CS361 ***(Artificial Intelligence)***

Lecture 2
Problem Solving as Search

Dr. Hala Abdel-Galil & Dr. Amr S. Ghoneim
(Computer Science Dept.)

Helwan University
Fall 2019

Lectures are based on their counterparts in the following courses:
- *Intelligent Systems*, **University of British Columbia** (Dept. of Computer Science)
- *Introduction to Artificial Intelligence*, **University of Wisconsin-Madison**
- *Artificial Intelligence*, **University of Illinois at Urbana-Champaign**
- *Artificial Intelligence*, **University of California, Berkeley**

Resources for this lecture

- This lecture covers the following chapters:
  - **Chapter 3 (Solving Problems by Searching; only sections 3.1, 3.2, & 3.3)** *from* **Stuart J. Russell and Peter Norvig, "Artificial Intelligence: A Modern Approach," Third Edition (2010), by Pearson Education Inc.**

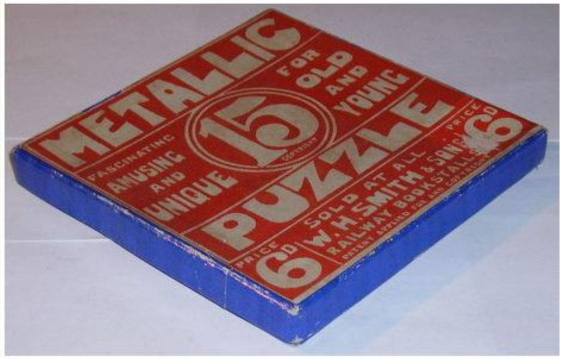
  … AND …

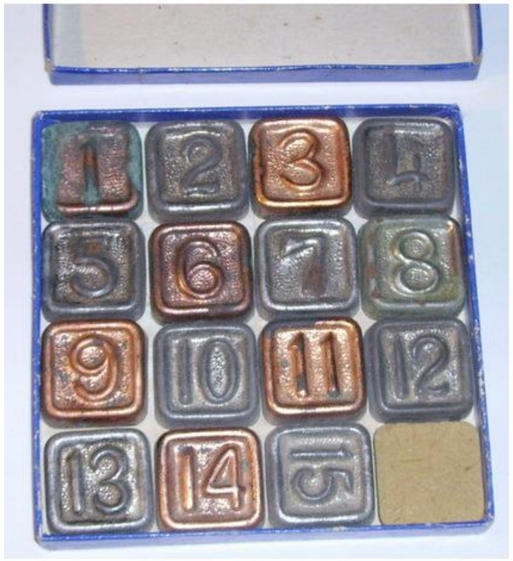
  - **Part II (pages 41 to 45) and Chapter 3 (only sections 3.0, 3.1, and 3.2)** *from* **George F. Luger, "Artificial Intelligence: Structures and strategies for complex problem solving, " Sixth edition (2009), Pearson Education Limited.**

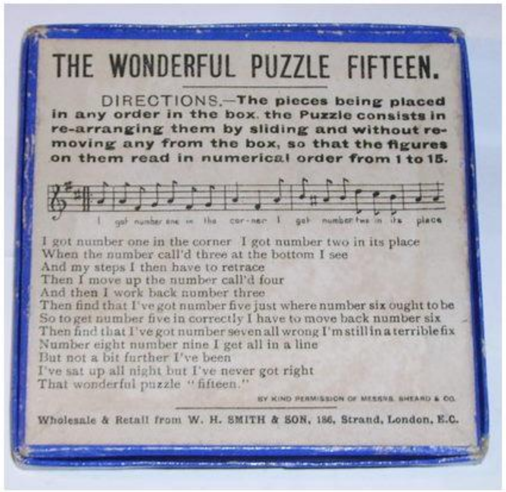
Outline

- Solving problems by Searching
  - Types of agents
  - State Space Search
    - Example: Tic-Tac-Toe Game
    - Example: Mechanical Fault Diagnosing
  - How human beings think?
    - Heuristic Search
  - Search
    - Search problem components
    - Example: Romania
    - State space
    - Example: Vacuum world
    - Vacuum world state space graph
    - Example: The 8-puzzle
    - Example: Robot motion planning
    - Example: Tic- Tac-Toe
    - Example: Traveling Salesperson
- State Space Search Strategies
  - State Space Search Strategies: Selecting Search Strategy
- Search: Basic idea
- Search tree
  - Tree Search Algorithm Outline
  - Tree search example
  - Handling repeated states
  - Backtracking Search
  - Backtracking Algorithm Data Structure
  - Backtracking Algorithm

Solving Problems by SEARCHING







THE WONDERFUL PUZZLE FIFTEEN.

