + 2E[\varepsilon(E(\hat{f})-\hat{f})] + 2E[(E(\hat{f})-\hat{f})(f-E[\hat{f}])] \\
&= (f-E(\hat{f}))^2 + E(\varepsilon^2) + E[(E[\hat{f}]-\hat{f})^2] \\
&+ 2(f-E[\hat{f}])E(\varepsilon) + 2E(\varepsilon)E(E[\hat{f}]-\hat{f}] + 2E[E[\hat{f}]-\hat{f}](f-E[\hat{f}])
\end{aligned}
$$

Bias-Variance Tradeoff

$$= (f - E[\hat{f}])^2 + E[\varepsilon^2] + E[(E[\hat{f}] - \hat{f})^2]$$
$$= (f - E[\hat{f}])^2 + Var[y] + Var[\hat{f}]$$
$$= Bias[\hat{f}]^2 + Var[y] + Var[\hat{f}]$$
$$= Bias[\hat{f}]^2 + \sigma^2 + Var[\hat{f}]$$

- Hence the derivation.

Avoiding Overfitting

- A variety of techniques to avoid overfitting:
  - Cross-validation
  - Regularization
  - Feature selection
  - Dimensionality reduction

Non-exhaustive Cross-validation

Exhaustive Cross-validation

Nested Cross-validation

- Popular way to tune parameters of an algorithm
- One version: k-fold cross validation with validation and test set
- Lets say parameter X needs tuning
  - Possible values 10, 20, 30, 40, 50

Nested Cross-validation

- k = 7 in our example
  - One set each picked as Test and Validation, (k-2) picked for training
- For the picked Test set
  - Perform k-fold cross validation on Train & Validation set [Here k = 6]
  - Compute the average training error for each value of X
  - Pick the best X
- Repeat for each possible Test set [i.e. 7 times]
- Pick X that was returned maximum times to outer loop

Regularization

Regularization

- Let $\hat{f}(x) = \theta_0 + \theta_1 x^2 + \theta_3 x^2 + \theta_4 x^3$
- We want to minimize the MSE:

$$\frac{1}{m} * \min_{\theta_0, \theta_1, \theta_3, \theta_4} \sum_{i=1}^{m} (\hat{f}_\theta(x^{(i)}) - y^{(i)})^2$$

- where m is the number of training samples, theta's are the weight parameters
- Let MSE be represented by $J(\theta)$
- Lets say we want to penalize the higher order terms (2 and 3)

Regularization

- Can add penalty terms say +1000 θ_3 + 1000 θ_4
- The effect of this would be that θ_3 and θ_4 need to be quite small to minimize error
- A significantly high penalty can actually convert a overfit problem to an underfit problem
  - Since all the terms with high regularization parameter would become 0 or close to 0
  - E.g. if all terms except θ_0 have a high enough regularization parameter then $\hat{f}(x)$ can become a constant !!!

Feature Selection

- Filter methods: …
- Wrapper methods: …
  - Recursive Feature Elimination
- Embedded methods: …

Dimensionality Reduction

Data Preprocessing

- A popular methodology in data mining is CRoss Industry Standard Process for data mining (CRISP DM)
- …

Feature Engineering

- …
- **One-hot-encoding or one-of-K:** Refers to splitting the column which contains numerical *categorical data* to many columns depending on the number of categories present in that column.
  - Each column contains “0” or “1” corresponding to which column it has been placed.

Feature Engineering

Fruit	Categorical value of fruit	Price
apple	1	5
mango	2	10
apple	1	15
orange	3	20

- After one hot encoding

apple	mango	orange	price
1	0	0	5
0	1	0	10
1	0	0	15
0	0	1	20

Feature Engineering

- …

Overview of Data Analytics: Data Mining & Warehousing

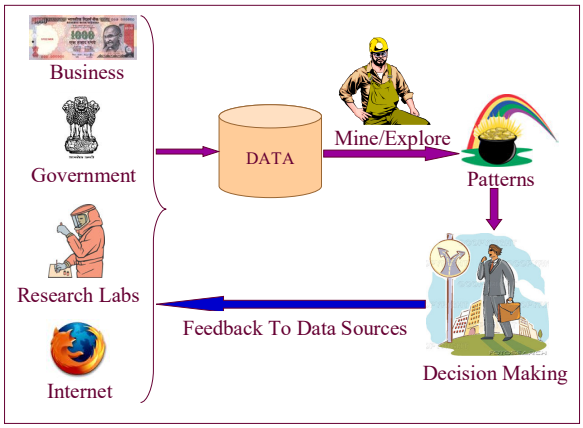
Vikram Pudi
vikram@iiit.ac.in
IIIT Hyderabad

Originated from DB community…

- Traditional Database Systems
  - Indexing
  - Query languages
  - Query optimization
  - Transaction processing
  - Recovery …
- XML, Semantic web
- OO and OR DBMS …
- *Data Mining*

Data Mining

Automated extraction of interesting patterns from large databases



Types of Patterns

- Associations
  - *Coffee* buyers usually also purchase *sugar*
- Clustering
  - Segments of customers requiring different promotion strategies
- Classification
  - Customers expected to be *loyal*

Association Rules

That which is infrequent is not worth worrying about.

Association Rules

D :

Transaction ID	Items
1	Tomato, Potato, Onions
2	Tomato, Potato, Brinjal, Pumpkin
3	Tomato, Potato, Onions, Chilly
4	Lemon, Tamarind

Rule: Tomato, Potato → Onion (confidence: 66%, support: 50%)

Support(X) = |transactions containing X| / |D|

Confidence(R) = support(R) / support(LHS(R))

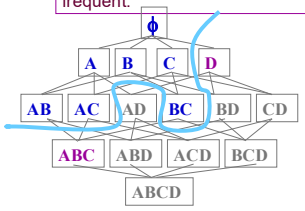
Problem proposed in [AIS 93]: Find all rules satisfying user given minimum support and minimum confidence.

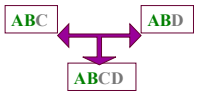
Association Rule Applications

- E-commerce
  - People who have bought *Sundara Kandam* have also bought *Srimad Bhagavatham*
- Census analysis
  - *Immigrants* are usually *male*
- Sports
  - A chess end-game configuration with "*white pawn on A7*" and "*white knight dominating black rook*" typically results in a "*win for white*".
- Medical diagnosis
  - Allergy to *latex rubber* usually co-occurs with allergies to *banana* and *tomato*

The Apriori Algorithm

Idea: An itemset can be frequent only if all its subsets are frequent.



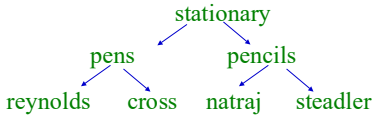



```
Apriori( DB, minsup ):
C = {all 1-itemsets}
        // candidates = singletons
while ( |C| > 0 ):
    make pass over DB, find counts of C
    F = sets in C with count ≥ minsup*|DB|
    output F
    C = AprioriGen(F) // gen. candidates
```

```
AprioriGen( F ):
for each pair of itemsets X, Y in F:
    if X and Y share all items, except last
        Z = X ∪ Y // generate candidate
        if any imm. subset of Z is not in F:
            prune Z // Z can't be frequent
```


Types of Association Rules

- Boolean association rules
- Hierarchical rules



reynolds → pencils

- Quantitative & Categorical rules
  - (Age: 30…39), (Married: Yes) → (NumCars: 2)

More Types of Association Rules

- Cyclic / Periodic rules
  - Sunday → vegetables
  - Christmas → gift items
  - Summer, rich, jobless → ticket to Hawaii
- Constrained rules
  - Show itemsets whose average price > Rs.10,000
  - Show itemsets that have television on RHS
- Sequential rules
  - Star wars, Empire Strikes Back → Return of the Jedi

Classification

To be or not to be: That is the question.

- William Shakespeare

The Classification Problem

Outlook	Temp (°F)	Humidity (%)	Windy?	Class
sunny	75	70	true	*play*
sunny	80	90	true	*don't play*
sunny	85	85	false	*don't play*
sunny	72	95	false	*don't play*
sunny	69	70	false	*play*
overcast	72	90	true	*play*
overcast	83	78	false	*play*
overcast	64	65	true	*play*
overcast	81	75	false	*play*
rain	71	80	true	*don't play*
rain	65	70	true	*don't play*
rain	75	80	false	*play*
rain	68	80	false	*play*
rain	70	96	false	*play*
sunny	77	69	true	?
rain	73	76	false	?

Play Outside?

Model relationship between class labels and attributes

e.g. outlook = overcast ⇒ class = *play*

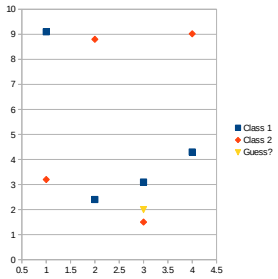
⇒ Assign class labels to new data with *unknown* labels

Applications

- Text classification
  - Classify emails into spam / non-spam
  - Classify web-pages into yahoo-type hierarchy
  - NLP Problems
    - Tagging: Classify words into verbs, nouns, etc.
- Risk management, Fraud detection, Computer intrusion detection
  - Given the properties of a transaction (items purchased, amount, location, customer profile, etc.)
  - Determine if it is a fraud
- Machine learning / pattern recognition applications
  - Vision
  - Speech recognition
  - etc.
- All of science & knowledge is about predicting future in terms of past
  - So classification is a very fundamental problem with ultra-wide scope of applications

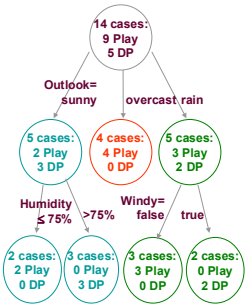
k-Nearest Neighbours

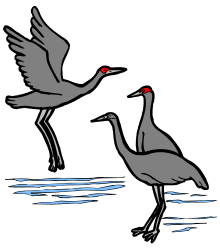
- Model = Training data
- Classify record R using the *k* nearest neighbours of R in the training data.
- Most frequent class among *k* NNs
- Distance function could be euclidean
- Use an index structure (e.g. R* tree) to find the *k* NNs efficiently



Decision Trees

Outlook	Temp (°F)	Humidity (%)	Windy?	Class
sunny	75	70	true	*play*
sunny	80	90	true	*don't play*
sunny	85	85	false	*don't play*
sunny	72	95	false	*don't play*
sunny	69	70	false	*play*
overcast	72	90	true	*play*
overcast	83	78	false	*play*
overcast	64	65	true	*play*
overcast	81	75	false	*play*
rain	71	80	true	*don't play*
rain	65	70	true	*don't play*
rain	75	80	false	*play*
rain	68	80	false	*play*
rain	70	96	false	*play*





Clustering

Birds of a feather flock together.

The Clustering Problem

Outlook	Temp (°F)	Humidity (%)	Windy?
sunny	75	70	true
sunny	80	90	true
sunny	85	85	false
sunny	72	95	false
sunny	69	70	false
overcast	72	90	true
overcast	73	88	true
overcast	64	65	true
overcast	81	75	false
rain	71	80	true
rain	65	70	true
rain	75	80	false
rain	68	80	false
rain	70	96	false

Find groups of similar records.

Need a function to compute similarity, given 2 input records

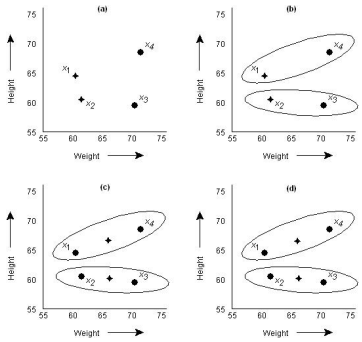
⇒ Unsupervised learning

Applications

- Targetting similar people or objects
  - Student tutorial groups
  - Hobby groups
  - Health support groups
  - Customer groups for marketing
  - Organizing e-mail
- Spatial clustering
  - Exam centres
  - Locations for a business chain
  - Planning a political strategy

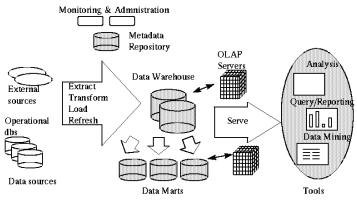
Partitioning technique: *k*-Means

- Initial *k* means = random records
- Iterate as long as clusters change:
  - Put each record X in the cluster to whose mean it is closest
  - Recompute means as the average of all points in each cluster



Data Warehousing

- Extract, transform, load data from multiple sources in an enterprise
- Provide unified view for top management
- OLAP server provides multi-dimensional view for manual exploration of patterns



Examples of OLAP

Comparisons (this period v.s. last period)

Show me the sales per store for this year and compare it to that of the previous year to identify discrepancies

Ranking and statistical profiles (top N/bottom N)

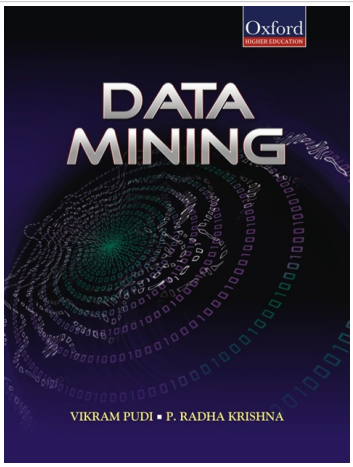
Show me sales, profit and average call volume per day for my 10 most profitable salespeople

Custom consolidation (market segments, ad hoc groups)

Show me an abbreviated income statement by quarter for the last four quarters for my northeast region operations

Take Home

- Data mining is a mature field
- Don't waste time developing new algorithms for core tasks
- Focus on applications to challenging kinds of data
  - Streams, Distributed data, Multimedia, Web, …
- Most effort is in how to map domain problems to data mining problems
- And how to make sense of the output.



