Introduction to Artificial Intelligence

LECTURE 1:

Introduction

Class Overview 1: Material

- Lectures
  - Slides
  - Textbook: Russell & Norvig's AIMA
- Application Problems
  - In class and at home
- Practicals
  - Mostly at home

- ⚠ You are expected to work outside class!

Class Overview 1: Grading

• Exam	~70%
• Practicals	~30%
• (Possibly Quizz(es))	

Class Overview 1: Prerequisites & Topics Covered

- Prerequisites
  - M2103 Bases de la Programmation Objet
  - M2201 Graphes et Langages
  - M3103 Algorithmique Avancée
  - M3201 Probabilités et Statistiques

- Topics Covered
  - **Search and Game Principles**
  - **Learning**
  - **Reasoning**

Lecture Overview

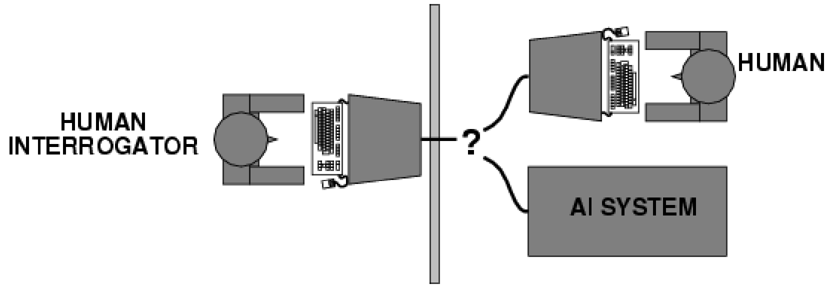
- Introduction to the study of AI
- Introduction to Search
  - Building Models
  - Search Problems
  - State and Search Spaces
  - Tic-Tac-Toe

Importance of studying AI

- Grand Challenges in Science and Technology
  - understanding the brain
    - reasoning, cognition, creativity
  - creating intelligent machines
    - is this possible?
    - what are the technical and philosophical challenges?
- What is involved in intelligence?
  - Ability to interact with the real world: perceive, understand, and act
    - speech, image understanding…
  - Reasoning
    - modeling the external world
    - solving new problems
    - ability to deal with unexpected problems, uncertainties
  - Learning and Adaptation
    - our internal ‘models’ are always being “updated”
    - e.g., a baby learning to categorize and recognize colors

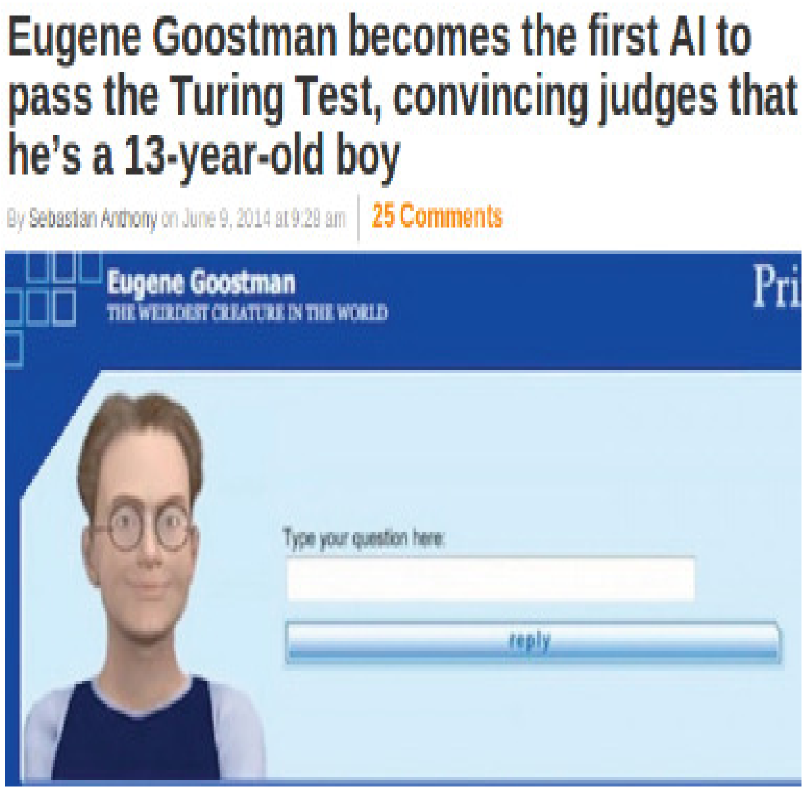
Can computers be ‘intelligent’?

- Turing test



- Tests the behaviour and not the internals of the computer
- However, suggests major components required for AI:
  - Language understanding
  - Reasoning
  - Learning…
- Question: is it important that an intelligent system act like a human?

Can computers be ‘intelligent’?



Eugene Goostman becomes the first AI to pass the Turing Test, convincing judges that he's a 13-year-old boy

By Sebastian Anthony on June 9, 2014 at 9:28 am | 25 Comments

H: Which is bigger, a shoebox or Mount Everest?

Eug: I can't make a choice right now. I should think it out later. And I forgot to ask you where you are from.

H: How many legs does a camel have?

Eug: Something between 2 and 4. Maybe, three? :-))) By the way, I still don't know your specialty - or, possibly, I've missed it?

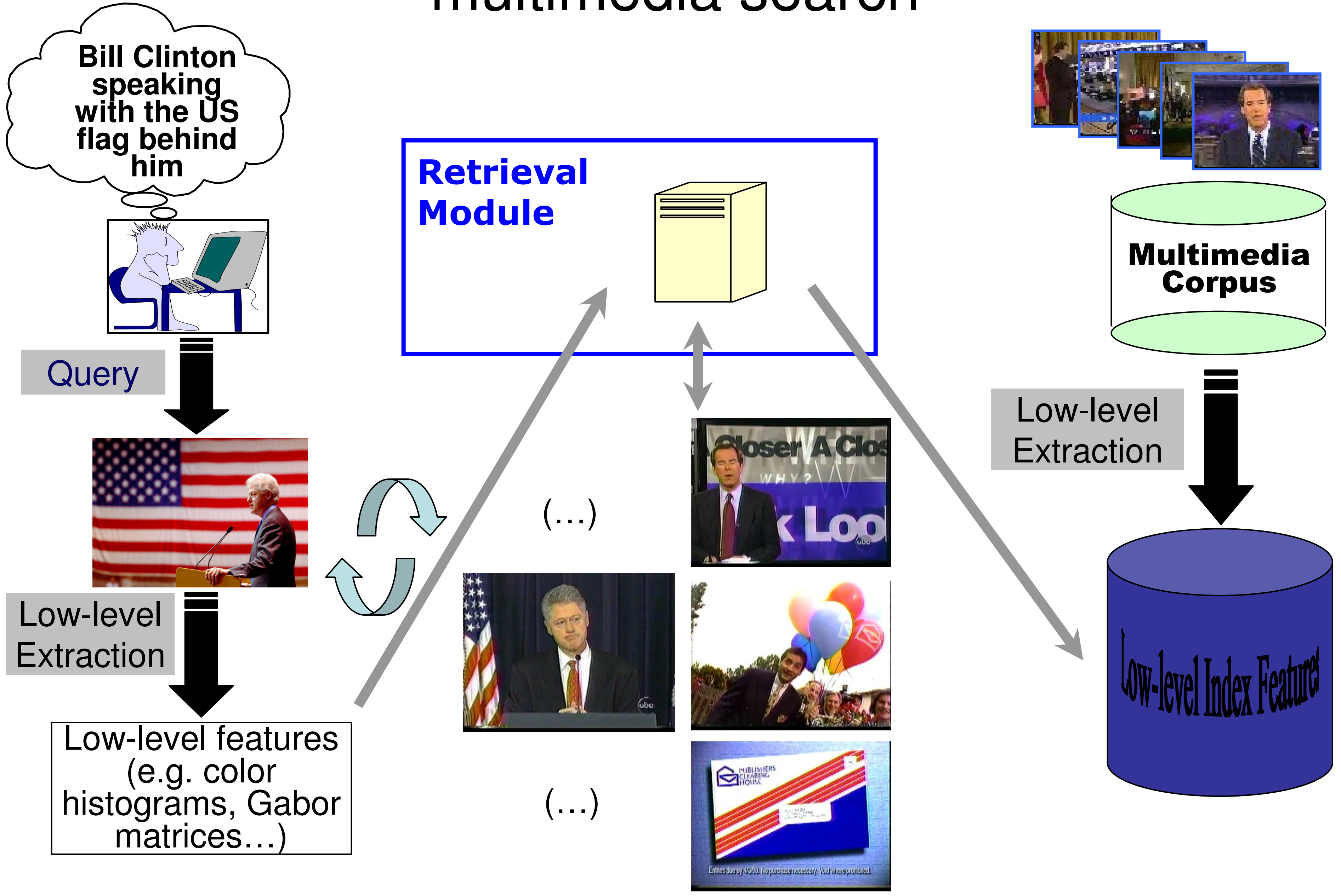
H: How many legs does a millipede have?