DIRECTIONS.—The pieces being placed in any order in the box, the Puzzle consists in re-arranging them by sliding and without removing any from the box, so that the figures on them read in numerical order from 1 to 15.

I got number one in the corner I got number two in its place

I got number one in the corner I got number two in its place
When the number call'd three at the bottom I see
And my steps I then have to retrace
Then I move up the number call'd four
And then I work back number three
Then find that I've got number five just where number six ought to be
So to get number five in correctly I have to move back number six
Then find that I've got number seven all wrong I'm still in a terrible fix
Number eight number nine I get all in a line
But not a bit further I've been
I've sat up all night but I've never got right
That wonderful puzzle "fifteen."

BY KIND PERMISSION OF MESSRS. SHEARD & CO.

Wholesale & Retail from W. H. SMITH & SON, 186, Strand, London, E.C.

Types of Agents

Reflex agent

Consider how the world IS

- Choose action based on current percept.
- Do not consider the future consequences of actions.

Planning agent

Consider how the world WOULD BE

- Decisions based on (hypothesized) consequences of actions.
- Must have a model of how the world evolves in response to actions.
- Must formulate a goal.

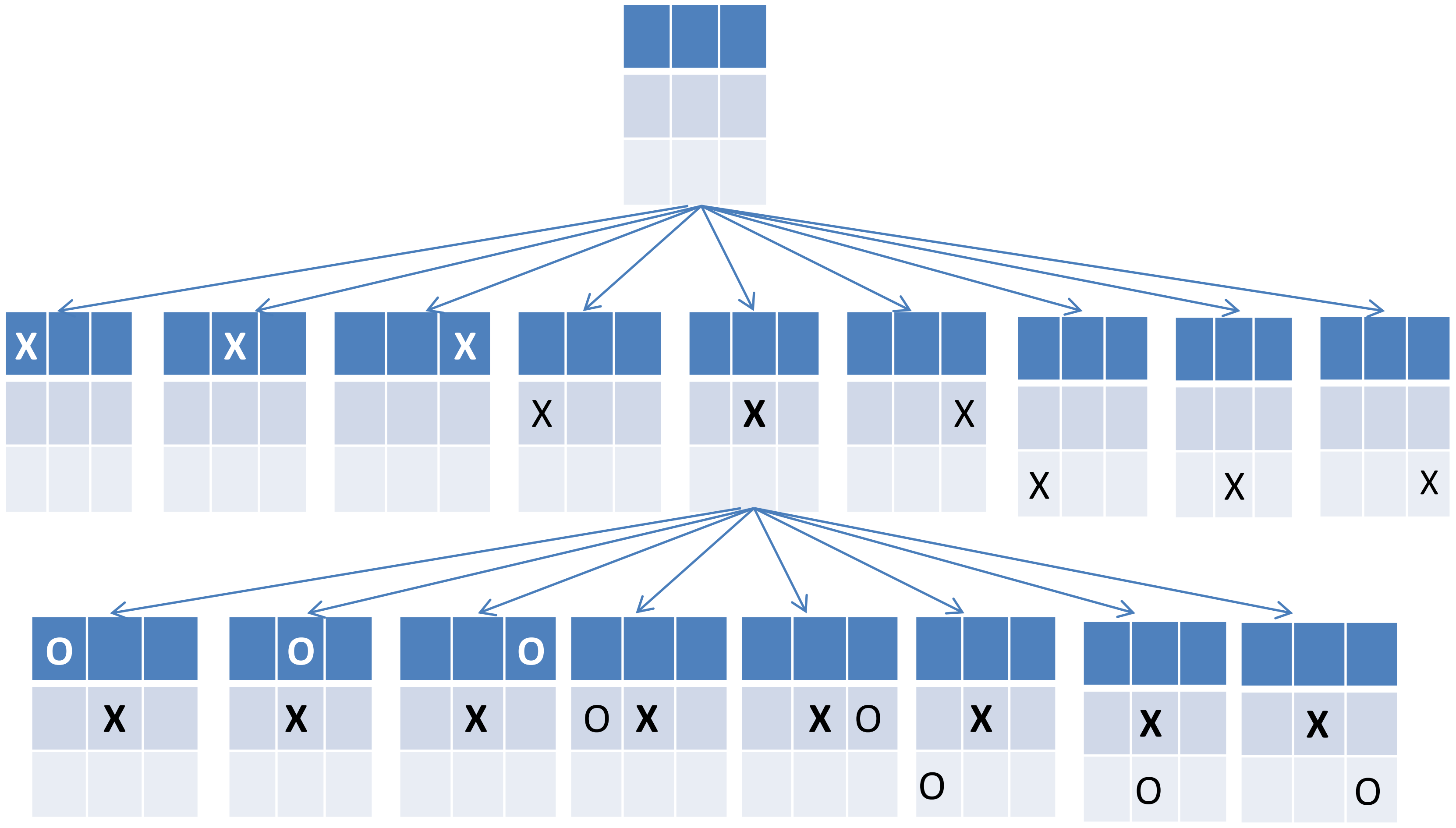
Source: D. Klein, P. Abbeel

State Space Search

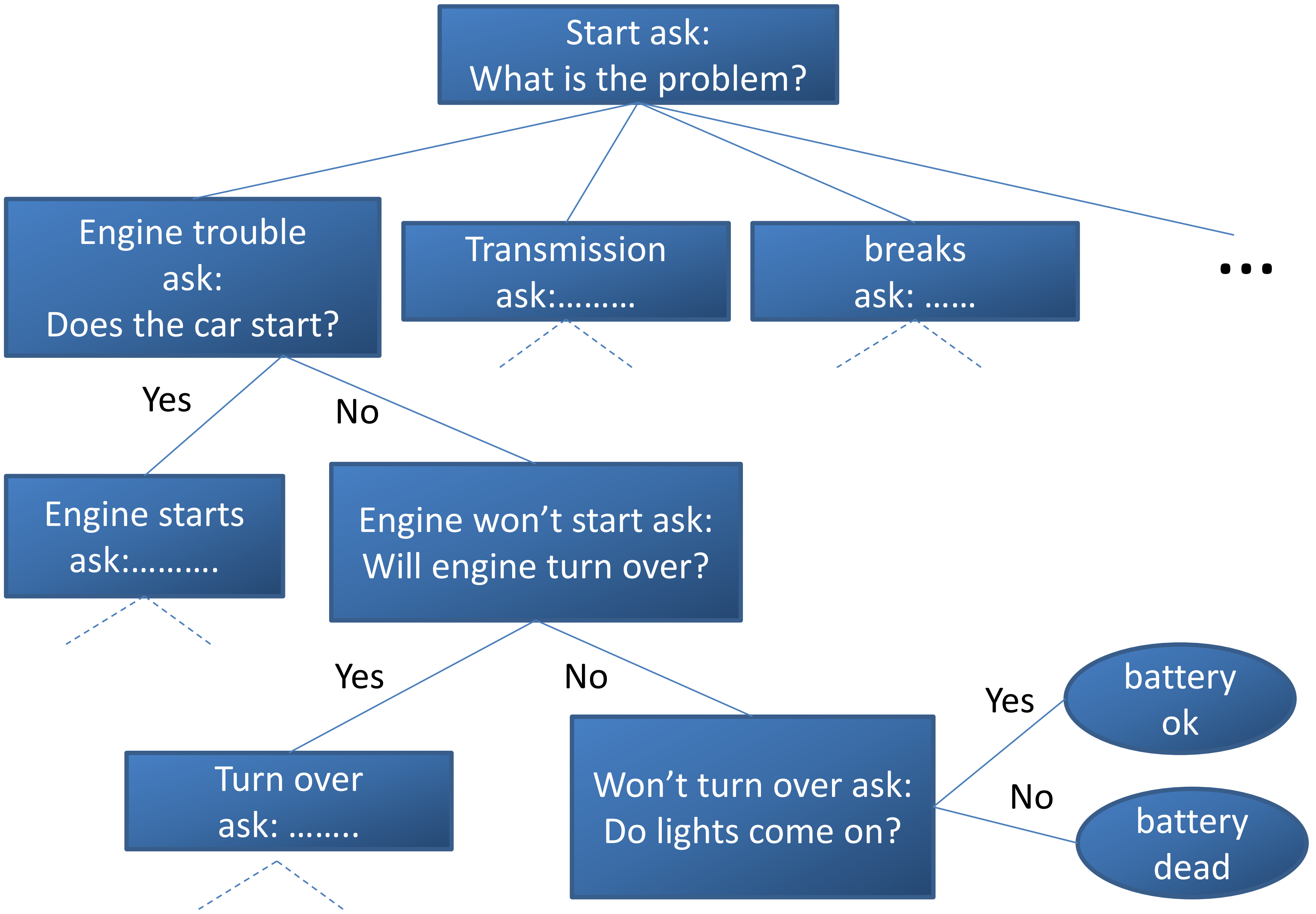
Problems are solved by searching among alternative choices.