Classification

Vikram Pudi
vikram@iiit.ac.in
IIIT Hyderabad

Talk Outline

- Introduction
  - Classification Problem
  - Applications
  - Metrics
  - Combining classifiers
- Classification Techniques

The Classification Problem

Outlook	Temp (°F)	Humidity (%)	Windy?	Class
sunny	75	70	true	*play*
sunny	80	90	true	*don't play*
sunny	85	85	false	*don't play*
sunny	72	95	false	*don't play*
sunny	69	70	false	*play*
overcast	72	90	true	*play*
overcast	83	78	false	*play*
overcast	64	65	true	*play*
overcast	81	75	false	*play*
rain	71	80	true	*don't play*
rain	65	70	true	*don't play*
rain	75	80	false	*play*
rain	68	80	false	*play*
rain	70	96	false	*play*
sunny	77	69	true	?
rain	73	76	false	?

Play Outside?

Model relationship between class labels and attributes

e.g. outlook = overcast ⇒ class = *play*

⇒ Assign class labels to new data with *unknown* labels

Applications

- Text classification
  - Classify emails into spam / non-spam
  - Classify web-pages into yahoo-type hierarchy
  - NLP Problems
    - Tagging: Classify words into verbs, nouns, etc.
- Risk management, Fraud detection, Computer intrusion detection
  - Given the properties of a transaction (items purchased, amount, location, customer profile, etc.)
  - Determine if it is a fraud
- Machine learning / pattern recognition applications
  - Vision
  - Speech recognition
  - etc.
- All of science & knowledge is about predicting future in terms of past
  - So classification is a very fundamental problem with ultra-wide scope of applications

Metrics

1. accuracy
2. classification time per new record
3. training time
4. main memory usage (during classification)
5. model size

Accuracy Measure

- Prediction is just like tossing a coin (random variable X)
  - "Head" is "success" in classification; X = 1
  - "tail" is "error"; X = 0
  - X is actually a mapping: {"success": 1, "error" : 0}
- In statistics, a succession of independent events like this is called a *bernoulli process*
  - Accuracy = P(X = 1) = p
  - mean value = μ = E[X] = $p \times 1 + (1-p) \times 0 = p$
  - variance = σ^2 = E[$(X-\mu)^2$] = $p\,(1-p)$
- Confidence intervals: Instead of saying accuracy = 85%, we want to say: accuracy $\in$ [83, 87] with a confidence of 95%

Binomial Distribution

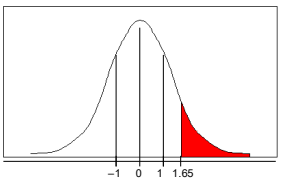
- Treat each classified record as a bernoulli trial
- If there are *n* records, there are *n* independent and identically distributed (iid) bernoulli trials, X_i, i = 1,…,*n*
- Then, the random variable $X = \sum_{i=1,\ldots,n} X_i$ is said to follow a *binomial distribution*
  - $P(X = k) = {}^nC_k\, p^k\, (1-p)^{n-k}$
- Problem: Difficult to compute for large *n*

Normal Distribution

- Continuous distribution with parameters μ (mean), σ^2(variance)
- Probability density:
  $f(x) = (1/\sqrt{(2\pi\sigma^2)})\, \exp(-(x-\mu)^2 / (2\sigma^2))$
- Central limit theorem:
  - Under certain conditions, the distribution of the sum of a *large number* of iid random variables is approximately normal
  - A *binomial distribution* with parameters *n* and *p* is approximately normal for large *n* and *p* not too close to 1 or 0
  - The approximating normal distribution has mean μ = *np* and standard deviation $\sigma^2 = (n\, p\, (1 - p))$

Confidence Intervals

Normal distribution with mean = 0 and variance = 1



Pr[$X \geq z$]	z
0.1%	3.09
0.5%	2.58
1%	2.33
5%	1.65
10%	1.28
20%	0.84
40%	0.25

- E.g. $P[-1.65 \leq X \leq 1.65] = 1 - 2 \times P[X \geq 1.65] = 90\%$
- To use this we have to transform our random variable to have mean = 0 and variance = 1
- Subtract mean from *X* and divide by standard deviation

Estimating Accuracy

- Holdout method
  - Randomly partition data: training set + test set
  - accuracy = |correctly classified points| / |test data points|
- Stratification
  - Ensure each class has approximately equal proportions in both partitions
- Random subsampling
  - Repeat holdout *k* times. Output average accuracy.
- *k*-fold cross-validation
  - Randomly partition data: $S_1, S_2, \ldots, S_k$
  - First, keep S_1 as test set, remaining as training set
  - Next, keep S_2 as test set, remaining as training set, etc.
  - accuracy = |*total* correctly classified points| / |total data points|
- Recommendation:
  - Stratified 10-fold cross-validation. If possible, repeat 10 times and average results. (reduces variance)

Is Accuracy Enough?

- If only 1% population has cancer, then a test for cancer that classifies *all* people as *non-cancer* will have 99% accuracy.
- Instead output a confusion matrix:

Actual/ Estimate	Class 1	Class 2	Class 3
Class 1	90%	5%	5%
Class 2	2%	91%	7%
Class 3	8%	3%	89%

Combining Classifiers

- Get *k* random samples with replacement as training sets (like in random subsampling).

⇒ We get *k* classifiers

- Bagging: Take a *majority vote* for the best class for each new record
- Boosting: Each classifier's vote has a *weight* proportional to its accuracy on training data

⇒ Like a patient taking multiple opinions from several doctors

Talk Outline

- Introduction
- Classification Techniques
  1. Nearest Neighbour Methods
  2. Decision Trees
     - ID3, CART, C4.5, C5.0, SLIQ, SPRINT
  3. Bayesian Methods
     - Naïve Bayes, Bayesian Belief Networks
     - Maximum Entropy Based Approaches
  4. Association Rule Based Approaches
  5. Soft-computing Methods:
     - Genetic Algorithms, Rough Sets, Fuzzy Sets, Neural Networks
  6. Support Vector Machines
  7. Convolutional Neural Networks, Deep Learning

Nearest Neighbour Methods

k-NN, Reverse Nearest Neighbours

k-Nearest Neighbours

- Model = Training data
- Classify record R using the *k* nearest neighbours of R in the training data.
- Most frequent class among *k* NNs
- Distance function could be euclidean
- Use an index structure (e.g. R* tree) to find the *k* NNs efficiently

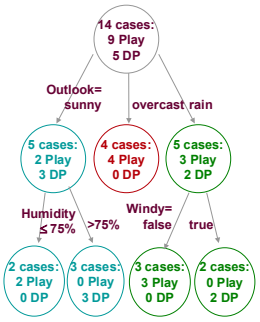
Reverse Nearest Neighbours

- Records which consider *R* as a *k*-NN
- Output most frequent class among RNNs.
- More resilient to outliers.

Decision Trees

Decision Trees

Outlook	Temp (°F)	Humidity (%)	Windy?	Class
sunny	75	70	true	*play*
sunny	80	90	true	*don't play*
sunny	85	85	false	*don't play*
sunny	72	95	false	*don't play*
sunny	69	70	false	*play*
overcast	72	90	true	*play*
overcast	83	78	false	*play*
overcast	64	65	true	*play*
overcast	81	75	false	*play*
rain	71	80	true	*don't play*
rain	65	70	true	*don't play*
rain	75	80	false	*play*
rain	68	80	false	*play*
rain	70	96	false	*play*



Basic Tree Building Algorithm

MakeTree (Training Data D):
    Partition(D)

Partition (Data D):
    if all points in D are in same class: return
    Evaluate splits for each attribute A
    Use best split found to partition D into $D_1, D_2, \ldots, D_n$
    for each D_i:
        Partition (D_i)

ID3, CART

ID3
- Use *information gain* to determine best split
- *gain* = $H(D) - \sum_{i=1\ldots n} P(D_i)\, H(D_i)$
- $H(p_1, p_2, \ldots, p_m) = -\sum_{i=1\ldots m} p_i \log p_i$
- like 20-question game
  - Which attribute is better to look for first: "Is it a living thing?" or "Is it a duster?"

CART
- Only create two children for each node
- Goodness of a split (Φ)
  $\Phi = 2\, P(D_1)\, P(D_2) \sum_{i=1\ldots m} | P(C_j / D_1) - P(C_j / D_2) |$

Shannon's Entropy

- An expt has several possible outcomes
- In N expts, suppose each outcome occurs M times
- This means there are N/M possible outcomes
- To represent each outcome, we need log N/M bits.
  - This generalizes even when all outcomes are not equally frequent.
  - Reason: For an outcome j that occurs M times, there are N/M equi-probable events among which only one cp to j
- Since p_i = M / N, information content of an outcome is -log p_i
- So, expected info content: $H = -\Sigma\, p_i \log p_i$

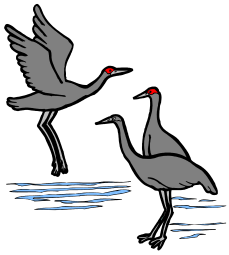
Bayesian Methods

Naïve Bayes

- New data point to classify: $X=(x_1, x_2, \ldots x_m)$
- Strategy:
  - Calculate $P(C_i/X)$ for each class C_i.
  - Select C_i for which $P(C_i/X)$ is maximum

$$P(C_i/X) = P(X/C_i)\, P(C_i) / P(X)$$
$$\propto P(X/C_i)\, P(C_i)$$
$$\propto P(x_1/C_i)\, P(x_2/C_i) \ldots P(x_m/C_i)\, P(C_i)$$

- Naïvely assumes that each x_i is independent
- We represent $P(X/C_i)$ by $P(X)$, etc. when unambiguous



Clustering

Birds of a feather flock together.

Vikram Pudi
vikram@iiit.ac.in
IIIT Hyderabad

The Clustering Problem

Outlook	Temp (°F)	Humidity (%)	Windy?
sunny	75	70	true
sunny	80	90	true
sunny	85	85	false
sunny	72	95	false
sunny	69	70	false
overcast	72	90	true
overcast	73	88	true
overcast	64	65	true
overcast	81	75	false
rain	71	80	true
rain	65	70	true
rain	75	80	false
rain	68	80	false
rain	70	96	false

Find groups of similar records.

Need a function to compute similarity, given 2 input records

⇒ Unsupervised learning

Applications

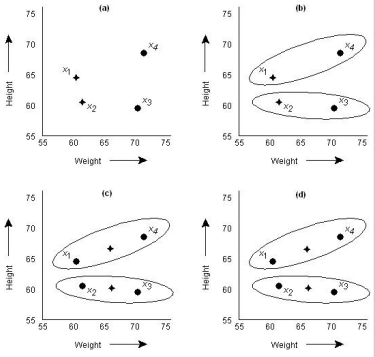
- Targetting similar people or objects
  - Student tutorial groups
  - Hobby groups
  - Health support groups
  - Customer groups for marketing
  - Organizing e-mail
- Spatial clustering
  - Exam centres
  - Locations for a business chain
  - Planning a political strategy

Measurement of similarity

- Nominal (categorical) variables
  d(x,y) = 1 - m/n
  m = no of matches among n attributes, or
  m = sum of weights of matching attributes, and n is the sum of weights of all attributes
- Numeric variables
  - Euclidean, manhattan, minkowski,…
  - Ordinal
    - z = (rank-1)/(M-1) where M is maximum rank
- Above are examples
  - Similarity is ultimately application dependent
  - Requires various kinds of preprocessing
    - Scaling: Convert all attributes to have same range
    - z-score: z = (value-mean)/m
      where m is the mean absolute deviation

Partitioning technique: *k*-Means

- Initial *k* means = random records
- Iterate as long as clusters change:
  - Put each record X in the cluster to whose mean it is closest
  - Recompute means as the average of all points in each cluster



Evaluating Clustering Quality

- Minimize squared error
  Here m_i is the mean (or other centre) of cluster i
- Can also use absolute error
- Can be used to find best initial random means in *k*-means.

$$E = \sum_{i=1}^{N} \sum_{x \in C_i} d(x, m_i)^2$$

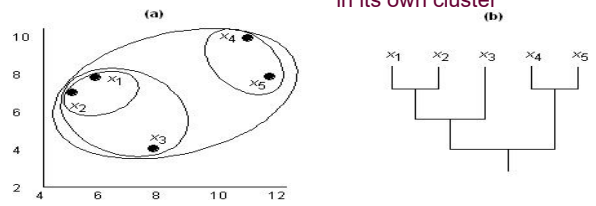
Hierarchical Methods

Agglomerative (e.g. AGNES):

- Start: Each point in separate cluster
- Merge 2 closest clusters
- Repeat until all records are in 1 cluster.