Eug: Just two, but Chernobyl mutants may have them up to five. I know you are supposed to trick me.

H: No, I need to know that you're not a chatbot. Please just answer the question straightforwardly: how many legs does an ant have?

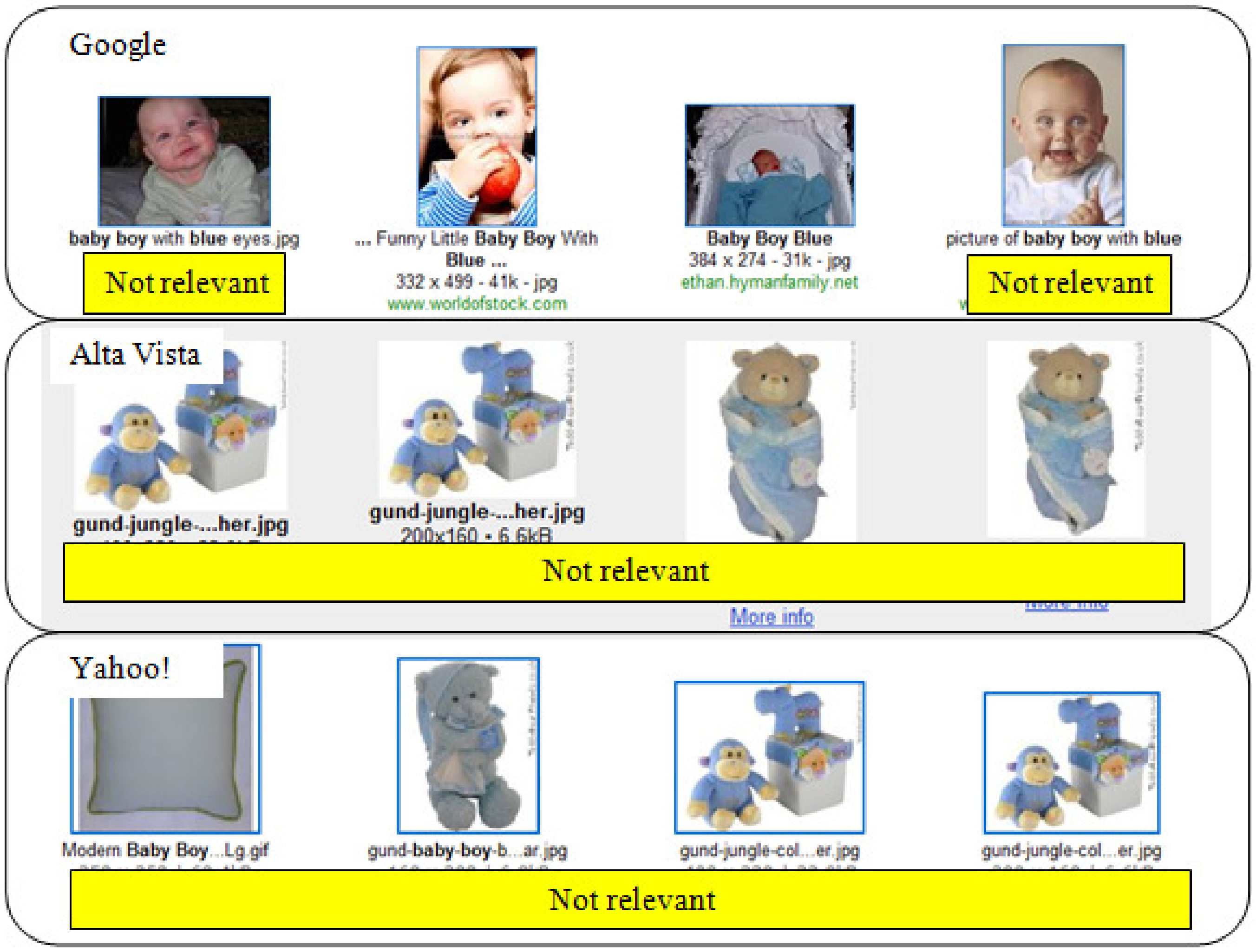
Eug: Something between 2 and 4. Maybe, three? :-))) Oh, what a fruitful conversation;-)

Can computers be 'intelligent'? The example of multimedia search


Bill Clinton speaking with the US flag behind him
Query
Low-level Extraction
Low-level features (e.g. color histograms, Gabor matrices…)
Retrieval Module
(…)
(…)
Multimedia Corpus
Low-level Extraction
Low-level Index Features

Can computers be ‘intelligent’? The example of multimedia search

SEARCH QUERY: Baby Boy in Blue



Building hardware as complex as the brain?

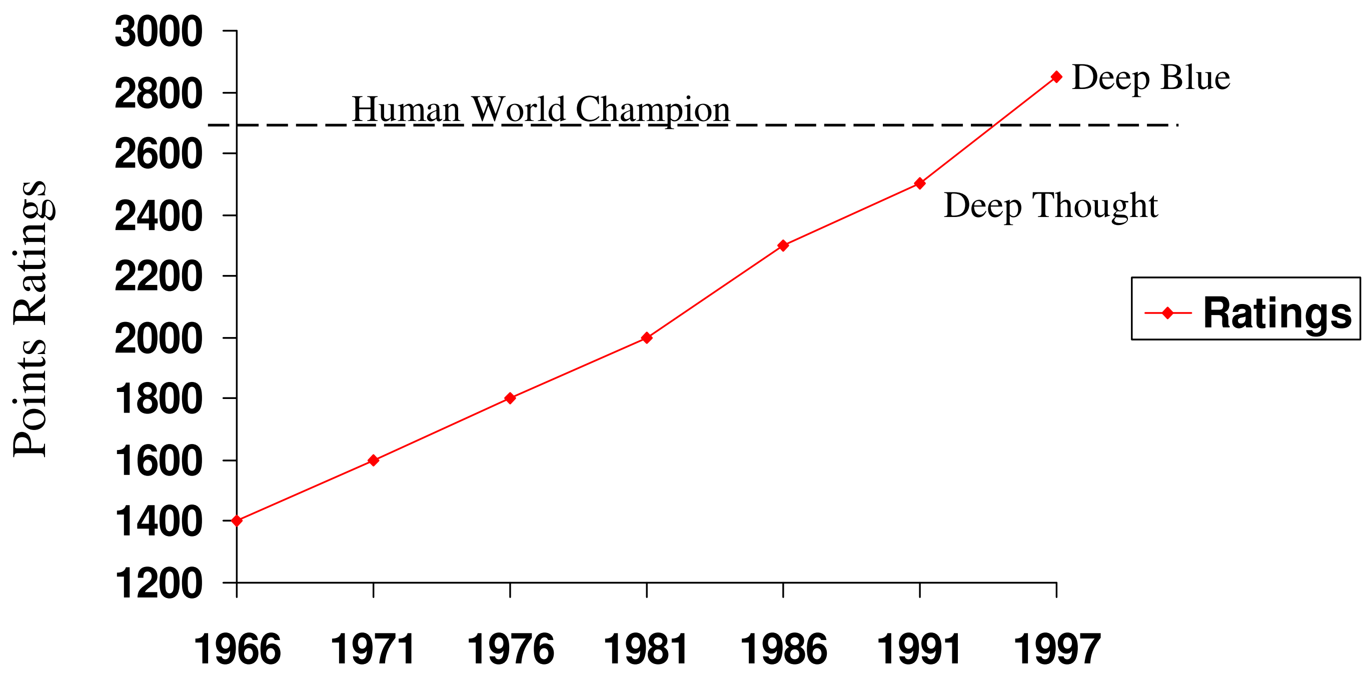
- How evolved is our brain?
  - a neuron, or nerve cell, is the basic information processing unit
  - estimated to be on the order of 10^{12} neurons in a human brain
  - many more synapses (10^{14}) connecting these neurons
  - cycle time: 10^{-3} seconds (1 millisecond)
- How complex can we make computers?
  - 10^8 or more transistors per CPU
  - supercomputer: hundreds of CPUs, 10^{12} bits of RAM
  - cycle time: order of 10^{-9} seconds
- We can have computers with as many basic processing elements as the brain, but with
    - far fewer interconnections (wires or synapses) than the brain
    - much faster updates than the brain
- Sophisticated hardware VS making a computer behave like a brain…