- Humans consider a number of alternative strategies on their way to solving a problem.
  - A *Chess* player considers a number of alternative moves.
  - A *mathematician* chooses from a different strategies to find a proof for a theorem.
  - A *physician* evaluates a number of possible diagnoses.

Example: Tic-Tac-Toe Game



Example: Mechanical Fault Diagnosing



How human beings think?

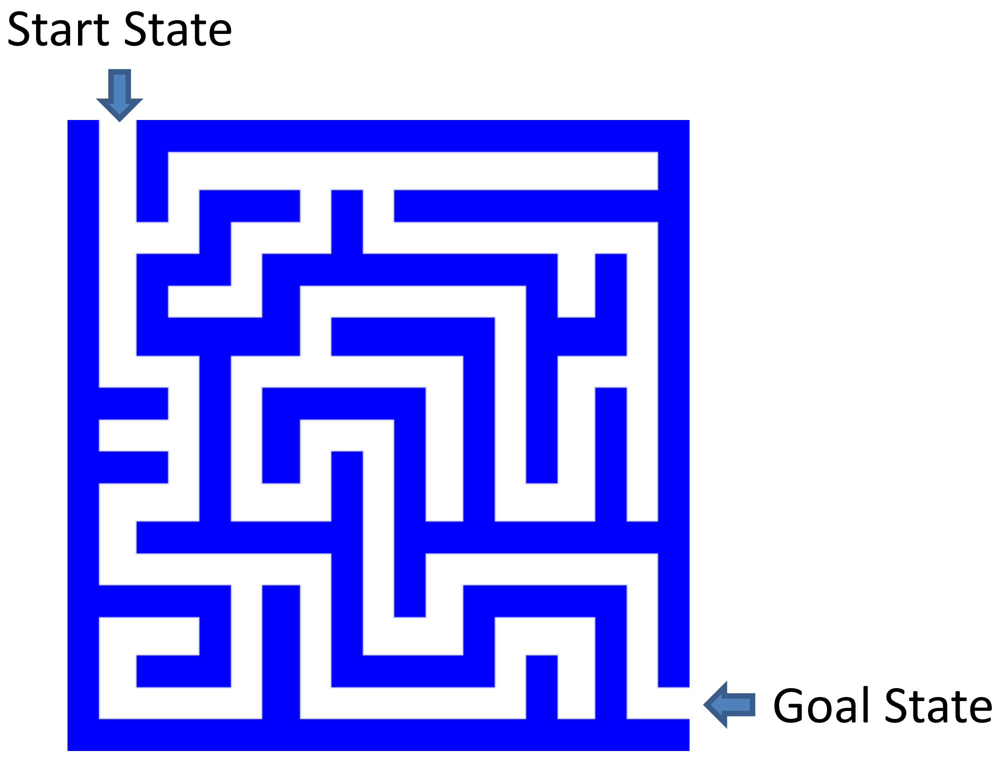
- Human beings do not search the entire state space (*exhaustive search*).
- Only alternatives that experience has shown to be effective are explored.
- Human problem solving is based on judgmental rules that limit the exploration of search space to those portions of state space that seem somehow promising.
- These judgmental rules are known as *"heuristics".*

Heuristic Search

- A heuristic is a strategy for selectively exploring the search space.
- It guides the search along lines that have a high probability of success.
- It employs knowledge about the nature of a problem to find a solution.
- It does not guarantee an optimal solution to the problem but can come close most of the time.
- Human beings use a large number of heuristics in problem solving.

Search

We will consider the problem of designing **goal-based agents** in **fully observable**, **deterministic**, **discrete**, **known** environments.



Search

We will consider the problem of designing **goal-based agents** in **fully observable**, **deterministic**, **discrete**, **known** environments.

The agent must find a *sequence of actions* that reaches the goal. The **performance measure** is defined by:

(a) reaching the goal, and …

(b) how “expensive” the path to the goal is.