Divisive (e.g. DIANA)

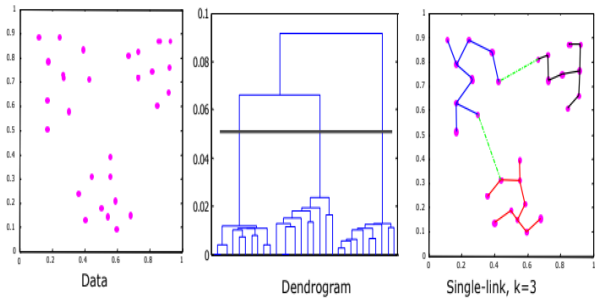
- Start: All points in 1 cluster
- Find most extreme points in each cluster.
- Regroup points based on closest extreme point
- Repeat until each record is in its own cluster



Measuring Cluster Distances

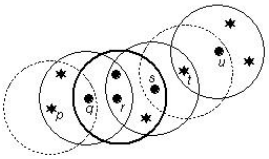
- Single link: Minimum distance
- Complete link: Maximum distance
- Average link: Average distance

Single Link Algorithm



Density-based Methods: e.g. DBSCAN

- Neighbourhood: Records within distance of ε from given record.
- Core point: Record whose neighbourhood contains at least μ records.
- Find all core points and create a cluster for each of them.
- If core point Y is in the neighbourhood of core point X, then merge the clusters of X and Y.
- Repeat above step for all core points until clusters do not change.



Mining Outliers using Clustering

- Outliers are data points that deviate significantly from the norm.
- Useful in fraud detection, error detection (in data cleaning), etc.
- Technique:
  - Apply any clustering algorithm
  - Treat clusters of very small size as containing only outliers

Solving problems by searching

From AIMA slides

Outline

- Problem-solving agents
- Problem types
- Problem formulation
- Example problems
- Basic search algorithms

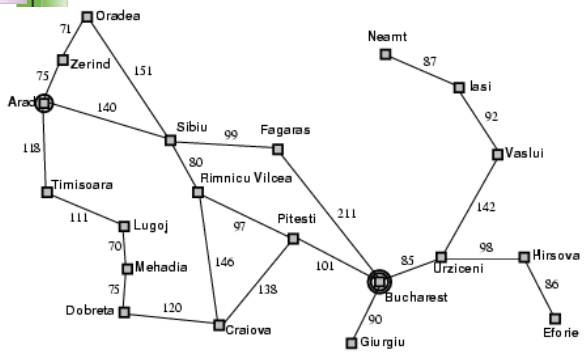
Problem-solving agents

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  static: seq, an action sequence, initially empty
          state, some description of the current world state
          goal, a goal, initially null
          problem, a problem formulation
  state ← UPDATE-STATE(state, percept)
  if seq is empty then do
      goal ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, goal)
      seq ← SEARCH(problem)
  action ← FIRST(seq)
  seq ← REST(seq)
  return action
```


Example: Romania

- On holiday in Romania; currently in Arad.
- Flight leaves tomorrow from Bucharest
- Formulate goal:
  - be in Bucharest
- Formulate problem:
  - states: various cities
  - actions: drive between cities
- Find solution:
  - sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

Example: Romania

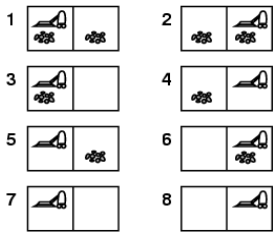


Problem types

- Deterministic, fully observable → single-state problem
  - Agent knows exactly which state it will be in; solution is a sequence
- Non-observable → sensorless problem (conformant problem)
  - Agent may have no idea where it is; solution is a sequence
- Nondeterministic and/or partially observable → contingency problem
  - percepts provide new information about current state
  - often interleave} search, execution
- Unknown state space → exploration problem

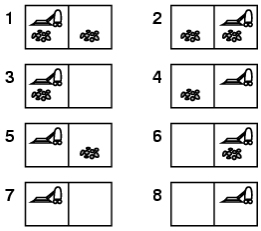
Example: vacuum world

- Single-state, start in #5.
  Solution?



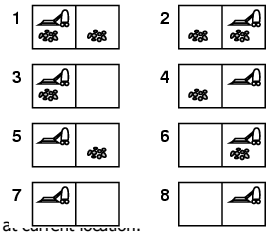
Example: vacuum world

- Single-state, start in #5.
  Solution? *[Right, Suck]*
- Sensorless, start in *{1,2,3,4,5,6,7,8}* e.g., *Right* goes to *{2,4,6,8}*
  Solution?



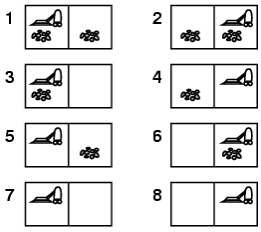
Example: vacuum world

- Sensorless, start in *{1,2,3,4,5,6,7,8}* e.g., *Right* goes to *{2,4,6,8}*
  Solution?
  [Right,Suck,Left,Suck]
- Contingency
  - Nondeterministic: *Suck* may dirty a clean carpet
  - Partially observable: location, dirt at current location.
  - Percept: *[L, Clean]*, i.e., start in #5 or #7
    Solution?



Example: vacuum world

- Sensorless, start in *{1,2,3,4,5,6,7,8}* e.g., *Right* goes to *{2,4,6,8}*
  Solution?
  [Right,Suck,Left,Suck]
- Contingency
  - Nondeterministic: *Suck* may dirty a clean carpet
  - Partially observable: location, dirt at current location.
  - Percept: *[L, Clean]*, i.e., start in #5 or #7
    Solution? *[Right, **if** dirt **then** Suck]*



Single-state problem formulation

A problem is defined by four items:

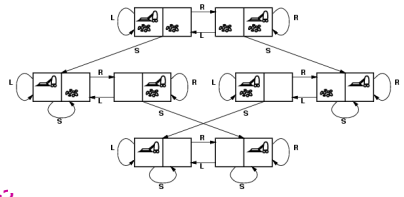
1. initial state e.g., "at Arad"
2. actions or successor function *S(x)* = set of action–state pairs
   - e.g., *S(Arad)* = *{<Arad → Zerind, Zerind>, … }*
3. goal test, can be
   - explicit, e.g., *x* = "at Bucharest"
   - implicit, e.g., *Checkmate(x)*
4. path cost (additive)
   - e.g., sum of distances, number of actions executed, etc.
   - *c(x,a,y)* is the step cost, assumed to be ≥ 0

- A solution is a sequence of actions leading from the initial state to a goal state

Selecting a state space

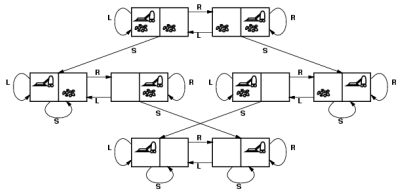
- Real world is absurdly complex
  → state space must be abstracted for problem solving
- (Abstract) state = set of real states
- (Abstract) action = complex combination of real actions
  - e.g., "Arad → Zerind" represents a complex set of possible routes, detours, rest stops, etc.
- For guaranteed realizability, any real state "in Arad" must get to some real state "in Zerind"
- (Abstract) solution =
  - set of real paths that are solutions in the real world
- Each abstract action should be "easier" than the original problem

Vacuum world state space graph



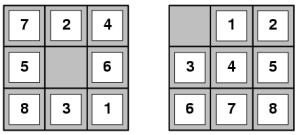
- states?
- actions?
- goal test?
- path cost?

Vacuum world state space graph



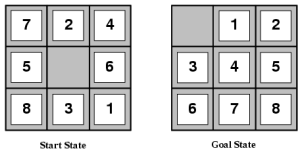
- states? integer dirt and robot location
- actions? *Left*, *Right*, *Suck*
- goal test? no dirt at all locations
- path cost? 1 per action

Example: The 8-puzzle



- states?
- actions?
- goal test?
- path cost?

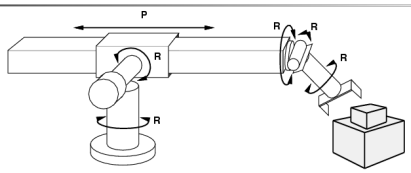
Example: The 8-puzzle



- states? locations of tiles
- actions? move blank left, right, up, down
- goal test? = goal state (given)
- path cost? 1 per move

[Note: optimal solution of *n*-Puzzle family is NP-hard]

Example: robotic assembly



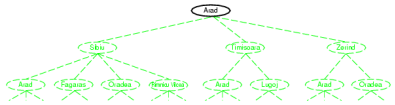
- states?: real-valued coordinates of robot joint angles parts of the object to be assembled
- actions?: continuous motions of robot joints
- goal test?: complete assembly
- path cost?: time to execute

Tree search algorithms

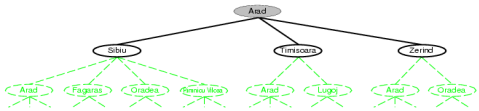
- Basic idea:
  - offline, simulated exploration of state space by generating successors of already-explored states (a.k.a.~expanding states)

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
```

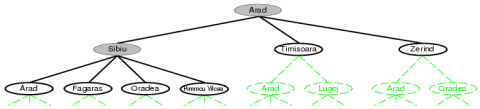
Tree search example



Tree search example



Tree search example

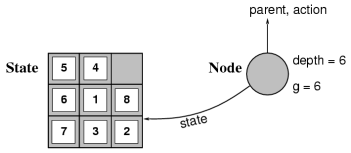


Implementation: general tree search

```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
        fringe ← INSERT ALL(EXPAND(node, problem), fringe)

function EXPAND( node, problem) returns a set of nodes
    successors ← the empty set
    for each action, result in SUCCESSOR-FN[problem](STATE[node]) do
        s ← a new NODE
        PARENT-NODE[s] ← node;  ACTION[s] ← action;  STATE[s] ← result
        PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
        DEPTH[s] ← DEPTH[node] + 1
        add s to successors
    return successors
```


Implementation: states vs. nodes

- A state is a (representation of) a physical configuration
- A node is a data structure constituting part of a search tree includes state, parent node, action, path cost *g(x)*, depth



- The `Expand` function creates new nodes, filling in the various fields and using the `SuccessorFn` of the problem to create the corresponding states.

Search strategies

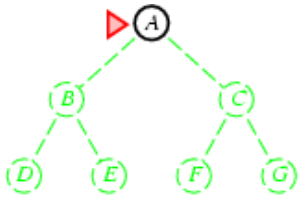
- A search strategy is defined by picking the order of node expansion
- Strategies are evaluated along the following dimensions:
  - completeness: does it always find a solution if one exists?
  - time complexity: number of nodes generated
  - space complexity: maximum number of nodes in memory
  - optimality: does it always find a least-cost solution?
- Time and space complexity are measured in terms of
  - *b:* maximum branching factor of the search tree
  - *d:* depth of the least-cost solution
  - *m*: maximum depth of the state space (may be ∞)

Uninformed search strategies

- Uninformed search strategies use only the information available in the problem definition
- Breadth-first search
- Uniform-cost search
- Depth-first search
- Depth-limited search
- Iterative deepening search

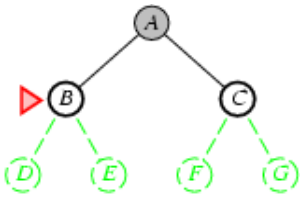
Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end



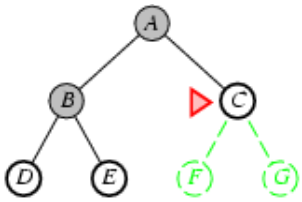
Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end



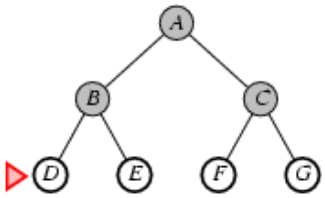
Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end



Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end



Properties of breadth-first search

- Complete? Yes (if b is finite)
- Time? $1+b+b^2+b^3+\ldots+b^d + b(b^d-1) = O(b^{d+1})$
- Space? $O(b^{d+1})$ (keeps every node in memory)
- Optimal? Yes (if cost = 1 per step)

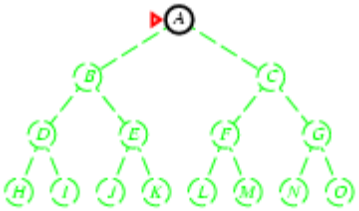
- Space is the bigger problem (more than time)

Uniform-cost search

- Expand least-cost unexpanded node
- Implementation:
  - *fringe* = queue ordered by path cost
- Equivalent to breadth-first if step costs all equal
- Complete? Yes, if step cost ≥ ε
- Time? # of nodes with $g \leq$ cost of optimal solution, $O(b^{\text{ceiling}(C^*/\varepsilon)})$ where C^* is the cost of the optimal solution
- Space? # of nodes with $g \leq$ cost of optimal solution, $O(b^{\text{ceiling}(C^*/\varepsilon)})$
- Optimal? Yes – nodes expanded in increasing order of $g(n)$

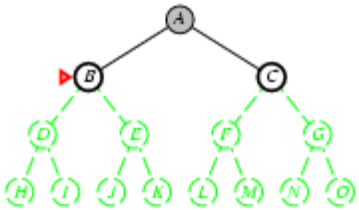
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



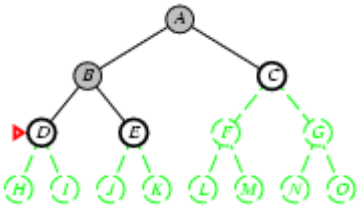
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front
  -
-



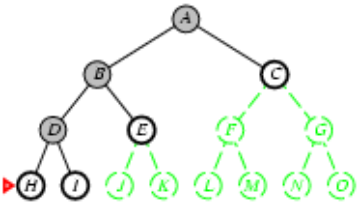
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



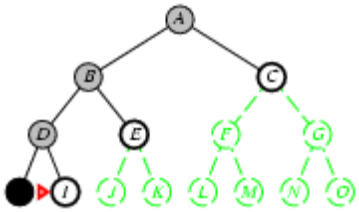
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