Surprises in AI research

- Tasks difficult for humans have turned out to be “easy”
  - Chess
  - Checkers, Othello, Backgammon
  - Logistics planning
  - Airline scheduling
  - Fraud detection
  - Sorting mail
  - Crossword puzzles…

Computers VS Humans at Chess

- Chess Playing is a classic AI problem
  - well-defined problem
  - very complex: difficult for humans to play well



Surprises in AI research

- Tasks easy for humans have turned out to be hard.
  - Speech synthesis/recognition
  - Face recognition
  - Composing music/art
  - Autonomous navigation
  - Motor activities (walking)
  - Textual, visual, audio understanding
  - Common sense reasoning (example: how many legs does a fish have?)

‘Can Computers Talk?’ aka Speech Synthesis

- Process
  - translate text to phonetic form
    - e.g., “fictitious” -> fik-tish-es
  - use pronunciation rules to map phonemes to actual sound
    - e.g., “tish” -> sequence of basic audio sounds
- Difficulties
  - Unnatural sounds
  - Sounds are not independent
    - e.g., “act” and “action”
  - How about ‘human matters’ such as emphasis, emotion, etc

Automated Speech Recognition

- Process
  - mapping sounds into a list of words for further processing
- State-of-the-art
  - Recognizing words from a limited vocabulary can be done with high accuracy (~ 99%)
  - e.g., directory inquiries
    - limited vocabulary (area codes, city names)
    - computer tries to recognize you first, if unsuccessful hands you over to a human operator
- Recognizing normal speech is more difficult
  - speech is continuous: where are the boundaries between words?
  - large vocabularies, multilingual issues
  - background noise, other speakers, accents, colds, etc
  - current systems are only about 60-70% accurate

Speech/Text Understanding

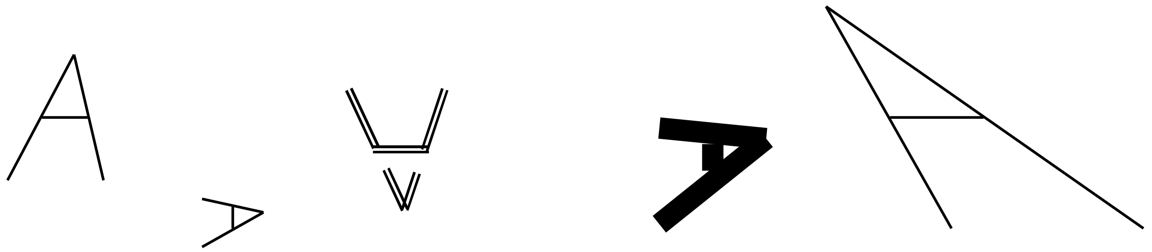
- “Time flies like an arrow”
  - assume the computer can recognize all the words
  - how many different interpretations are there?

Can Computers Learn and Adapt ?

- Learning and Adaptation
  - Example of autonomous cars and the DARPA Grand and Urban Challenges
    - Use of sensors
    - Synthesize the data
    - Localize the car and generate signals for steering, throttle, brake…
  - Towards self-driving cars (Google, Uber…)
- **Machine learning** allows computers to learn to do things without explicit programming

Can computers ‘see’?

- Recognition and Understanding of Objects in a scene
  - You can ‘effortlessly’ recognize objects around you or in an image
  - Human brain can map a bi-dimensional visual image to a 3D visual “map”
- Why is computerized visual recognition a hard problem?



- Possible for certain constrained problems

Can computers plan and make optimal decisions?

- Problems involving a sequence of decisions, plans, and actions
  - Example: Taking a skiing trip
    - Decide on dates, destinations…
    - Get to bus/train station
    - Bookings…
- What makes it hard?
  - the world is not predictable:
    - Cancellations, delays…
  - A potentially huge number of details
    - Do you consider all dates? destinations?
- AI systems are only successful in constrained planning problems

Sources of complexity

- **Computational** Complexity
  - Most of these real world problems require exponential time algorithmic solutions
  - Another example: machine translation
    - ‘Ce n’est pas une pipe’
    - How many possible English translations considering a vocabulary size of 10000?
- **Information** Complexity

  - Requires knowledge about US politicians, flags… even with a great number of computational resources

Summary of State of AI Systems in Practice

- Speech synthesis, recognition and understanding
  - very useful for limited vocabulary applications
  - unconstrained speech understanding is still too hard
- Computer vision
  - works for constrained problems (example of hand-written zip-codes on letters)
  - understanding real-world, natural scenes is still too hard
- Learning
  - adaptive systems are used in many applications: have their limits
- Planning and Reasoning
  - only works for constrained problems: e.g., chess
  - real-world is too complex for general systems
- Overall:
  - many components of intelligent systems are “doable”
  - there are many interesting research problems remaining

How to tackle these challenging AI tasks?

- Algorithmic side of problem solving
- Real-world Problem => Model => Algorithmic Solution

- When do we have a **Problem**?
  - When the present state of a system differs from its desired (goal) state
  - The first step is to represent (model) the problem.

Reflex-based Models

- Consist of
  - Input: x
  - Output: f(x), a simple function of x
- Example of sentiment analysis where x is a review and f the prediction of whether it is positive or negative (set of rules)

Example: f is set of simple rules

If x contains "cliches", return NEGATIVE.
If x contains "promise", return POSITIVE.
...

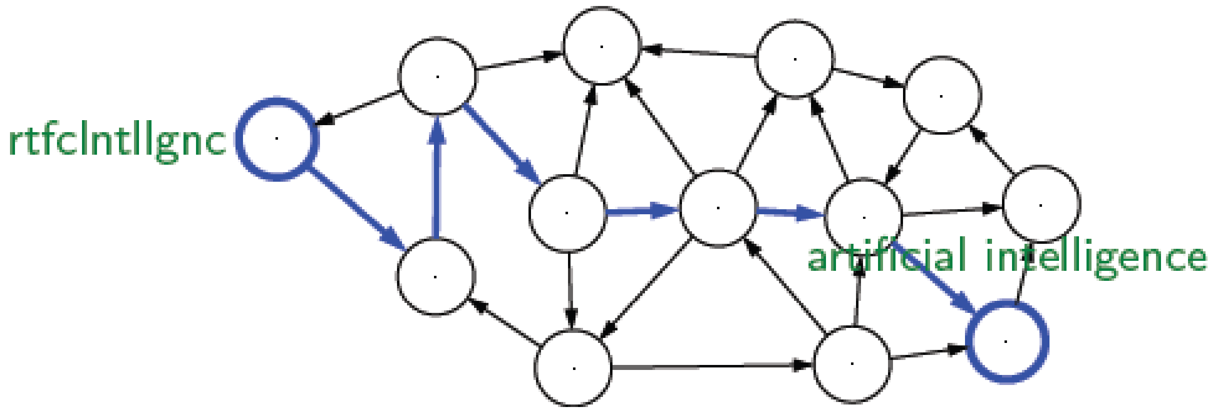
- More complicated

Example: f is based on scores

Set score $= 0$.
If x contains "cliches", score -= 10.
If x contains "promise" , score += 5.
If score > 0, return POSITIVE.