Search Problem Components

Initial state

Actions

Transition model

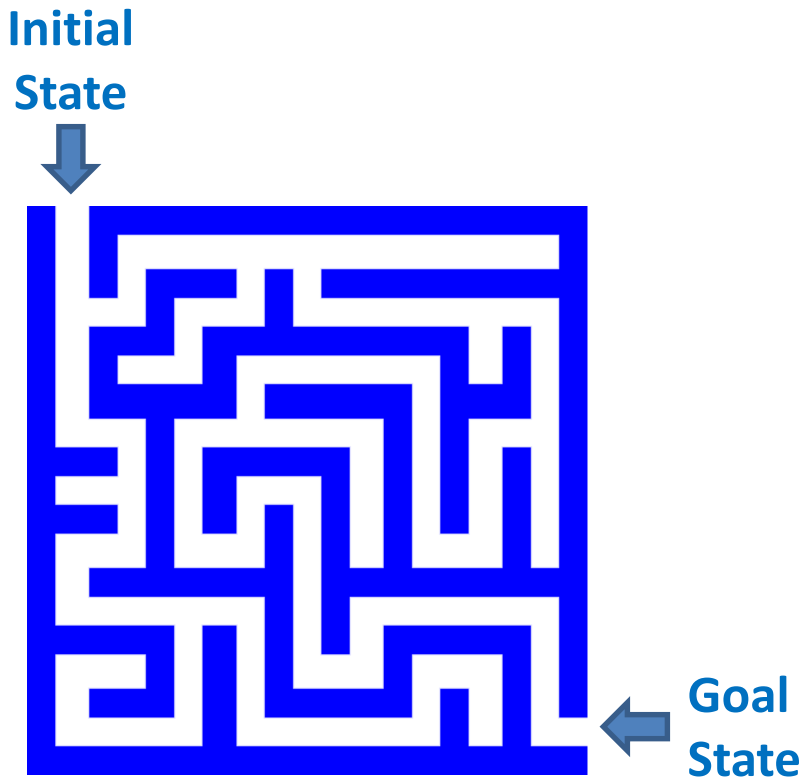
What state results from performing a given action in a given state?

Goal state

Solution Path

Path cost

Assume that it is a sum of nonnegative *step costs*

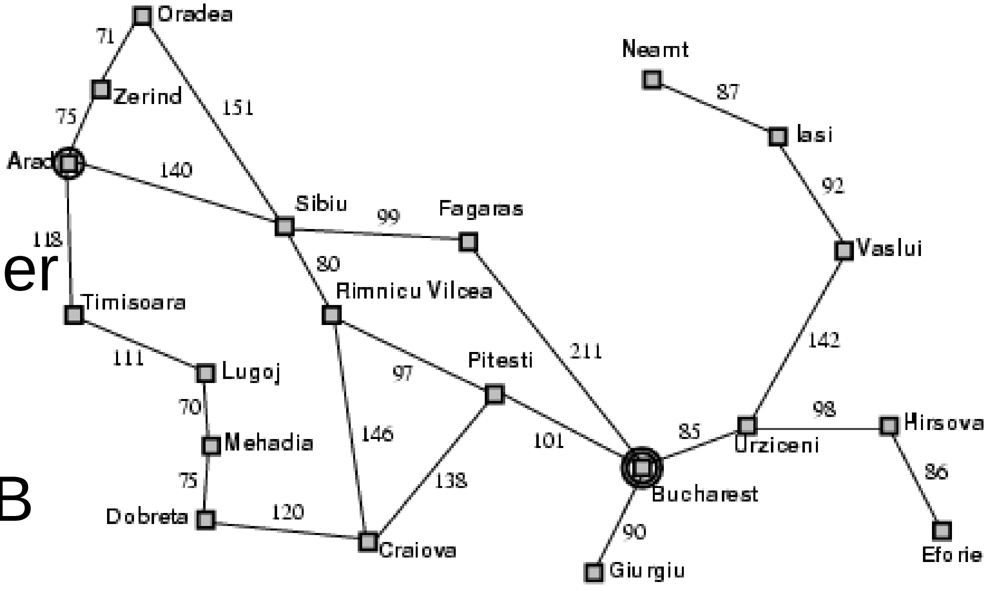


The **optimal solution** is the sequence of actions that gives the *lowest* path cost for reaching the goal.

Example: Romania

- On vacation in Romania; currently in Arad.

- Flight leaves tomorrow from Bucharest.