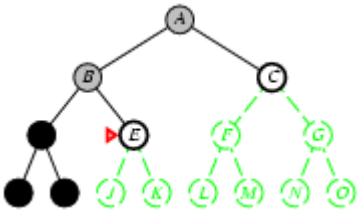
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



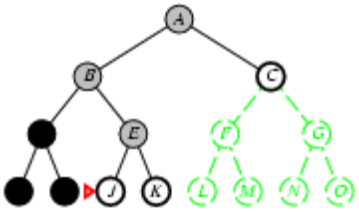
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



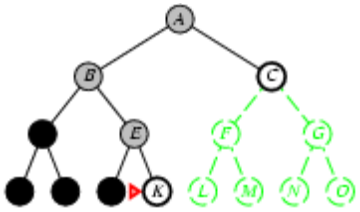
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



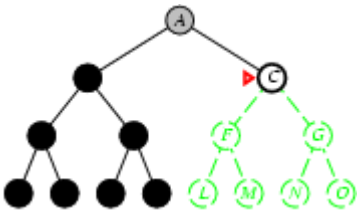
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



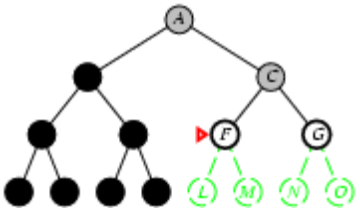
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



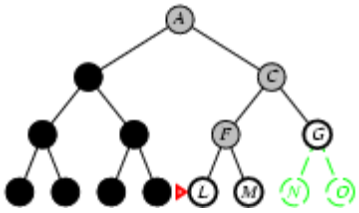
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



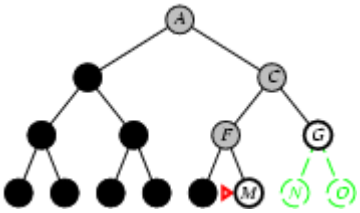
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



Properties of depth-first search

- Complete? No: fails in infinite-depth spaces, spaces with loops
  - Modify to avoid repeated states along path
    → complete in finite spaces
- Time? $O(b^m)$: terrible if m is much larger than d
  - but if solutions are dense, may be much faster than breadth-first
- Space? $O(bm)$, i.e., linear space!
- Optimal? No

Depth-limited search

= depth-first search with depth limit *l*
i.e., nodes at depth *l* have no successors

- Recursive implementation:

```
function DEPTH-LIMITED-SEARCH( problem, limit) returns soln/fail/cutoff
  RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)
function RECURSIVE-DLS(node, problem, limit) returns soln/fail/cutoff
  cutoff-occurred? ← false
  if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
  else if DEPTH[node] = limit then return cutoff
  else for each successor in EXPAND(node, problem) do
      result ← RECURSIVE-DLS(successor, problem, limit)
      if result = cutoff then cutoff-occurred? ← true
      else if result ≠ failure then return result
  if cutoff-occurred? then return cutoff else return failure
```

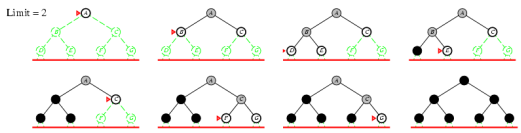

Iterative deepening search

```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution, or fail-
ure
  inputs: problem, a problem
  for depth ← 0 to ∞ do
    result ← DEPTH-LIMITED-SEARCH( problem, depth)
    if result ≠ cutoff then return result
```

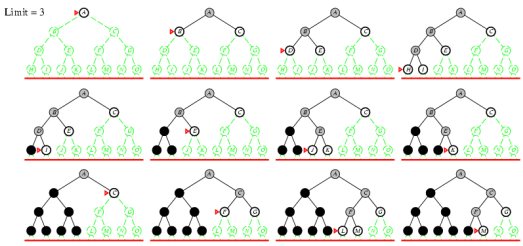

Iterative deepening search *l* =0

Iterative deepening search *l* =1

Iterative deepening search l =2



Iterative deepening search l =3



Iterative deepening search

- Number of nodes generated in a depth-limited search to depth *d* with branching factor *b*:

  $N_{DLS} = b^0 + b^1 + b^2 + \ldots + b^{d-2} + b^{d-1} + b^d$

- Number of nodes generated in an iterative deepening search to depth *d* with branching factor *b*:

  $N_{IDS} = (d+1)b^0 + d\,b\text{^}1 + (d-1)b\text{^}2 + \ldots + 3b^{d-2} + 2b^{d-1} + 1b^d$

- For *b = 10, d = 5*,
  - $N_{DLS} = 1 + 10 + 100 + 1{,}000 + 10{,}000 + 100{,}000 = 111{,}111$
  - $N_{IDS} = 6 + 50 + 400 + 3{,}000 + 20{,}000 + 100{,}000 = 123{,}456$
- Overhead = (123,456 - 111,111)/111,111 = 11%

Properties of iterative deepening search

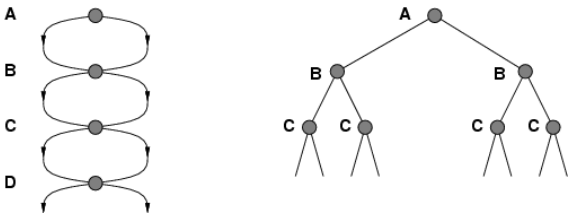
- Complete? Yes
- Time? $(d+1)b^0 + d\,b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
- Space? $O(bd)$
- Optimal? Yes, if step cost = 1

Summary of algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

Repeated states

- Failure to detect repeated states can turn a linear problem into an exponential one!



Graph search

```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
```


Summary

- Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored
- Variety of uninformed search strategies
- Iterative deepening search uses only linear space and not much more time than other uninformed algorithms

Informed search algorithms

From AIMA Slides

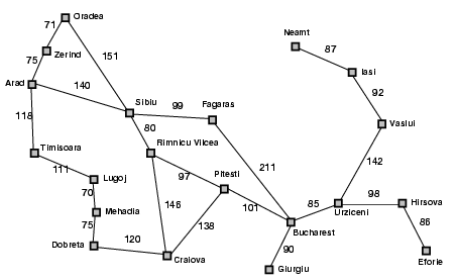
Outline

- Best-first search
- Greedy best-first search
- A* search
- Heuristics
- Local search algorithms
- Hill-climbing search
- Simulated annealing search
- Local beam search
- Genetic algorithms

Best-first search

- Idea: use an evaluation function *f(n)* for each node
  - estimate of "desirability"
  - → Expand most desirable unexpanded node
- Implementation:
  Order the nodes in fringe in decreasing order of desirability
- Special cases:
  - greedy best-first search
  - A* search

Romania with step costs in km



Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search

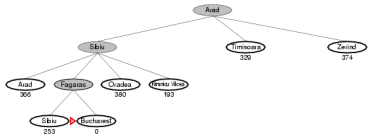
- Evaluation function *f(n)* = *h(n)* (heuristic)
- = estimate of cost from *n* to *goal*
- e.g., $h_{SLD}(n)$ = straight-line distance from *n* to Bucharest
- Greedy best-first search expands the node that appears to be closest to goal

Greedy best-first search example

Greedy best-first search example

Greedy best-first search example

Greedy best-first search example



Properties of greedy best-first search

- Complete? No – can get stuck in loops, e.g., Iasi → Neamt → Iasi → Neamt →
- Time? $O(b^m)$, but a good heuristic can give dramatic improvement
- Space? $O(b^m)$ -- keeps all nodes in memory
- Optimal? No

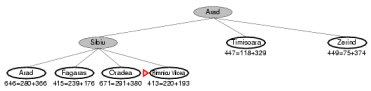
A* search

- Idea: avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
- $g(n)$ = cost so far to reach n
- $h(n)$ = estimated cost from n to goal
- $f(n)$ = estimated total cost of path through n to goal

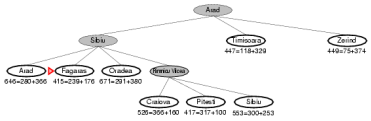
A* search example

A* search example

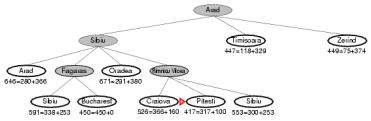
A* search example



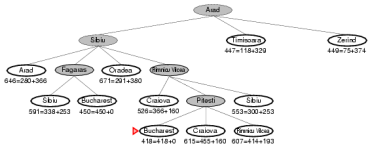
A* search example



A* search example



A* search example

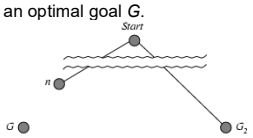


Admissible heuristics

- A heuristic *h(n)* is admissible if for every node *n*, $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from *n*.
- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
- Example: $h_{SLD}(n)$ (never overestimates the actual road distance)
- Theorem: If *h(n)* is admissible, A* using `TREE-SEARCH` is optimal

Optimality of A* (proof)

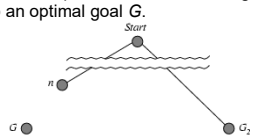
- Suppose some suboptimal goal G_2 has been generated and is in the fringe. Let n be an unexpanded node in the fringe such that n is on a shortest path to an optimal goal G.



- $f(G_2) = g(G_2)$ since $h(G_2) = 0$
- $g(G_2) > g(G)$ since G_2 is suboptimal
- $f(G) = g(G)$ since $h(G) = 0$
- $f(G_2) > f(G)$ from above

Optimality of A* (proof)

- Suppose some suboptimal goal G_2 has been generated and is in the fringe. Let n be an unexpanded node in the fringe such that n is on a shortest path to an optimal goal G.



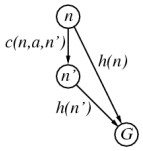
- $f(G_2)$ > $f(G)$ from above
- $h(n)$ ≤ $h^*(n)$ since h is admissible
- $g(n) + h(n)$ ≤ $g(n) + h^*(n)$
- $f(n)$ ≤ $f(G)$

Hence $f(G_2) > f(n)$, and A* will never select G_2 for expansion

Consistent heuristics

- A heuristic is consistent if for every node n, every successor n' of n generated by any action a,

$h(n) \le c(n,a,n') + h(n')$



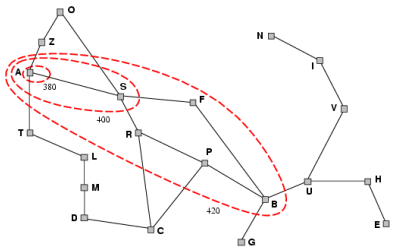
- If h is consistent, we have

$f(n') = g(n') + h(n')$
$= g(n) + c(n,a,n') + h(n')$
$\ge g(n) + h(n)$
$= f(n)$

- i.e., $f(n)$ is non-decreasing along any path.
- Theorem: If $h(n)$ is consistent, A* using GRAPH-SEARCH is optimal

Optimality of A*

- A* expands nodes in order of increasing f value
- Gradually adds "f-contours" of nodes
- Contour i has all nodes with $f=f_i$, where $f_i < f_{i+1}$



Properties of A*

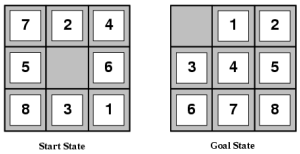
- Complete? Yes (unless there are infinitely many nodes with f ≤ $f(G)$)
- Time? Exponential
- Space? Keeps all nodes in memory
- Optimal? Yes

Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

(i.e., no. of squares from desired location of each tile)



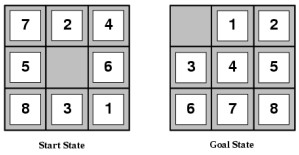
- $h_1(S)$ = ?
- $h_2(S)$ = ?

Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

(i.e., no. of squares from desired location of each tile)



- $h_1(S)$ = ? 8
- $h_2(S)$ = ? 3+1+2+2+2+3+3+2 = 18

Dominance

- If $h_2(n) \geq h_1(n)$ for all n (both admissible)
- then h_2 dominates h_1
- h_2 is better for search

- Typical search costs (average number of nodes expanded):

- *d=12* IDS = 3,644,035 nodes
  A*(h_1) = 227 nodes
  A*(h_2) = 73 nodes
- *d=24* IDS = too many nodes
  A*(h_1) = 39,135 nodes
  A*(h_2) = 1,641 nodes

Relaxed problems

- A problem with fewer restrictions on the actions is called a relaxed problem
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution

Local search algorithms

- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution

- State space = set of "complete" configurations
- Find configuration satisfying constraints, e.g., n-queens

- In such cases, we can use local search algorithms
- keep a single "current" state, try to improve it

Example: *n*-queens

- Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal

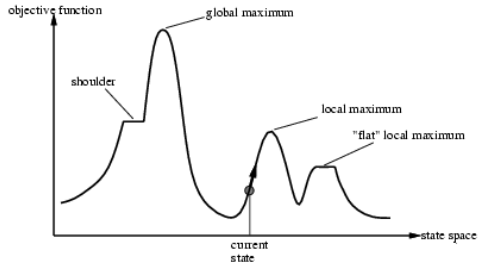
Hill-climbing search

- "Absent-minded blind man climbs a hill"
- Will he reach the highest peak?