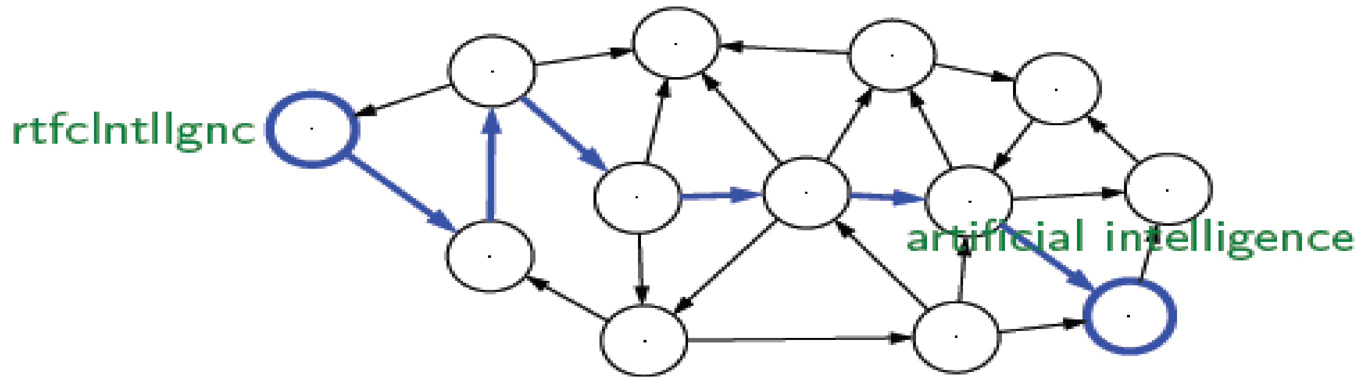
State-based Models

- Required for more complex problems requiring more forethought (e.g. chess, planning a trip)
  - Model real-world problems as finding (minimum cost) paths through graphs
- Consist of
  1. A domain of objects
  2. A set of states $S=\{S_i\}$ describing joint properties or relations over objects
  3. A set of operations $f_i : S \rightarrow S$
- Example of Text Reconstruction



Search & Solutions

- A model of a search problem
  - States captures all the relevant information about the past in order to act optimally in the future.
  - Manipulating the model (searching its state space) allows us to search for a goal state of the model
- What is a **solution**?
  - A sequence of operations from an initial state to a goal state.



State and Search Spaces

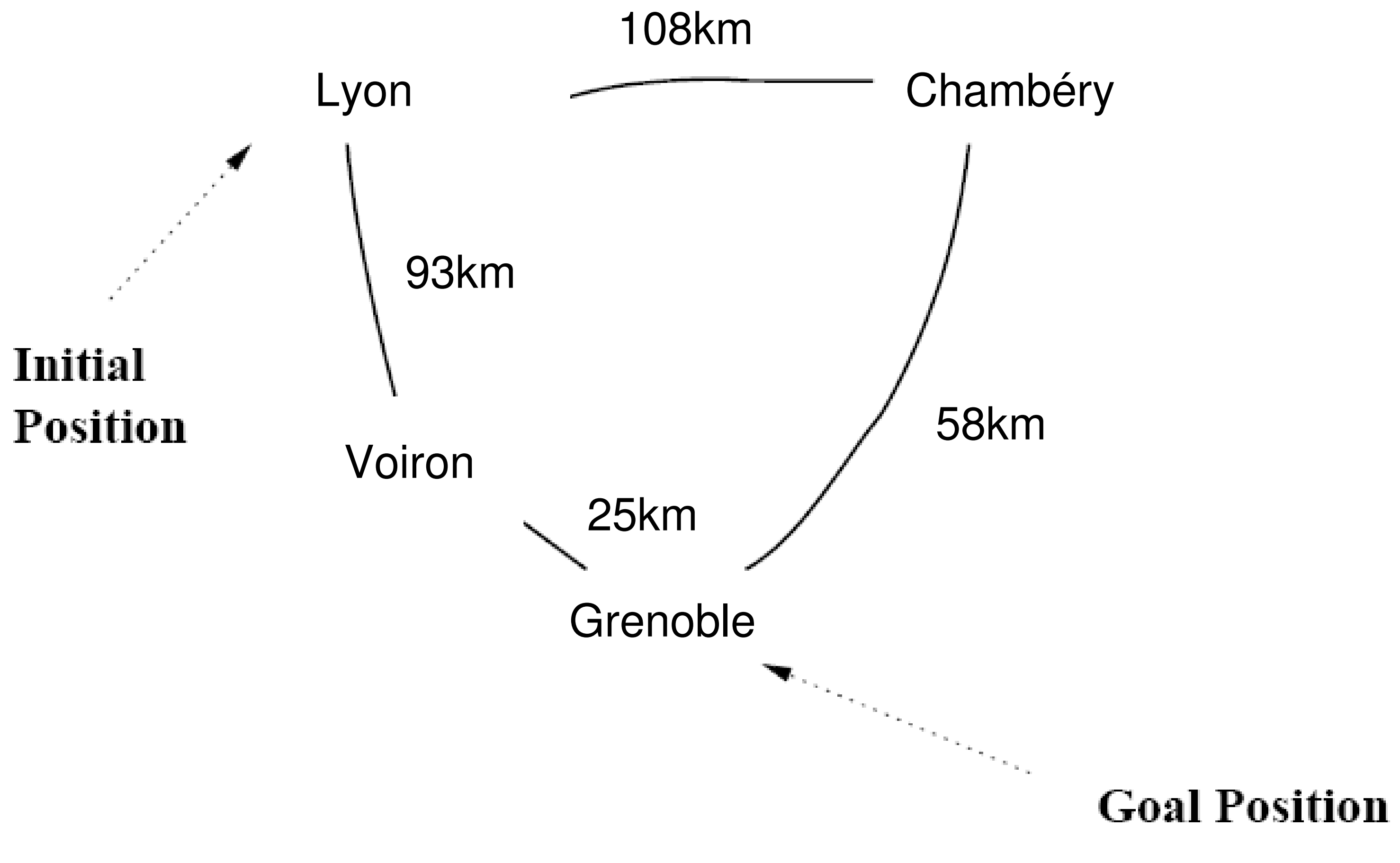
- The state space of a model is the Cartesian product of its variables.

  Example: If the model has variables $X \in \{0, 1\}$ and $Y \in \{a, b, c\}$, then $\mathcal{S} = \{\langle 0, a\rangle, \langle 0, b\rangle, \langle 0, c\rangle, \langle 1, a\rangle, \langle 1, b\rangle, \langle 1, c\rangle\}$

- The search space of a problem is the space of candidate solutions.
- The state and search spaces are often the same but not necessarily!
  - E.g., if search operators allow re-visiting states, the search space can easily become innite even if the state space is finite (or even small).

Problem Solving Example

Shortest way to Grenoble?