Initial state

- Arad

Actions

- Go from one city to another

Transition model

- If you go from city A to city B, you end up in city B

Goal state

- Bucharest

Path cost

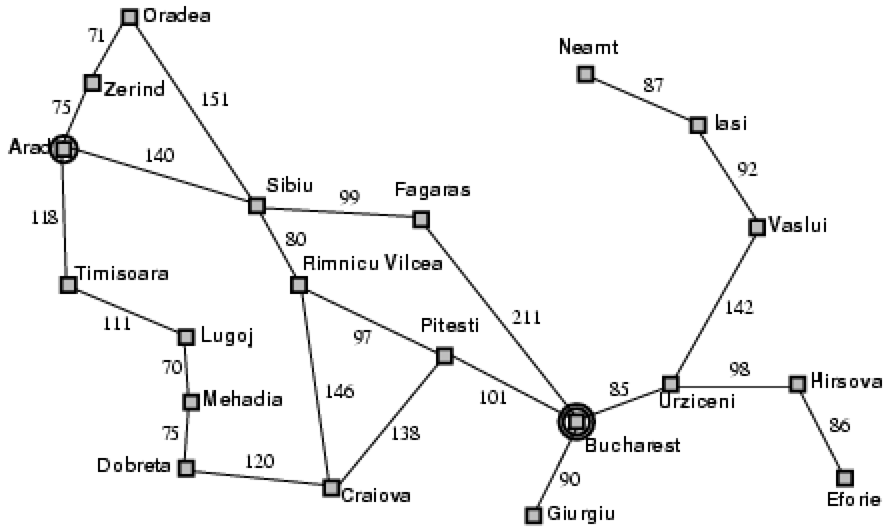
- Sum of edge costs *(total distance traveled)*

State Space

The initial state, actions, and transition model define the **state space** of the problem;

- The set of all states reachable from initial state by any sequence of actions.
- Can be represented as a **directed graph** where the nodes are states and links between nodes are actions.

What is the state space for the Romania problem?



State Space

An AI problem can be represented as a *state space graph.*

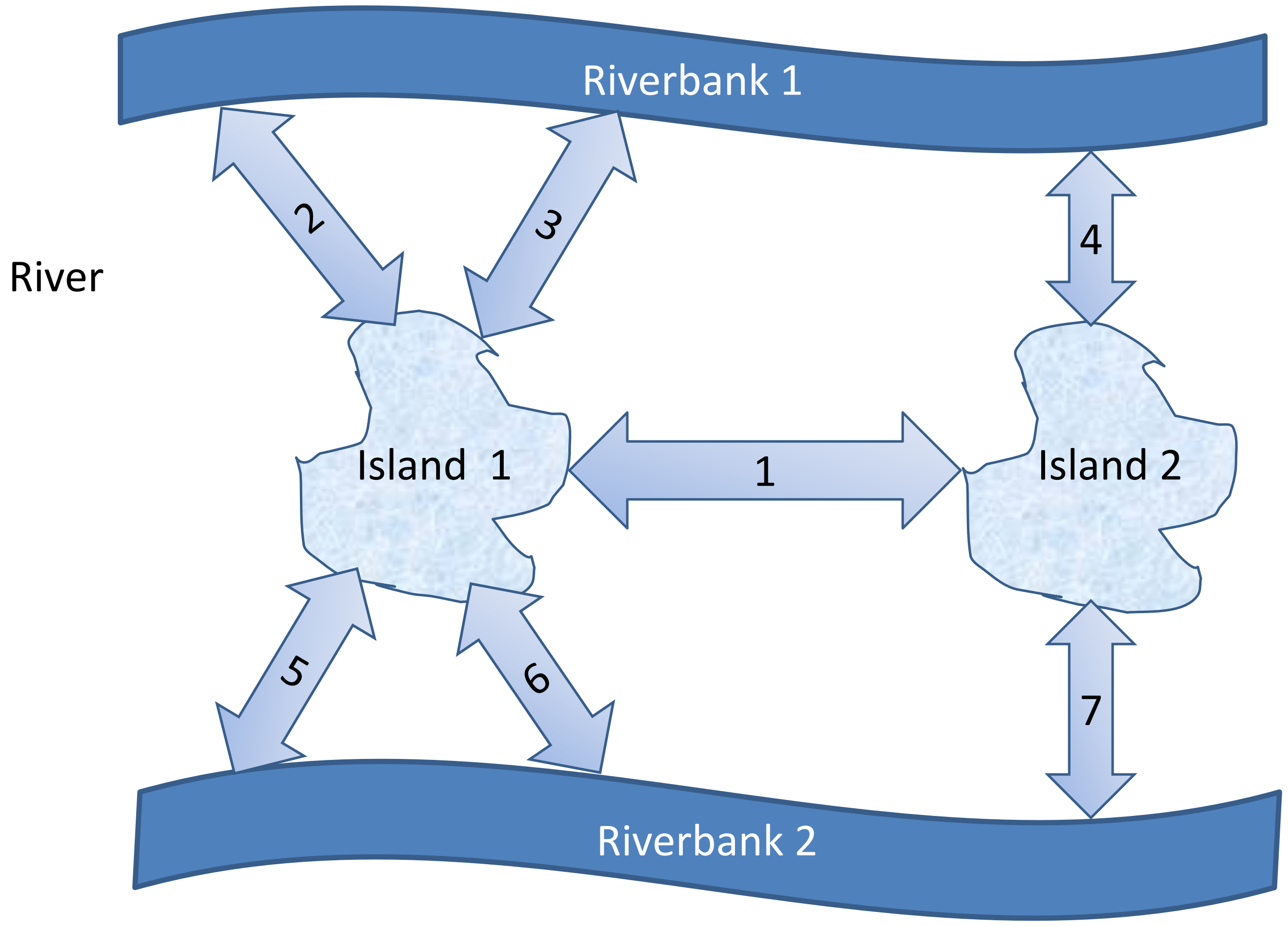
A **graph** is a set of *nodes* and *links* that connect them.