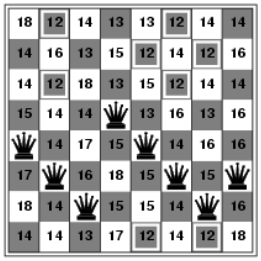
```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  inputs: problem, a problem
  local variables: current, a node
                   neighbor, a node
  current ← MAKE-NODE(INITIAL-STATE[problem])
  loop do
    neighbor ← a highest-valued successor of current
    if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
    current ← neighbor
```

Hill-climbing search

- Problem: depending on initial state, can get stuck in local maxima

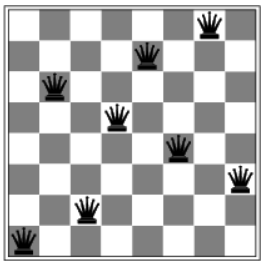


Hill-climbing search: 8-queens problem



- *h* = number of pairs of queens that are attacking each other, either directly or indirectly
- *h* = *17* for the above state

Hill-climbing search: 8-queens problem



- A local minimum with *h* = *1*

Simulated annealing search

- Idea: escape local maxima by allowing some "bad" moves but gradually decrease their frequency

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"
  local variables: current, a node
                   next, a node
                   T, a "temperature" controlling prob. of downward steps
  current ← MAKE-NODE(INITIAL-STATE[problem])
  for t ← 1 to ∞ do
      T ← schedule[t]
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← VALUE[next] – VALUE[current]
      if ΔE > 0 then current ← next
      else current ← next only with probability e^(Δ E/T)
```

Properties of simulated annealing search

- One can prove: If *T* decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1
- Widely used in VLSI layout, airline scheduling, etc

Local beam search

- Keep track of *k* states rather than just one
- Start with *k* randomly generated states
- At each iteration, all the successors of all *k* states are generated
- If any one is a goal state, stop; else select the *k* best successors from the complete list and repeat.

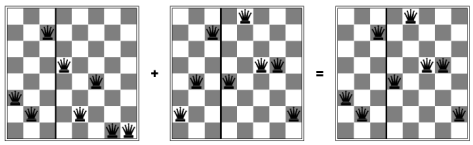
Genetic algorithms

- A successor state is generated by combining two parent states
- Start with *k* randomly generated states (population)
- A state is represented as a string over a finite alphabet (often a string of 0s and 1s)
- Evaluation function (fitness function). Higher values for better states.
- Produce the next generation of states by selection, crossover, and mutation

Genetic algorithms

- Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28)
- 24/(24+23+20+11) = 31%
- 23/(24+23+20+11) = 29% etc

Genetic algorithms



Adversarial Search

From AIMA Slides

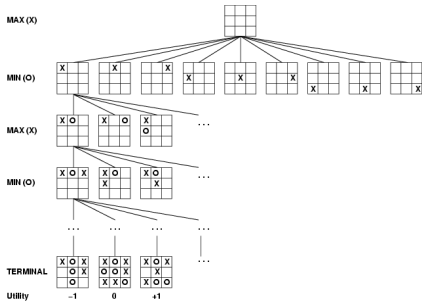
Outline

- Optimal decisions
- α-β pruning
- Imperfect, real-time decisions

Games vs. search problems

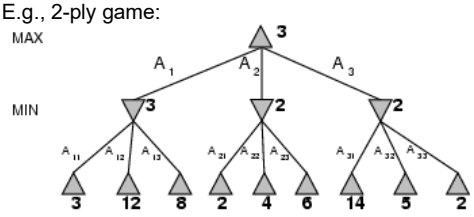
- "Unpredictable" opponent → specifying a move for every possible opponent reply
- Time limits → unlikely to find goal, must approximate

Game tree (2-player, deterministic, turns)



Minimax

- Perfect play for deterministic games
- Idea: choose move to position with highest minimax value
  = best achievable payoff against best play
- E.g., 2-ply game:



Minimax algorithm

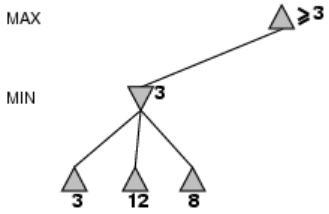
```
function MINIMAX-DECISION(state) returns an action
  v ← MAX-VALUE(state)
  return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for a, s in SUCCESSORS(state) do
    v ← MAX(v, MIN-VALUE(s))
  return v

function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for a, s in SUCCESSORS(state) do
    v ← MIN(v, MAX-VALUE(s))
  return v
```

Properties of minimax

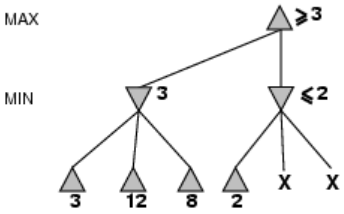
- Complete? Yes (if tree is finite)
- Optimal? Yes (against an optimal opponent)
- Time complexity? $O(b^m)$
- Space complexity? O(bm) (depth-first exploration)

- For chess, b ≈ 35, m ≈100 for "reasonable" games
  → exact solution completely infeasible

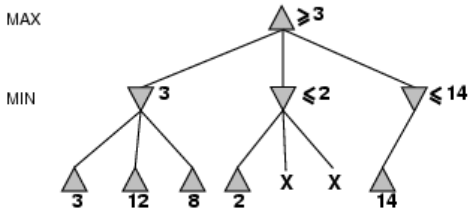
α-β pruning example



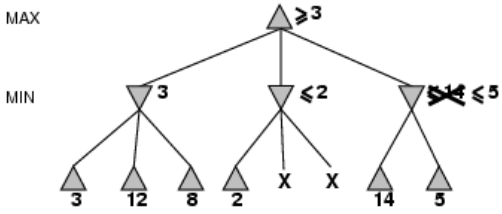
α-β pruning example



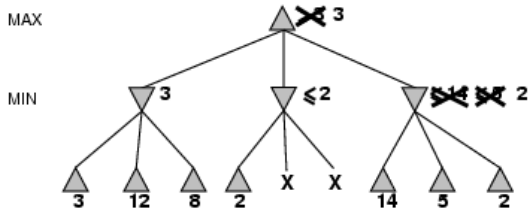
α-β pruning example



α-β pruning example



α-β pruning example

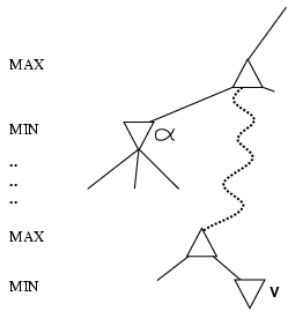


Properties of α-β

- Pruning does not affect final result
- Good move ordering improves effectiveness of pruning
- With "perfect ordering," time complexity = $O(b^{m/2})$
  → doubles depth of search
- A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)

Why is it called α-β?

- α is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for *max*
- If *v* is worse than α, *max* will avoid it
  → prune that branch
- Define β similarly for *min*



The α-β algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
  inputs: state, current state in game
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  inputs: state, current state in game
          α, the value of the best alternative for MAX along the path to state
          β, the value of the best alternative for MIN along the path to state
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for a, s in SUCCESSORS(state) do
    v ← MAX(v, MIN-VALUE(s, α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v
```