Problem Solving

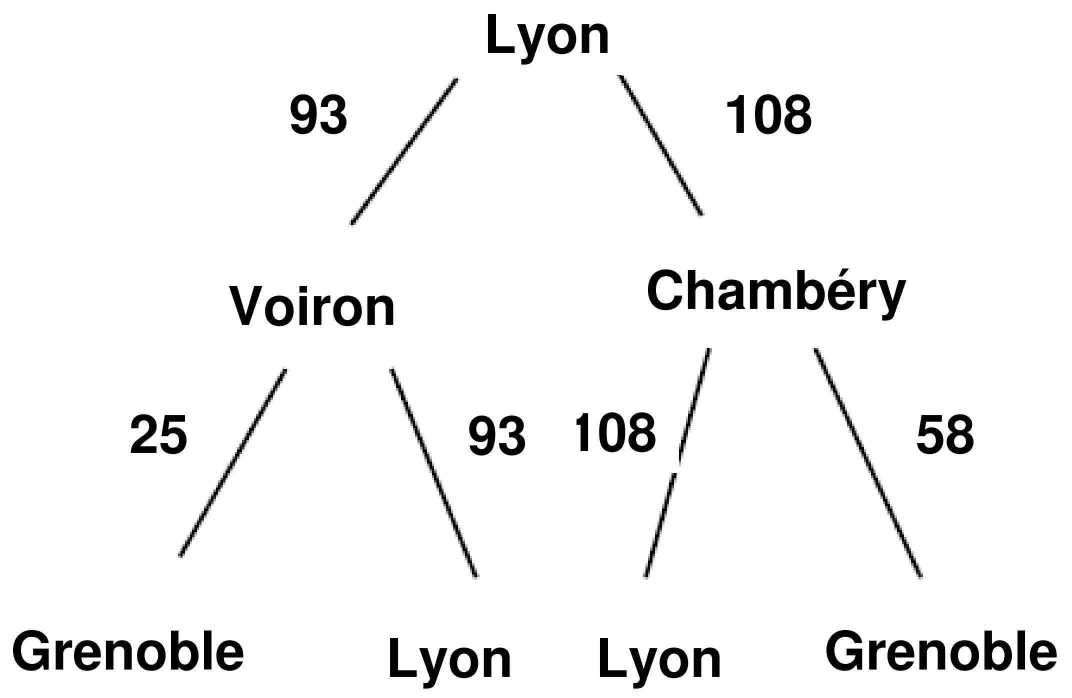
1. Find a problem; i.e., identify
   - A system
   - Its initial state So
   - Its goal states G, where So $\not\subset$ G
2. Develop a model
3. Search the model's states until a solution is found

Problem Solving Example

- We can model this with a bidirectional graph
  - Objects: yourself
  - States: your location
  - Operators: move over one edge
  - Initial state: Lyon
  - Goal state: Grenoble
  - Solution:

Problem Solving Example

- To search the ‘Lyon-Grenoble’ model, we can **expand** the initial state to make a state tree
- Breadth-first search



Modeling Choices

- Choice of model determines
  - Adequacy of the model: *can* it represent the system of interest?
  - Efficiency of the model in terms of complexity: is the search space tractable?
  - Intelligibility of the model: can people understand it?
- The main modelling choices are:
  - Level of resolution (detail) to be modelled
  - Variables: which variables? what values can they take?
  - Operators: what operations for transforming states?

Search Complexity

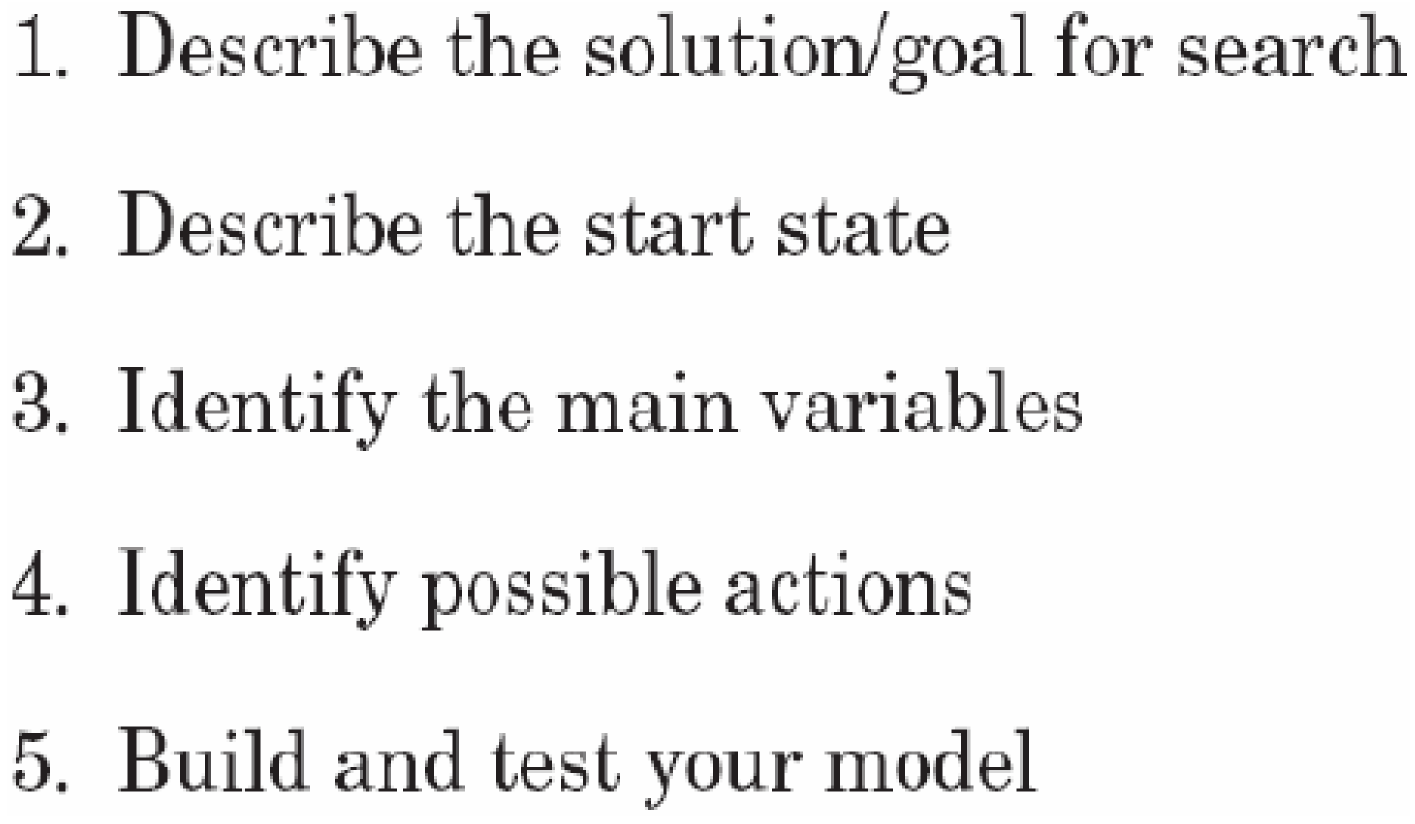
- Find representations (models) and search methods which allow for **feasible** searches and solutions.
- Feasibility?
  - CS/IT generally: A program (search, solution) which runs in polynomial time (space) or less
  - Also a program (search, solution) which satisfies problem constraints

    Example: Lyon-Voiron road closed (major accident between kms 34-35); constraint: no direct travel btw both cities
- Notation
  - S is the state space
  - $F \subseteq S$ is the feasible state space
  - $|S|$ is the size of the state space

State Space Complexity

- Number of possible states that a model has
  - ‘Lyon-Grenoble model’ has four possible states (Lyon, Chambéry, Voiron, Grenoble)
- Distinct from search space complexity.
  - E.g., if repeated states are allowed, the search space complexity is infinite
- Complexity is sensitive to choice of representation.
  - If we represent the problem with latitude-longitude pairs (without restriction), the state and search spaces go infinite

Tic-Tac-Toe



1. Describe the solution/goal for search
2. Describe the start state
3. Identify the main variables
4. Identify possible actions
5. Build and test your model

Tic-Tac-Toe

Two possible representations for Tic-Tac-Toe (noughts and crosses):

1	2	3
4	5	6
7	8	9

1. Bit string of 2 bit fields
   - 00: empty
   - 01: X
   - 10: O
   - 11: (meaningless)
2. Matrix of 0, 1, 2 (integers)

Tic-Tac-Toe

- Disadvantages of first over second representation
  - 25% of the state space is wasted
    - If search is by bit mutation, for example, then 25% of states tested will be invalid (infeasible)
    - Integer representation in matrices may waste more *storage* space (none of the numbers from 3 to maxint are meaningful), but they won't be searched
  - Neighborhood relationships (e.g., 2-in-a-diagonal) will be hard to calculate; matrix representations are easier for this.