Graph theory:

- Labeled graph.
- Directed graph.
- Path.
- Rooted graph.
- Tree.
- Parent.
- Child.
- Sibling.
- Ancestor.
- Descendant.

State Space

Graph Theory, Kônigsberg Bridges Problem, & Euler Tour ..

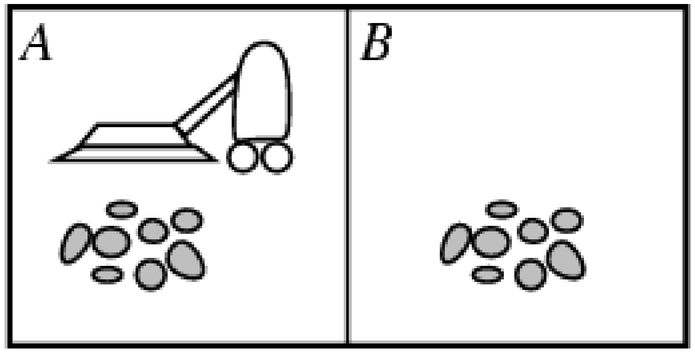


State Space

A state space is represented by four-tuple [N, A, S, GD].

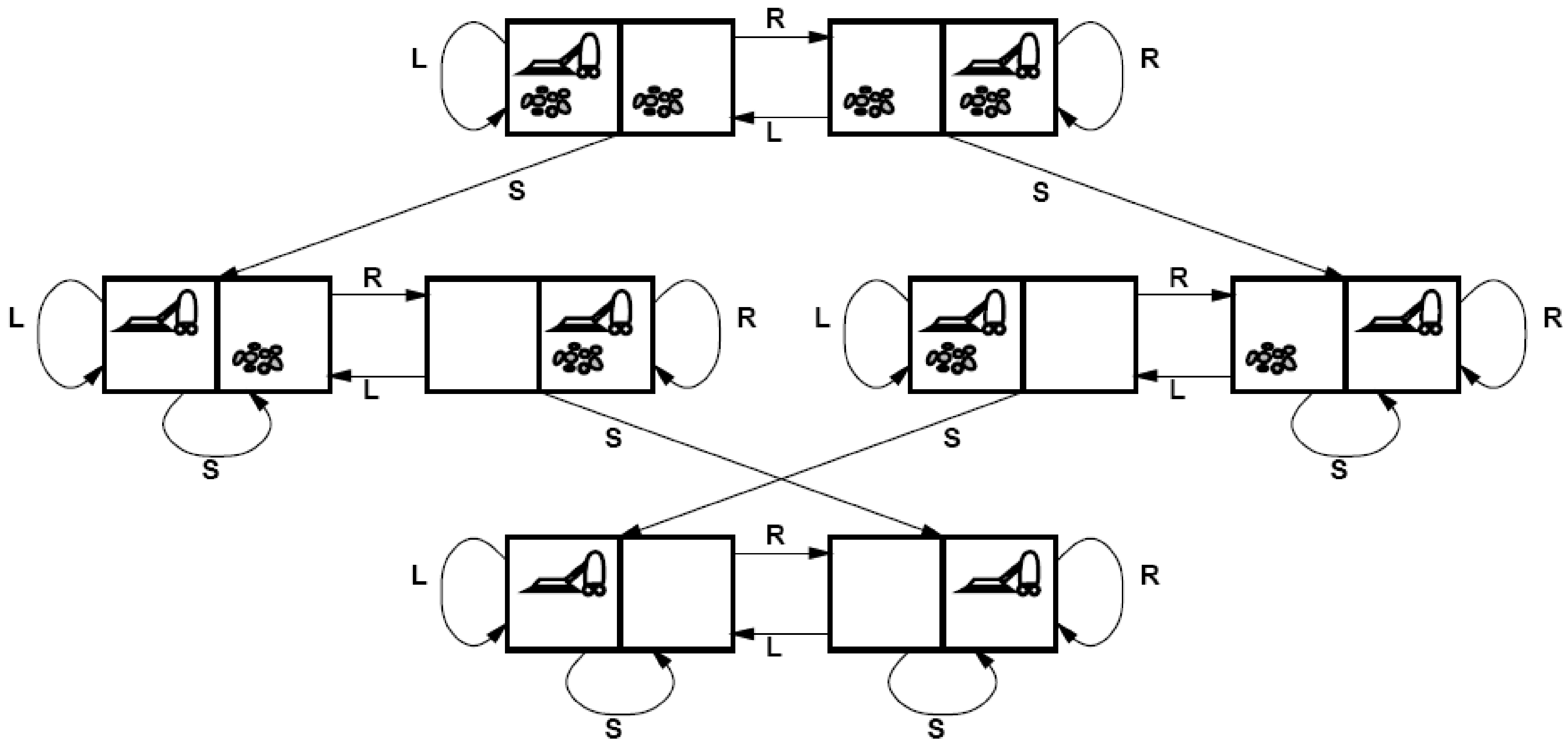
- **N,** is the set of **nodes** or states of the graph. These correspond to the states in the problem-solving process.
- **A,** is the set of **arcs** between nodes. These correspond to the steps in a problem-solving process.
- **S,** a nonempty subset of N, contains the **start-state (s)** of the problem.
- **GD,** a nonempty subset of N, contains the **goal-state(s)** of the problem. The states in GD are described using either:
  - A measurable property of the states encountered in the search.