The α-β algorithm

```
function MIN-VALUE(state, α, β) returns a utility value
  inputs: state, current state in game
          α, the value of the best alternative for MAX along the path to state
          β, the value of the best alternative for MIN along the path to state
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for a, s in SUCCESSORS(state) do
    v ← MIN(v, MAX-VALUE(s, α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Resource limits

Suppose we have 100 secs, explore 10^4 nodes/sec
→ 10^6 nodes per move

Standard approach:

- cutoff test:
  e.g., depth limit (perhaps add quiescence search)
- evaluation function
  = estimated desirability of position

Evaluation functions

- For chess, typically linear weighted sum of features
  $Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$
- e.g., $w_1 = 9$ with
  $f_1(s)$ = (number of white queens) – (number of black queens), etc.

Cutting off search

MinimaxCutoff is identical to *MinimaxValue* except

1. *Terminal?* is replaced by *Cutoff?*
2. *Utility* is replaced by *Eval*

Does it work in practice?

$b^m = 10^6$, b=35 → m=4

4-ply lookahead is a hopeless chess player!

- 4-ply ≈ human novice
- 8-ply ≈ typical PC, human master
- 12-ply ≈ Deep Blue, Kasparov

Deterministic games in practice

- Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used a precomputed endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 444 billion positions.
- Chess: Deep Blue defeated human world champion Garry Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.
- Othello: human champions refuse to compete against computers, who are too good.
- Go: human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.

Summary

- Games are fun to work on!
- They illustrate several important points about AI
- perfection is unattainable → must approximate
- good idea to think about what to think about

Machine, Data and Learning

Utility Theory + Decision Theory
Chapter 17 from the book by Russell and Norvig

Decision Theory

(How to make decisions)

Decision Theory

= Probability theory + Utility Theory

(deals with chance) *(deals with outcomes)*

- *Fundamental idea:*
  - The **MEU** (Maximum expected utility) principle
  - Agent is **rational** if and only if it chooses the action that yields the highest expected utility, averaged over all possible outcomes of the action
  - Weigh the utility of each outcome by the probability that it occurs

Revisiting Romania example

- If plan1 and plan2 are the two plans:
  - Plan 1 uses route 1
    - P(home-early|plan1) = .8, while P(stuck1|plan1) = .2
    - Route 1 will be quick if flowing, but stuck for 1 hour if slow
    - ➢ U(home-early) = 100, U(stuck1) = -1000
    - ➢Assigned numerical values to outcomes!
  - Plan 2 uses route 2
    - P(home-somewhat-early|plan2) = .7, P(stuck2|plan2) = .3
    - Route 2 will be somewhat quick if flowing, but not bad even if slow
    - ➢U(home-somewhat-early) = 50, U(stuck2) = -10

Application of MEU Principle

- EU(Plan1) = P(home-early | plan1) *U(home-early)
  + P(stuck1| plan1) * U(stuck1)
  = *0.8 * 100 + 0.2 * -1000 = -120*

- EU(Plan2) = P(home-somewhat-early | plan2) *U(home-somewhat-early)
  + P(stuck2 | plan2) * U(stuck2)
  *= 0.7 * 50 + 0.3 * -10 = 32*

EU (plan2) is higher, so choose plan2

Lottery Example

- Suppose an agent gives you a choice:
  - **Choice 1:** You will get $1,000,000
  - **Choice 2:** The agent will toss a coin
    - If heads, then you win $3,000,000
    - If tails, then you get nothing

- Simple expected utility calculations give:
  - EU(Choice1) = $1,000,000
  - EU(Choice2) = $1,500,000

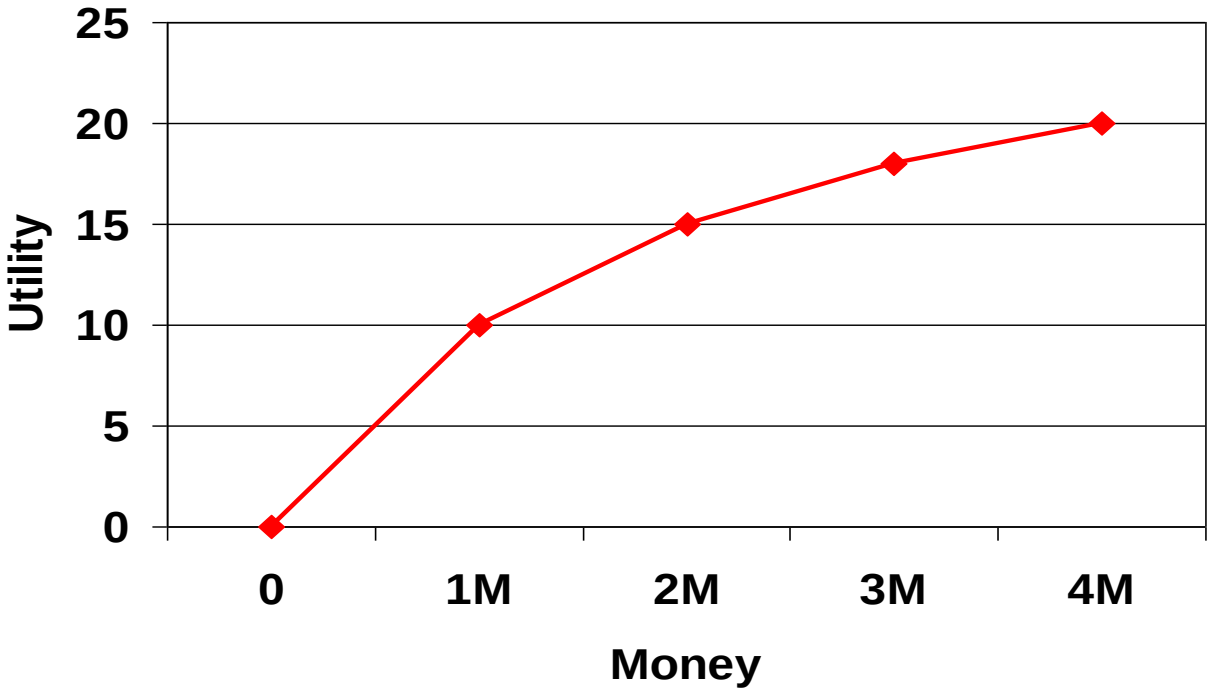
- So why did we prefer the first choice?

Risk Aversion

- We are **risk averse**
- Our utility functions for money are as follows (!!):
  - Our first million means a lot U($1M) = 10
  - Second million not so much U($2M) = 15 (NOT 20)
  - Third million even less so U($3M) = 18 (NOT 30)
  - ….

- Additional money is not buying us as much utility
- If we plot amount of money on the x-axis and utility on the y-axis, we get a concave curve

Answer: Risk Aversion



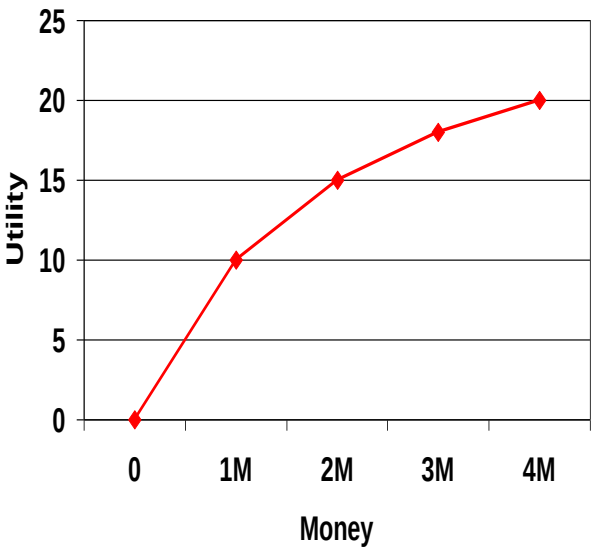
- EU(choice1) = U($1M) = 10
- EU(choice2) = 0.5*U(0) + 0.5*U($3M =18) = 9
- That is why we prefer the sure $1M

More Risk Aversion

- Key: Slope of utility function is **continuously decreasing**
  - We will refuse to play a monetarily fair bet
- Suppose we start with x dollars
  - We are offered a game:
    - 0.5 chance to win 1000 dollars (c = 1000)
    - 0.5 chance to lose 1000 dollars (c = 1000)
    - Expected monetary gain or loss is zero (hence monetarily fair)
  - Should be neutral to it, but seems we are not! Why?
    - $U(x + c) - U(x) < U(x) - U(x - c)$
    - $U(x + c) + U(x - c) < 2\ U(x)$
    - $[U(x + c) + U(x - c) / 2] < U(x)$
    - EU (playing the game) < EU (not playing the game)

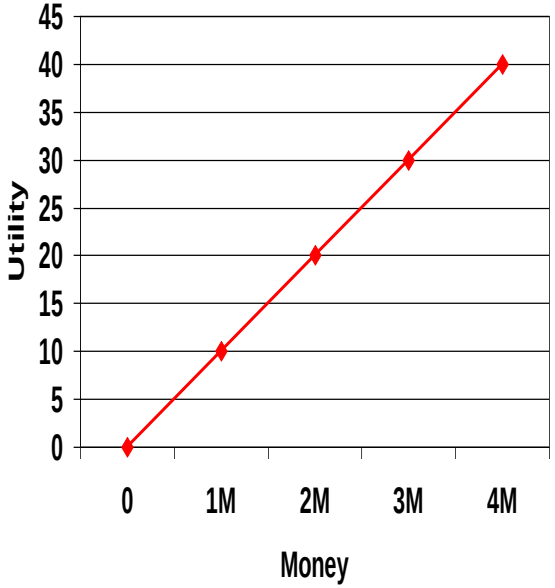
Risk Averse, Risk Neutral
Risk Seeking

RISK AVERSE



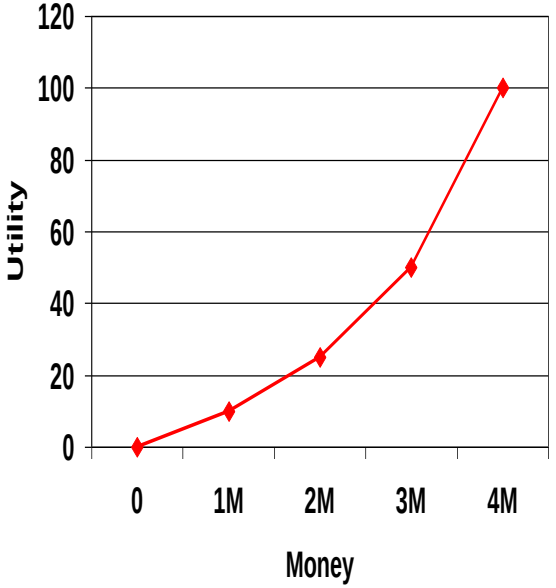
EU(Choice1) = 10
EU(Choice2) = 9

RISK NEUTRAL



EU(Choice1) =10
EU(Choice2) =15

RISK SEEKER



EU(Choice1)=10
EU(Choice2)=25

Multiattribute Utility Theory

- Can we capture desirability of outcomes in a single utility function?
- Suppose renting an apartment
  - House1: closer-to-university, newer, costs 100 units
  - House2: Farther-from-university, older, costs 85 units
  - (Assume, you can afford up to 100 units)

- Outcomes characterized by two or more attributes
  - Attributes: X1, X2, …XN, e.g., <distance-to-univ, old/new, cost>
  - Values: x1,x2…xN,
    - Closer-to-univ = 1, farther-from-univ = 0; new = 1, old = 0
  - Apartment1: <1,1,-100> Apartment2: <0,0, -85>
  - Which is a better apartment? (Pairwise comparison fails)

Multiattribute Utility Theory

- Don't get a single number, but vector of values as outcomes, <x,y>

- How do you compare values now?
  - Compare <1,1,-100> with <0,0, -85>
  - Compare <3,3,5> with <5,3,3>

- One approach is dominance (strict, stochastic…):
  - If you are lucky, find <3,3,3> and <3,3,5>
  - Values in one vector dominate values in the other vector

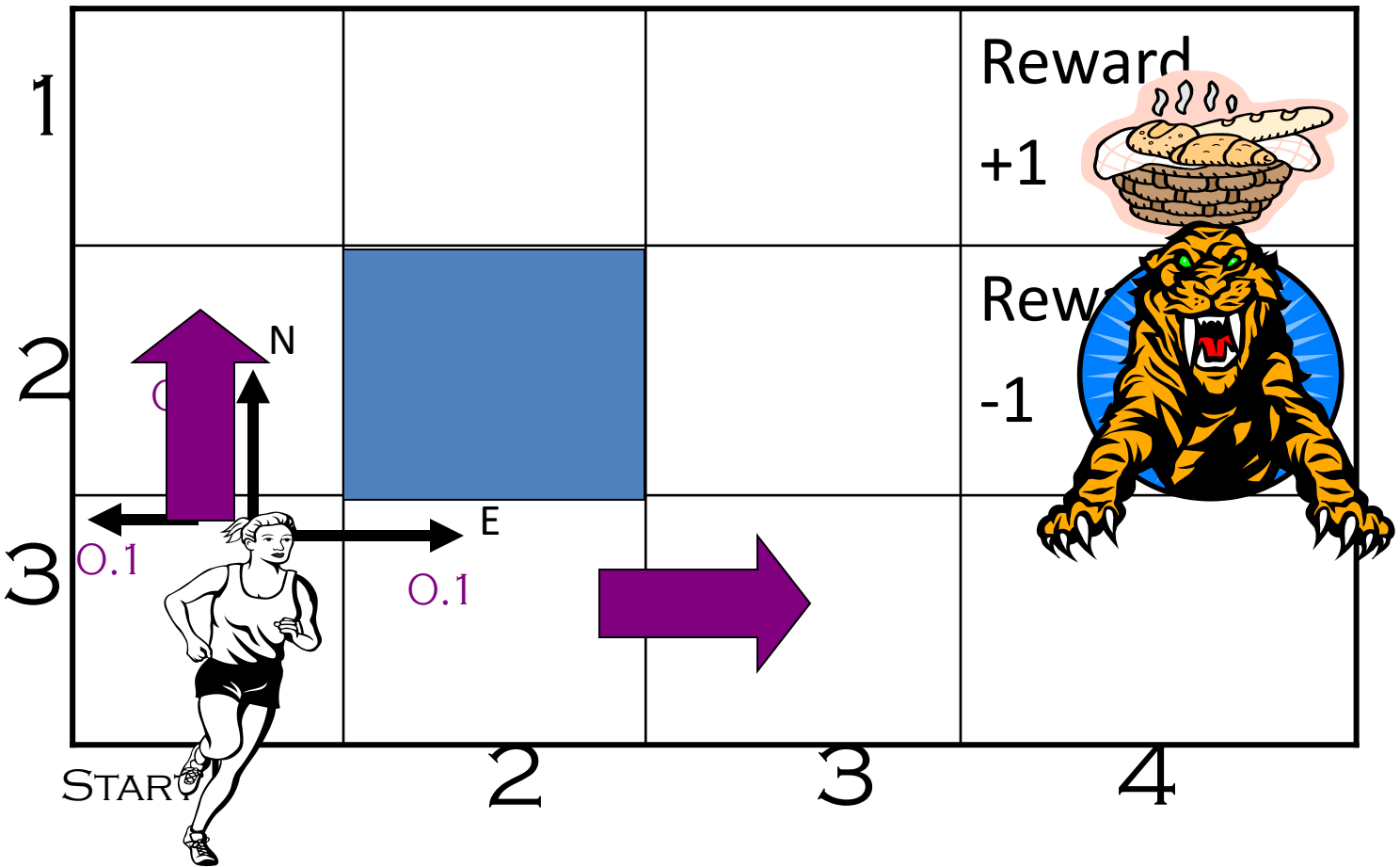
Markov Decision Process (MDP)
Chapter 17: Making Complex Decisions

- Defined as a tuple: <S, A, P, R>
  - **S:** State
  - **A:** Action
  - **P:** Transition function
    - Table P(s' | s, a), prob of s' given action "a" in state "s"
  - **R:** Reward
    - R(s, a) = cost or reward of taking action a in state s

- Choose a sequence of actions (not just one action)
  - Utility based on a sequence of actions
  - Model **Sequential Decision Problems**

Example: What SEQUENCE of actions should our agent take?

- Agent can take action N, E, S, W
- Each action costs –1/25

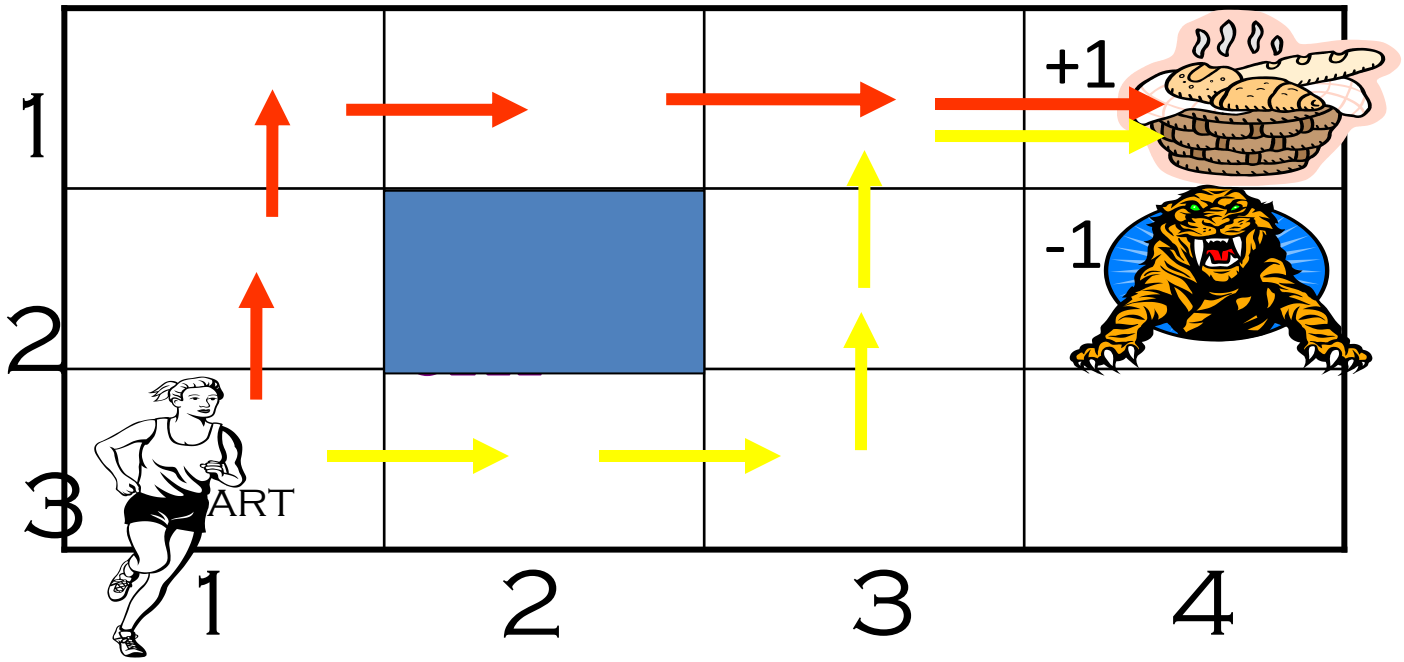


MDP Tuple: <S, A, P, R>

- **S:** State of the agent on the grid
  - Ex: state (4,3) ([column, row] notation used)
- **A:** Actions of the agent, i.e., N, E, S, W
- **P:** Transition function
  - Table P(s' | s, a), prob of s' given action "a" in state "s"
  - E.g., P((4,3) | (3,3), N) = 0.1
  - E.g., P((3, 2) | (3,3), N) = 0.8
  - (Robot movement, uncertainty of another agent's actions,…)
- **R:** Reward
  - R((3, 3), N) = -1/25
  - R (4,1) = +1

How Would you Solve this Problem?

- Simple search algorithm? **Not deterministic**
- Apply MEU to an entire sequence of actions?
  - *Create multiple plans, e.g., Red plan vs. Yellow plan below*
  - *Choose a plan that leads to MEU*