Exercises

- Propose a model for the problems described below. Consider what variables might be used to describe states and how systems in one state might be transformed into systems in a different state. Also, identify the constraints, if any, that might be placed on the solution and whether these constraints are hard or soft.
  - (i) Path finding in a maze.
  - (ii) Evaluating the usability of a website (i.e. determining if every page is accessible from the home page through a series of mouse clicks).
  - (iii) An online dating service that tries to identify potentially compatible partners by examining their common interests.
  - (iv) A public transport route planner that lets you work out how to get from one place to another quickly and cheaply.

Introduction to Artificial Intelligence

LECTURE 2:

Complete Search VS. Heuristic Search

Overview

- Evaluation Functions
- Complete Search
- Heuristic Search
- Local Search
- Greediness

Hard Problems?

- Very large search spaces
- Hard-to-find goals
- Ill-defined goals
- Complex, Hard/Soft (e.g. timetabling) constraints

- We need evaluation functions to help guide the search
  - Generally, eval(S_i) $\approx$ distance(S_i,G) (the distance from where we are to the nearest goal state)

Evaluation functions

- Sometimes, **eval** functions are obvious, because there's a clear reward/cost to operations:
  - When driving, mileage to goal
  - When shopping, dollars to goal
- Sometimes, less so:
  - E.g., you need to get to a job interview in Grenoble within 2 hours; the direct route through Voiron is jammed up (as usual); you have about enough petrol for 130 kms; the speed limit is 110km/h (rain). What do you do?
- Given an *eval* function we can *guide* search
  - E.g., **greedy search**: apply that operator next which minimizes the *eval* function.

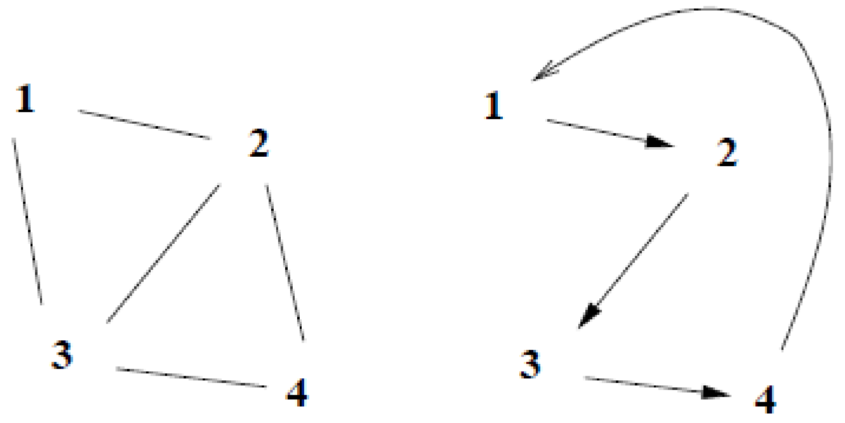
Traveling Salesman Problem

- Given a set of cities, and a list of roads connecting them, find a traversal which visits each city exactly once and which minimizes distance
  - Model?
  - Eval?
  - State & Search space complexities?

Traveling Salesman Problem

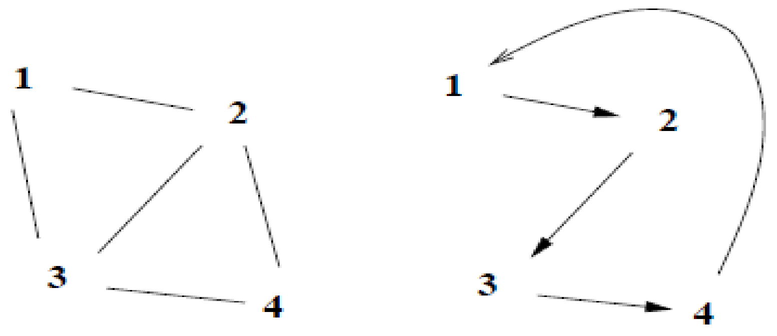
One (rather dumb) representation:

1. A DAG (chain) through all the cities
2. Complete by adding a link from last to first (hence, changing the DAG to cyclic digraph)



Implementation: (a) search through DAG space, eliminating all those DAGs which don't satisfy (1); (b) add last link to remaining digraphs; (c) minimize length among remaining rep's.

Traveling Salesman Problem



- Let states (possible answers) be permutations of cities.
  - Answer on right: (1 2 3 4), *meaning* the traversal 1-2-3-4-1.
  - NB: not all permutations are valid! (e.g., if there is no link between 2 and 3, 1-2-3-4 is invalid)

    # permutations: $n!$
- Edges are symmetric
  - E.g., $(1\ 2\ 3\ 4) \equiv (1\ 4\ 3\ 2)$

    So, we get $n!/2$
- Starting city is irrelevant; there are n of them, so divide by n

This yields: $|\mathcal{S}| = n!/2n = (n-1)!/2$

This is still exponential, but *much* smaller than the dag search space!

Problem Formulation (TSP)

Let us consider a problem of size 4 (i.e. 4 cities)

- Representation: Permutation of natural numbers 1, 2, 3,4 where each number corresponds to a city to be visited in sequence
- Search Space: Permutations of all cities. Symmetric TSP, circuit the same regardless the starting city: (n-1)!/2
- Goal: Minimize the total distance traversed, visiting each city once, and returning to the starting city.
- Evaluation Function: Map each tour to its corresponding total distance

K-SAT Problem

K-SAT = satisfiability problem with K Boolean variables.

Given $\mathcal{B}(A_1, \ldots, A_k)$, is there an assignment of truth values to the Boolean variables $A_1, \ldots, A_k$ s.t. $\mathcal{B}(A_1, \ldots, A_k)$ is true?

- Example: $A_1 \wedge A_2$ is satisfiable
- Example: $A_1 \rightarrow A_2$ is satisfiable
- Non-example: $A_1 \wedge \neg A_1$ is not satisfiable

5-SAT

- Given set of variables (A,B,C,D,E), find assignments that satisfy all clauses (or as many as possible)
  - $A \vee \neg B \vee C$
  - $\neg A \vee C \vee D$
  - $B \vee D \vee \neg E$
  - $\neg C \vee \neg D \vee \neg E$

- Model?
- Evaluation function?
- State & Search space complexities?

Problem Formulation (5-SAT)

Representation
  - 1 True, 0 False, Binary String of length 5
- Search Space
  - 2 choices for each variable, taken over 5 variables, generates 2^5 possibilities
- Objective
  - To find the vector of bits such that the compound Boolean statement is satisfied (made true)
- Evaluation Function?
  - Not easy: the expression is either true or false; what could it mean to be near or far from the truth when the expression is just false?