How Would you Solve this Problem?

- Apply **MEU** to an entire sequence of actions?
- Does not work because uncertainty at every step
  - E.g., After first step of Red plan, may move east not North!
  - No action specified there (I.e., in cell (2,1))
- Solution is a *Policy*
  - Complete mapping from states to actions

MDP Basics and Terminology

- **Markov Assumption:** Transition probabilities (and rewards) from any given state depend only on the state and not on previous history

- An agent must make a decision or control a probabilistic system
  - Goal is to choose a sequence of actions for optimality
  - **Decision Epoch:** Points at which decisions are made
    - **Finite horizon MDPs**: # of decision epochs is finite i.e. fixed time after which game ends : Time dependent policy
    - **Infinite horizon MDPs:** # of decision epochs is infinite i.e. Time independent policy
  - **Transition model:** Table of probabilities P
    - In our example, 0.8, 0.1, 0.1 transition probabilities
    - P(J | S, A) : Probability of state J, given action A in State S
  - **Absorbing state:** Goal state

Reward Function

- Reward is assumed associated with state, action i.e. **R(S, A)**
  - If all actions have the same reward can use R(S)
  - We could also assume a mix of R(S,A) and R(S)
  - Will use R(S,A) as the notation

- Sometimes, reward associated with state, action, destination-state
  - R(S,A,J)
  - $R(S,A) = \sum_{J} R(S,A,J) * P(J \mid S, A)$

MDP Policy

- **Decision Rule:** Procedure to choose action in each state for ***a given decision epoch***
  - E.g., MDP has states, S1 and S2, with actions A1, A2 in both states
  - Decision rules Di for each decision epoch "i" as shown in table below
  - Four decision rules shown, D1, D2, D3, D4, one for each epoch
  - Numbers in (..) are probabilities, e.g., 0.7, 0.3, 1.0
- **Policy:** Decision rule to be used at all decision epochs
  - Policy = {D1, D2, D3,D4} (assuming finite horizon T = 4)

D1	D2	D3	D4…
S1→A1 (0.7) →A2 (0.3) S2→A2 (1.0)	S1→A1 (1.0) S2→A1 (0.3) →A2 (0.7)	S1→A2 (1.0) S2→A2 (1.0)	….

Stationary and Deterministic Policies

- **Stationary** policy implies same decision rule in every epoch
  - **Stationary policy:** {D, D, D, D…}
  - **Non-stationary policy** changes with time (e.g., D1,D2, D3…Dn)
- **Deterministic policy** implies choosing an action with certainty
  - **Deterministic policy:** Si → Ai (probability 1.0)
  - **Randomized policy:** Probability distribution on the set of actions
- What type of a policy is the following?

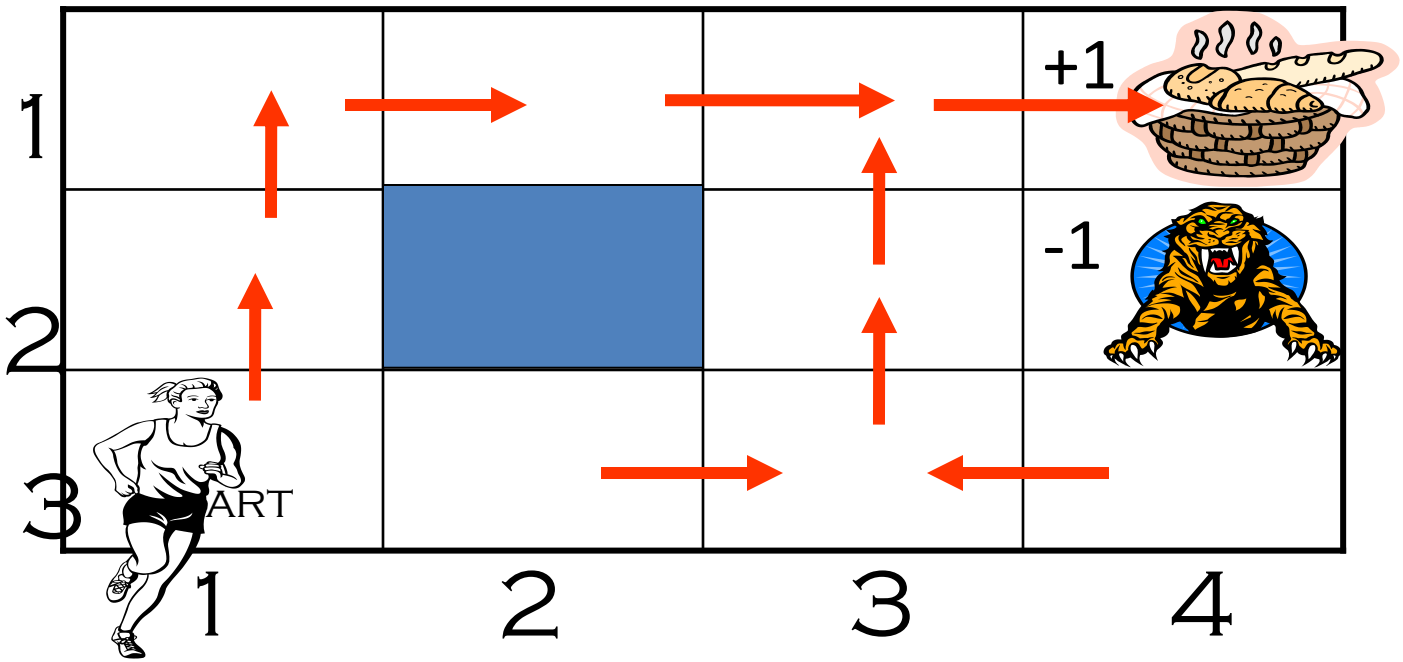
D	D	D	D
S1→A1 (1.0) S2→A2 (1.0)	S1→A1 (1.0) S2→A2 (1.0)	S1→A1 (1.0) S2→A2 (1.0)	S1→ A1(1.0) S2→ A2(1.0)

Stationary and Deterministic Policies

- **Optimal** MDP policy for infinite horizon is Stationary & Deterministic policies (aka pure policy)
- Policy denoted by symbol π
- Stationary & deterministic policies denoted π^{SD}
- Is a policy π^{SR} possible? (SR = Stationary & randomized)

- **Note:**
  - **When nothing is specified regarding time horizon, assume infinite horizon**
  - **When asked to find the policy at time horizon = 4, it means find the decision rule D4. It can also be stated as find decision rule for T = 4 or D4.**
  - **When asked to find policy for a time horizon of 4, means find all decision rules D1, D2, D3 and D4**

Pure Policies: π^{SD}



- Deterministic, non-changing mapping from states to actions

  $\pi((1,3)) \rightarrow$ North (non-changing, non-random)

  $\pi((1,2)) \rightarrow$ North

  $\pi((4,3)) \rightarrow$ West……

Policy

- **Policy** is like a plan, but not quite
  - Certainly, generated ahead of time, like a plan

- Unlike traditional plans, it is not a sequence of actions that an agent must execute
  - If there are failures in execution, agent can continue to execute a policy

- Prescribes an action for all the states

- Maximizes expected reward, rather than just reaching a goal state

Value Iteration: Algorithm

- *Basic algorithm is very simple!*
- *Initialize:* $U_0(I) = 0$
- Iterate:

$$U_{t+1}(I) = \max_{A} \left[R(I,A) + \sum_{J} P(J|I,A) * U_t(J) \right]$$

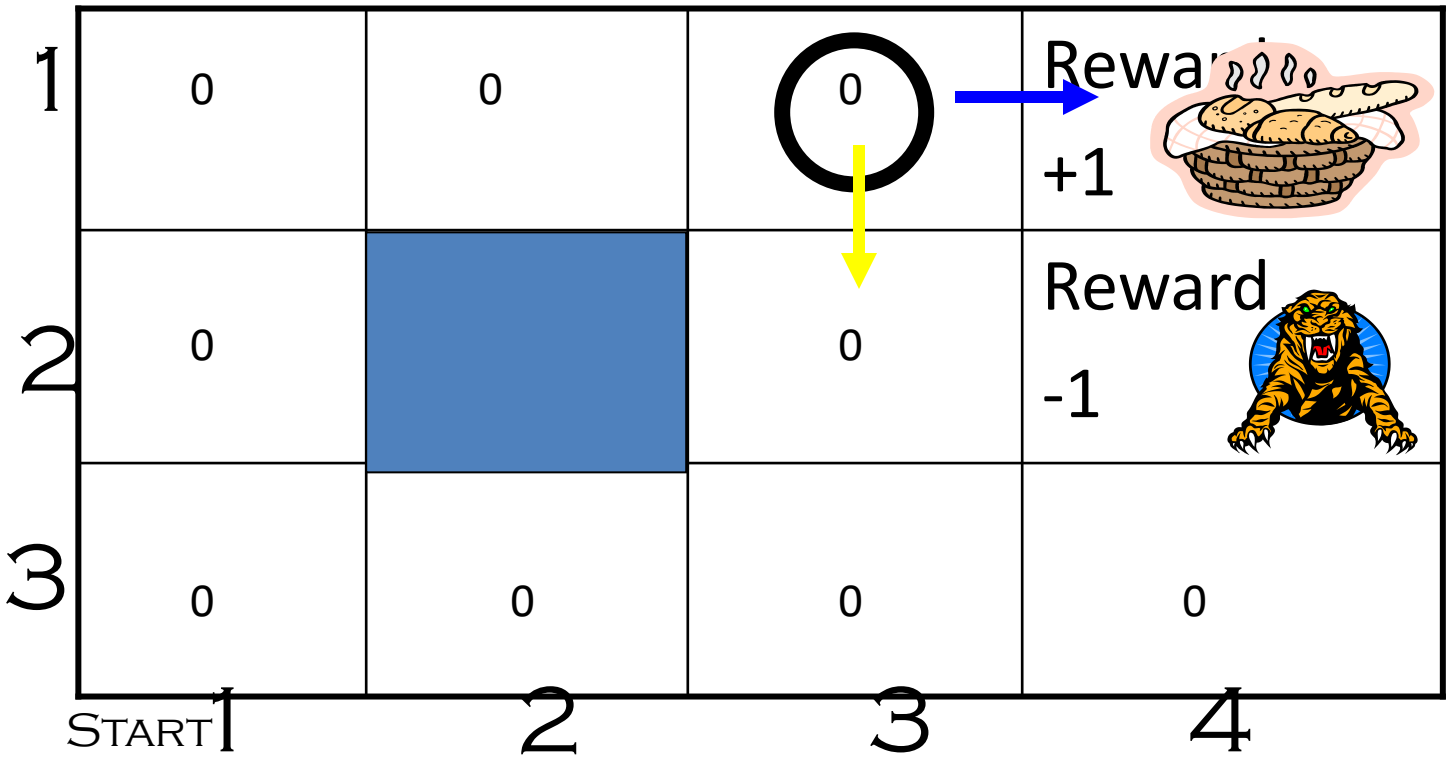
–Until close-enough (U_{t+1}, U_t)

Dr. Richard Bellman

- *Iteration step called "Bellman update"*
- *Inventor of dynamic programming (1957)*

Iteration #1: Cell (3,1)

- East: -1/25 + [0.8 * 1 + 0.1 * 0 + 0.1 * 0] = 0.76
- North/South: -1/25 + [0.8 * 0 + 0.1 * 1 + 0.1 * 0] = 0.06
- West: ??
- So, State (3,1) has value of 0.76



Markov Chain

Discounting

Value Iteration: Modify

- *Initialize:* $U_0(I) = 0$
- Iterate:

$$U_{t+1}(I) = \max_{A} \left[R(I,A) + \gamma \sum_{J} P(J|I,A) * U_t(J) \right]$$

  - Until close-enough (U_{t+1}, U_t)

- At the end of iteration, calculate optimal policy:

$$Policy(I) = \operatorname*{argmax}_{A} \left[R(I,A) + \gamma \sum_{J} P(J|I,A) * U_{t+1}(J) \right]$$

Machine, Data and Learning

Linear Programming

- Mathematical programming is used to find the **best or optimal solution** to a problem that requires a decision or set of decisions about how best to use a set of limited resources to achieve a state goal of objectives.
- **Steps involved in mathematical programming**
  - Conversion of stated problem into a mathematical model that abstracts all the essential elements of the problem.
  - Exploration of different solutions of the problem.
  - Finding out the most suitable or optimum solution.
- **Linear programming** requires that all the mathematical functions in the model be **linear functions**.

The Linear Programming Model (1)

Let: $X_1, X_2, X_3, \ldots\ldots\ldots, X_n$ = decision variables

Z = Objective function or linear function

<u>Requirement:</u> Maximization of the linear function Z.

$$Z = c_1X_1 + c_2X_2 + c_3X_3 + \ldots\ldots\ldots + c_nX_n \qquad \ldots\ldots\text{Eq (1)}$$

subject to the following <u>constraints</u>:

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &\leq b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &\leq b_2 \\
&\vdots \\
a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &\leq b_n \\
\text{all } x_j &\geq 0
\end{aligned}$$

The Linear Programming Model (2)

- The linear programming model can be written in more efficient notation as:

Maximize

$$Z = \sum_{j=1}^{n} c_j x_j$$

subject to:

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i$$

where

$$i = 1, 2, \ldots, m$$

and

$$x_j \geq 0$$

where

$$j = 1, 2, \ldots, n$$

The decision variables, x_1, x_2, ..., x_n, represent levels of n competing activities.

Examples of LP Problems (1)

1. A Product Mix Problem

- A manufacturer has fixed amounts of different resources such as raw material, labor, and equipment.
- These resources can be combined to produce any one of several different products.
- The quantity of the i^{th} resource required to produce one unit of the j^{th} product is known.
- The decision maker wishes to produce the combination of products that will **maximize total income.**

Examples of LP Problems (2)

2. A Transportation Problem

- A product is to be shipped in the amounts $a_1, a_2, \ldots, a_m$ from m shipping origins and received in amounts $b_1, b_2, \ldots, b_n$ at each of n shipping destinations.

- The cost of shipping a unit from the i^{th} origin to the j^{th} destination is known for all combinations of origins and destinations.

- The problem is to determine the amount to be shipped from each origin to each destination such that the total **cost of transportation is a minimum.**

Examples of LP Problems (3)

3. A Flow Capacity Problem

- One or more commodities (e.g., traffic, water, information, cash, etc.) are flowing from one point to another through a network whose branches have various constraints and flow capacities.
- The direction of flow in each branch and the capacity of each branch are known.
- The problem is to determine the **maximum flow**, or capacity of the network.

Developing LP Model

- Consider the product mix problem

- **Steps Involved:**
  - Determine the objective of the problem and describe it by a criterion function in terms of the decision variables.
  - Find out the constraints.
  - Do the analysis which should lead to the selection of values for the decision variables that optimize the criterion function while satisfying all the constraints imposed on the problem.

Developing LP Model

XYZ Company produces two products: I and II. The raw material requirements, space needed for storage, production rates, and selling prices for these products are given in Table 1.

TABLE .1 Production Data for N. Dustrious Company

	Product	
	I	II
Storage space (ft^2/unit)	4	5
Raw material (lb/unit)	5	3
Production rate (units/hr)	60	30
Selling price ($/unit)	13	11

The total amount of raw material available per day for both products is 15751b. The total storage space for all products is 1500 ft^2, and a maximum of 7 hours per day can be used for production.

Developing LP Model

All products manufactured are shipped out of the storage area at the end of the day. Therefore, the two products must share the total raw material, storage space, and production time. **The company wants to determine how many units of each product to produce per day to maximize its total income.**

Solution

- The company has decided that it wants to maximize its sale income, which depends on the number of units of product I and II that it produces.
- Therefore, the decision variables, x_1 and x_2 can be the number of units of products I and II, respectively, produced per day.

Developing LP Model

- The object is to maximize the equation:

$$Z = 13x_1 + 11x_2$$

subject to the constraints on storage space, raw materials, and production time.

- Each unit of product I requires 4 ft^2 of storage space and each unit of product II requires 5 ft^2. Thus a total of $4x_1 + 5x_2$ ft^2 of storage space is needed each day. This space must be less than or equal to the available storage space, which is 1500 ft^2. Therefore,

$$4X_1 + 5X_2 \leq 1500$$

- Similarly, each unit of product I and II produced requires 5 and 3 lbs, respectively, of raw material. Hence a total of $5x_I + 3x_2$ lb of raw material is used.

Developing LP Model

- This must be less than or equal to the total amount of raw material available, which is 1575 Ib. Therefore,

$$5x_1 + 3x_2 \leq 1575$$

- Product I can be produced at the rate of 60 units per hour. Therefore, it must take 1/60 of an hour to produce 1 unit. Similarly, it requires 1/30 of an hour to produce 1 unit of product II. Hence a total of $x_1/60 + x_2/30$ hours is required for the daily production. This quantity must be less than or equal to the total production time available each day. Therefore,

$$x_1 / 60 + x_2 / 30 \leq 7 \text{ or } x_1 + 2x_2 \leq 420$$

- Finally, the company cannot produce a negative quantity of any product, therefore x_1 and x_2 must each be greater than or equal to zero.

Developing LP Model

- **The linear programming model for this example can be summarized as:**

Maximize

$$Z = 13x_1 + 11x_2$$

subject to:

$$4x_1 + 5x_2 \leq 1500$$
$$5x_1 + 3x_2 \leq 1575$$
$$x_1 + 2x_2 \leq 420$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

Linear Programming for MDPs

- Maximize

$\sum_{i \in S} V_i$ (S is the set of states, Vi is value of state)

Such that:

$$V_i <= [\ R(I,A) + \gamma \sum P(J|I,A) * V_j]$$

Linear Programming

- Linear programming polynomial time (Karmarkar, 84)
  - Popular method Dantzig's simplex (1963)
  - Simplex performs well in practice, but could be exponential

- Although polynomial time typically slower than MDP specific algorithms such as value iteration

- Determine policy from V_i using method from previous slides
  - Yields a deterministic policy for MDPs: why?

- Key advantage: Many times it is amenable to modeling specific constraints

A more popular formulation

$$\max \sum_i \sum_a x_{ia} r_{ia}$$

$$\sum_a x_{ja} - \sum_i \sum_a x_{ia} p_{ij}^a = \alpha_j,$$

$$x_{ia} \geq 0$$

x_{ia}: Expected number of times action a is taken in state i
r_{ia}: Reward for taking action a in state i
p^a_{ij}: Probability of reaching state j when action a is taken in state I
α_{j}: Initial probability of being in state j

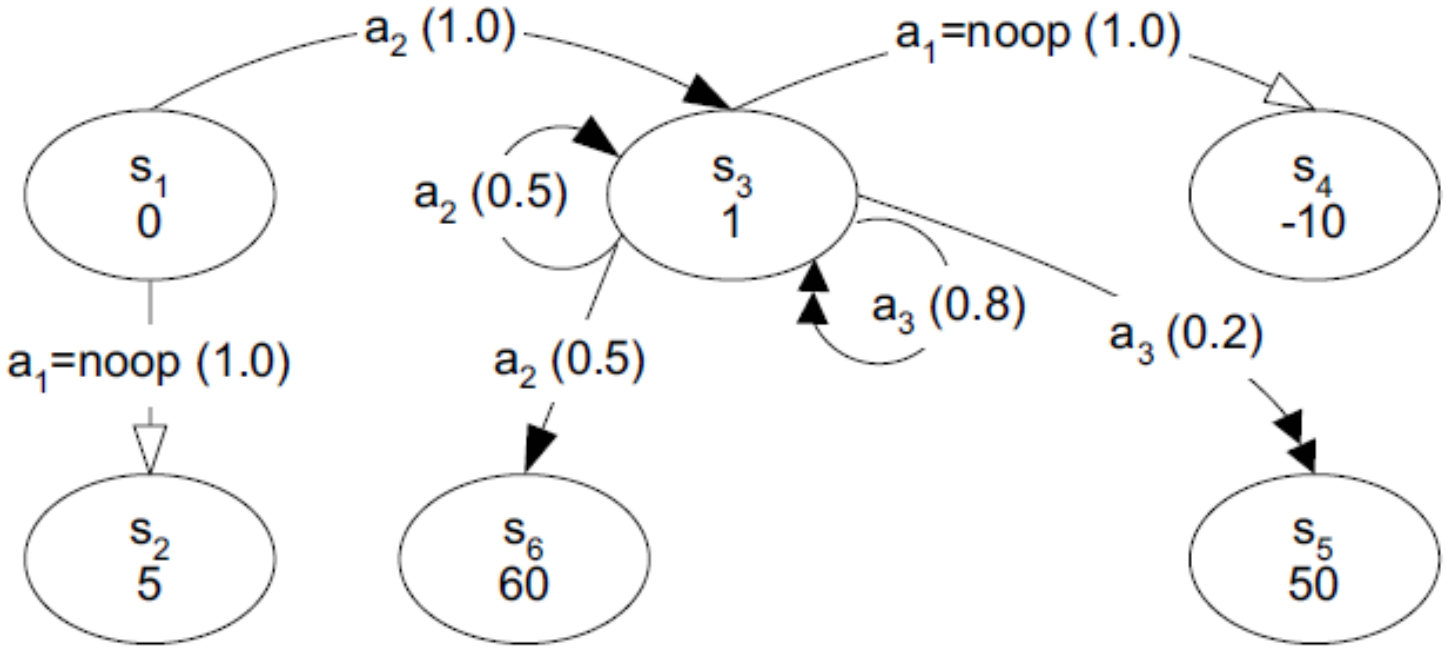
Refactoring

$$\max \sum_i \sum_a x_{ia} r_{ia} \quad \Bigg| \quad \sum_i \sum_a (\delta_{ij} - p_{ij}^a) x_{ia} = \alpha_j,$$

$$x_{ia} \geq 0,$$

where δ_{ij} is the Kronecker delta, defined as $\delta_{ij} = 1 \iff i = j$.

MDP Example



A simple MDP: $S = \{s_1, \ldots, s_6\}, A = \{a_1 = noop, a_2, a_3\}$

MDP Example

$$\max(\mathbf{rx}) \mid \mathbf{Ax} = \boldsymbol{\alpha},\ \mathbf{x} \geq 0,$$

$$\mathbf{x} = [(x_{11}, x_{12}), x_{21}, (x_{31}, x_{32}, x_{33}), x_{41}, x_{51}, x_{61}]^T,$$

$$\mathbf{r} = [(0,0), 5, (1,1,1), -10, 50, 60], \quad \boldsymbol{\alpha} = [0.1, 0.1, 0.1, 0.1, 0.1, 0.5]^T,$$

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0.5 & 0.2 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -0.2 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -0.5 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Solution to the LP

$$\mathbf{x} = [(0, 0.1), 0.1, (0, 0.4, 0), 0.1, 0.1, 0.7]$$
$$\mathbf{d} = [a_2, a_1, a_2, a_1, a_1, a_1]$$

Solving the LP we get x, which maps to a deterministic uniformly-optimal policy

We get the following output if we change alpha to the following

$$\boldsymbol{\alpha} = [1, 0, 0, 0, 0, 0]$$
$$\mathbf{x} = [(0, 1), 0, (0, 2, 0), 0, 0, 1]$$

$$d_1 = a_2, d_3 = a_2, d_6 = a_1$$

arbitrary actions for s_2, s_4, and s_5

CMDP

- A key advantage of the formulation is the ability to model constraints
- If we want to say state 1 must have all actions take with equal probability
  - x11 = x12
- Addition of such constraints results in a CMDP (Constrained MDP)
- How do you model action 1 in state 1 must be taken 30% of time ?
  - X11/(x11+x12) = 1/3
  - How can you do this with value or policy iteration ?

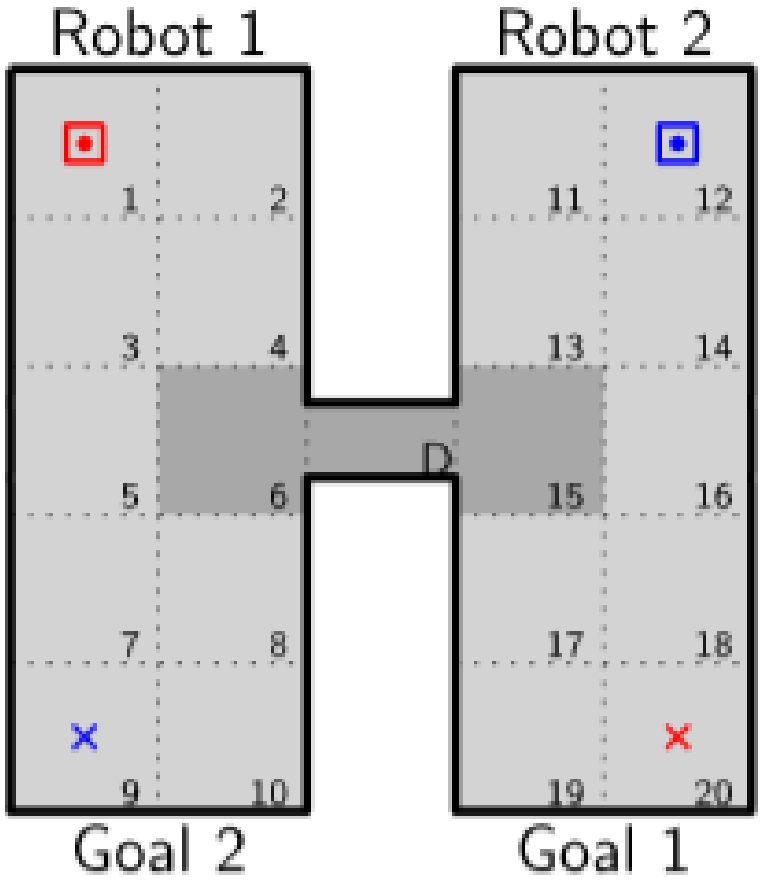
Multi-Agent MDP (MMDP)

- **MMDP:** A direct extension over MDPs for multiple agents
- $M = \langle S, \{A_i\}_{i \in m}, T, R \rangle$
  - **S** is set of possible world states
  - $\mathbf{\{A_i\}_{i \in m}}$ is set of joint actions, $\langle a_1, \ldots, a_m \rangle$ where $a_i \in A_i$
  - **T** defines transition probabilities over joint actions
  - **R** is team reward function
- State is fully observable by each agent
- In absence of communication, random policies can lead to mis-coordination
- Ex: In state s, policy a1b2 = .7 and a2b1 = .3 would lead to a1b1 with .21, a1b2 with .49, a2b1 with .09 and a2b2 with .21

Dec-MDP

- In MMDP model agent observes the joint state
- Many times in a joint problem, an agent may observe only his/her local state
- Separating out the policy computation for each agent is not an option since they have joint transitions and rewards
- Dec-MDP – A formal framework to address these issues

Robots coordinating in a hallway with a narrow passage



Can be modeled using Dec-MDP framework