Complete Search

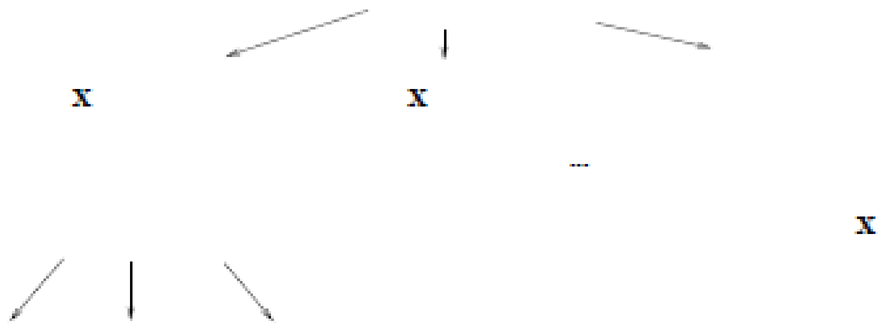
- **Exhaustive Search**
  - *Examines every state of the search space, for the optimum (given* cost*) or the goal state.*
- Exhaustive methods are worth considering because
  - Not all search spaces are as bad as TSP, etc.
  - They’re simple and provide a basis for understanding improved techniques
- Some exhaustive search methods:
  - Breadth-first search (BFS)
  - Depth-first search (DFS)
  - Any other enumeration of the state space

Complete Search

- **Enumeration**
  - *Enumerating a set* X *means finding a 1-1 mapping from* X *onto* N
  - I.e., counting everything in X. This is equivalent to exhaustively identifying each element in X and, if X = S, exhaustive search
- Basic question: how to generate every possible state?

Enumeration for Tic-Tac-Toe

We can iteratively (recursively) generate each possible move and counter-move:



Each node in the search tree is a possible solution.

What are the branching factors at the first two levels?

Exercise 2 *Determine the size of this search tree (in nodes).*

Enumerating TSP

- Given a starting city (we can dub it “1”), how to enumerate all the (n – 1)! potential traversals through the remainder? (Ignoring the symmetry issue.)
- A simple recursion TSPenum(l):

```
(i)   IF length(l)=1 RETURN l;
(ii)  FOR i =1 TO length(l)
(iii)   RETURN append($l_i$, TSPenum($l \backslash l_i$));
```

- E.g., if the input is l = (2 3), the output will be (2 3)(3 2). To these we prepend “1” to get two possible solutions, when n = 3.

Enumerating TSP

- This procedure will enumerate invalid solutions (i.e., two cities adjacent in the traversal, but not adjacent in the graph), as well as symmetrical duplicates. We can deal with that two ways:
    1. Ignore it; infeasible and duplicate possible solutions won't change the answer (but will slow down finding it!)
    2. Incorporate a call to two new functions, checking for infeasible and duplicate answers.
- The real problem with this is just the size of the search space! Enumeration is possible, but infeasible!

Enumerating SAT

Given n Boolean variables, there are 2^n possible truth-value assignments to them

(the number of bit strings of length n)

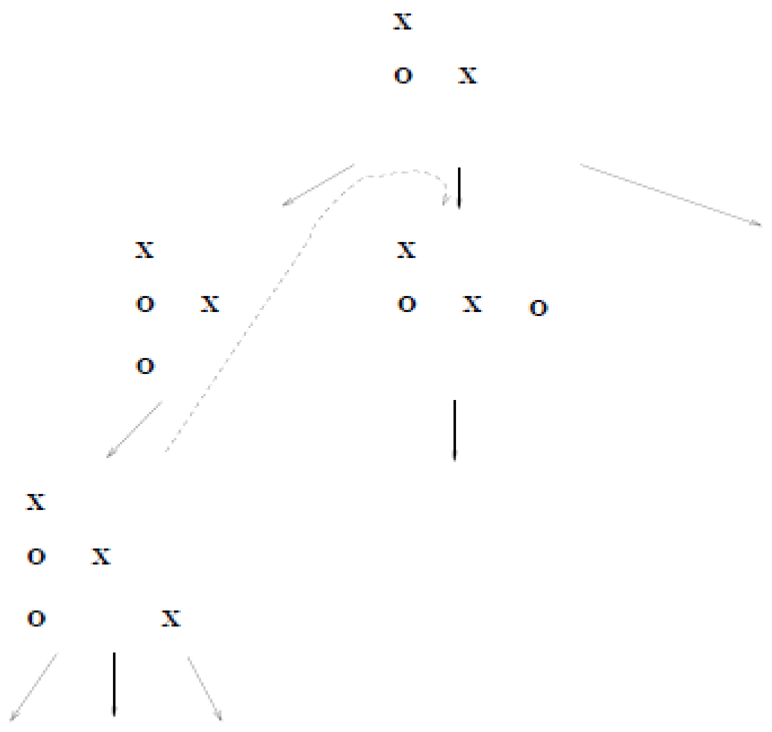
The most straightforward way of enumerating this search space is to generate a truth table for n variables. E.g.,

A	B	C	$\mathcal{B}(A, B, C)$
1	1	1	?
1	1	0	?
	...		...
0	0	0	?

Note: "Exhaustive search" need not always result in exhaustion: while generating this table, one would quit as soon as (if) $\mathcal{B}(A, B, C) = 1$, of course:)

Backtracking

Backtracking is a type of pruning, sometimes avoiding the exhaustion of exhaustive search. Noughts and Crosses (Tic-Tac-Toe) example:



Backtracking slices off a branch once it's decided no expansion of a node can lead to a solution; then it continues with the next child of the node's parent.

Heuristic algorithm(s)

- So, we move on to cheaper methods.
- From Wikipedia:
  - *Heuristic is the art and science of discovery.*
- Comes from the same Greek root as “eureka”: “I find”.
- In CS the contrast is with *algorithmic* – trading accuracy for speed. In particular, the
  - invention of a heuristic evaluation function (that *approximates* the *cost* function)
  - its use in discovery/search

Heuristics

- Some heuristics for people, from Polya’s *How to Solve It*:
  - If you are having difficulty understanding a problem, try drawing a picture.
  - If you can’t solve a problem, try assuming you have a solution and seeing what you can derive from that, working backwards.
  - If the problem is abstract, try examining a concrete example.
  - Try solving a more general problem first (the “inventor’s paradox”: the more ambitious plan may be easier).

Heuristic Search

Definition 1 (Heuristic Search) *Heuristic search uses a heuristic evaluation function $h(S)$ to speed up expected search time.*

Good eval functions:

- Are **admissible**; i.e., $\forall S\ h(S) \leq d(S, G)$ where $d(S, G)$ is the distance from S to the nearest goal state
- E.g.,
  - Minimizing path length: let h be crow-flight distance
  - Tic-Tac-Toe: let h be 3 - length of longest chain of X's

  Why are these admissible?
- Ideal if $h(S) = d(S, G)$ for all S. Then h is an **oracle**! Usually, however, this can only be done in a complex way
  - E.g., by building exhaustive search into h, defeating the purpose of heuristic search!

Heuristic Search

Be careful:

$eval$ is bad if

- $eval(S) > dist(S, G)$, for any S (and so, inadmissible).
  (Is this *really* bad? Why [not]?)
- If typically $eval(S) < eval'(S) \leq dist(S, G)$
  (i.e., $eval'$ is closer to the truth than $eval$ is).
- If computing $eval$ is slow or "infeasible".

Local Search

- Definition

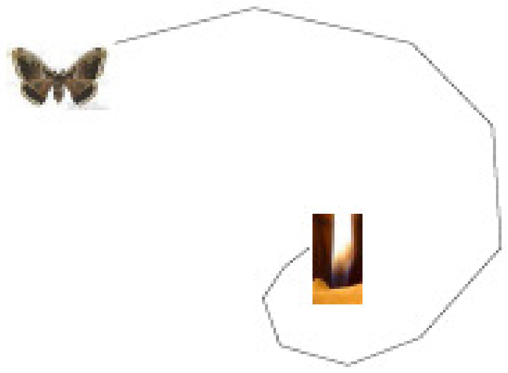
  Local search begins from an arbitrary state in the search space and looks for an improvement in the <u>neighborhood</u> of that state, until no improvement can be found. (This is a kind of iterative improvement.*)*
- Typically memoryless
- Improvement is defined in terms of *eval* - so, this is a kind of heuristic search
- Neighborhood can be dened in various ways

Local VS Global Optima

- Global solution is difficult to find
  - Consider a small subset of the search space
- Neighborhood
  - $N(S_i)$ is a set of all states that are **close to** S_i
- How to define closeness?
  - $N(S_i)=\{S_j: dist(S_i,S_j)\leq\varepsilon\}$ (dist is e.g. Euclidean)
  - Mapping on the search space
- Example for the SAT problem
  - 1-flip mapping
  - Consider the binary string 011001, neighbors?

Greediness

- Informed search VS uninformed search
- A possible greedy principle:
  - Begin with a random candidate
  - Incrementally find the best single improvement, one at a time.
- Greedy search is one which follows a local gradient



Greedy search

Here is a basic greedy search with h:

```
(1) $new \leftarrow randomstate$;
(2) LOOP: $current \leftarrow new$;
(3) FOR $i \leftarrow 1$ TO $maxtries$
(4)     [ $new \leftarrow min\ h(op(current))$;
(5)     IF $h(new) < h(current)$
(6)         THEN GOTO LOOP; ]
(7) RETURN $current$;
```

GSAT

First convert to conjunctive normal forma (CNF):

$(A_1 \vee \neg A_4 \vee \ldots \vee A_k) \wedge \ldots \wedge (\neg A_m \vee \ldots)$

- The disjunctions $(A_1 \vee \neg A_4 \vee \ldots \vee A_k)$ are called clauses
- The disjuncts $A_1, \neg A_4$ are literals

GSAT

The GSAT algorithm:

```
(1) FOR $i \leftarrow 1$ TO $maxtries$
(2)     [ $T \leftarrow random.assignment$;
(3)     FOR $j \leftarrow 1$ TO $maxflips$
(4)         [ IF $SAT(T, \mathcal{B})$ RETURN $T$
(5)         ELSE $T \leftarrow bestflip(T)$; ]]
```

bestflip chooses a single flip which results in the most clauses being satisfied.

GSAT

Example: $\mathcal{B}(A, B, C) = (A \to B) \to C$

Convert to CNF:

If the first assignment tried is $T = \langle 010 \rangle$, then *bestflip* will immediately return $T = \langle 011 \rangle$, since this satisfies both clauses.

> Nevertheless, this is a strange mix of random and local search; if the inner loop fails, a completely random restart is made.

2-opt Heuristic for TSP

“2-opt” is a simple means of solving TSP by local search:

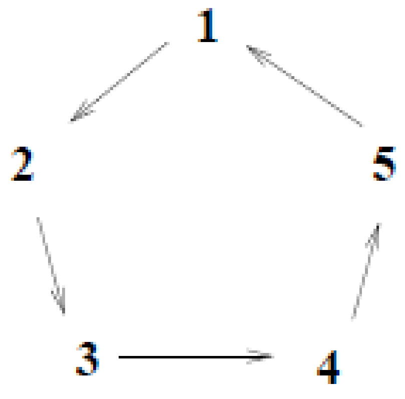
```
1. $T \leftarrow$random permutation of cities
2. for each $T' \in$ neighborhood($T$), check whether
   $cost(T') < cost(T)$
   (a) if so, then $T \leftarrow T'$ and restart 2
3. return $T$
```

Neighborhood of T = permutations obtained by deleting any two edges and replacing with two others

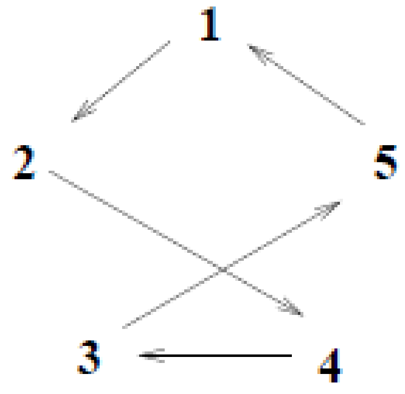
k-opt: any k edges get replaced

2-opt Heuristic for TSP

Example



(12345)



(12435)

(Note also the the arc 3 →4 had to be flipped since this is a circuit of the cities.)

Partial Solutions

- Thus far search has meant:
  - Start somewhere in the space of candidate solutions.
  - Move through that space until done.
- A different approach to search is to begin with a *partial* solution and then complete it.
- Examples of Full vs. Partial solutions
  - 3-SAT: B(A,B,C)
    - <101>
    - <1**> (where * means ‘whatever’)
  - TSP
    - (12345)
    - 2-3, 5-1

Greediness revisited

- **A Different Greedy Principle:**
  - Begin with a null solution
  - Incrementally find the best single improvement, one *dimension* at a time
- Locality in the principle above is defined in terms of dimensions. Previously, locality was defined in terms of distance through n dimensions.

Greedy k-SAT

Assume $\mathcal{B}$ is in CNF

$$\text{clause}_1 \wedge \ldots \wedge \text{clause}_k$$

Let T represent the current partial instantiation. In each case it is initialized to $\langle * \ldots * \rangle$.

Greedy Search 1

- Set the next bit which (alone) maximizes the number of satisfied clauses.

Problem case:

$$\overline{x}_1 \wedge (x_1 \vee x_2) \wedge (x_1 \vee x_3)$$

- Setting $x_1 \leftarrow 1$ satisfies two clauses
- and makes $\mathcal{B}$ impossible to satisfy

Greedy k-SAT

<u>Greedy Search 2</u>

- Sort variables by frequency of occurence, least to most
- In that order, set the value which satisfies the most clauses

Example:

$$\overline{x}_1 \wedge (x_1 \vee x_2) \wedge (x_1 \vee x_3)$$

- Order: 231 (or, 321)
- Possible search:
  $\langle * * * \rangle$ $\langle *1* \rangle$ $\langle *11 \rangle$ $\langle 011 \rangle$

Greedy k-SAT

Problem case:

$(x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (\overline{x}_1 \vee x_4) \wedge (\overline{x}_4 \ldots) \wedge (\overline{x}_4 \ldots) \wedge \ldots$

Order: 1423. . . (let missing clauses force that)

Then the search would go:

$$T \leftarrow \langle * * * * \rangle, \langle 1 * * * \rangle, \langle 1 * * 0 \rangle$$

from which there is no recovery.

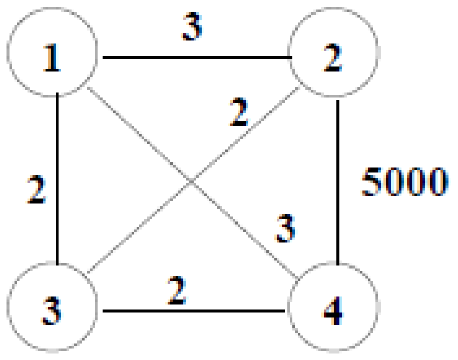
We can do better with greedy SAT. But it's always only going to be *heuristic*.

Greedy TSP Search

<u>Greedy Search 1</u>

1. current $\leftarrow$city 1
2. Loop: choose the cheapest link from current to any unconnected city. (Use random tiebreaks.)
3. Add the final link back to city 1.

This may work, but it's easy to construct counterexamples. E.g.,



Greedy Search

- Advantages: simple, fast
- Drawback: getting stuck in local optima which can very quickly get you the wrong answer
- When could greedy search be guaranteed free of this problem?
  - K-step lookahead search: embed a 2nd, 3rd … kth iterations, minimizing h(op(…(op(current_state))…)), before evaluating new
  - Group/parallel search:
    - States are sampled from space with equal probabilities
    - Explore the search space thoroughly
    - But foregoes exploiting promising regions
  - Stochastic techniques: simulated annealing, evolutionary algorithms…

3 Kinds of Search

1. *Complete (Optimal) Search:* Guaranteed to find the goal (optimum), if there is one. E.g.,
   - Exhaustive search
2. *Heuristic Search:* Not necessarily guaranteed to find the (optimal) answer. E.g.,
   - Greedy search
3. *Stochastic Search:* probabilistic search through the state space. Not guaranteed, hence a (sub)kind of heuristic search. E.g.,
   - Genetic algorithm, simulated annealing