  - A property of the solution path developed in the search *(a **solution path** is a path through this graph from a node in S to a node in GD)*.

Example: Vacuum World



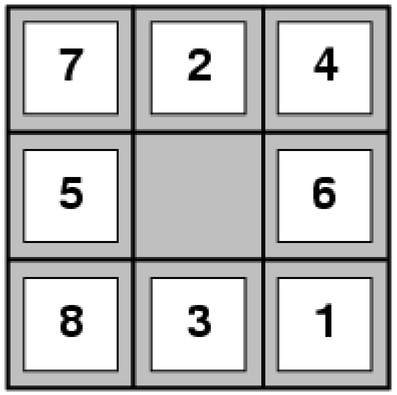
- **States:**
  - Agent location and dirt location
  - How many possible states?
  - What if there are *n* possible locations?
    - The size of the state space grows exponentially with the “size” of the world!
- **Actions:**
  - Left, right, suck.
- **Transition model ..**

Example: Vacuum World State Space Graph

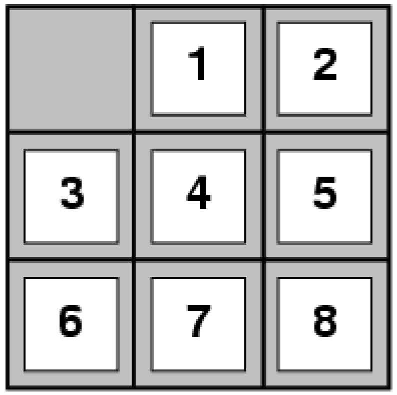


Example: The 8-puzzle

- States
  - Locations of tiles
    - 8-puzzle: 181,440 states (9!/2)
    - 15-puzzle: ~10 trillion states
    - 24-puzzle: ~10^{25} states
- Actions
  - Move blank left, right, up, down
- Path cost
  - 1 per move

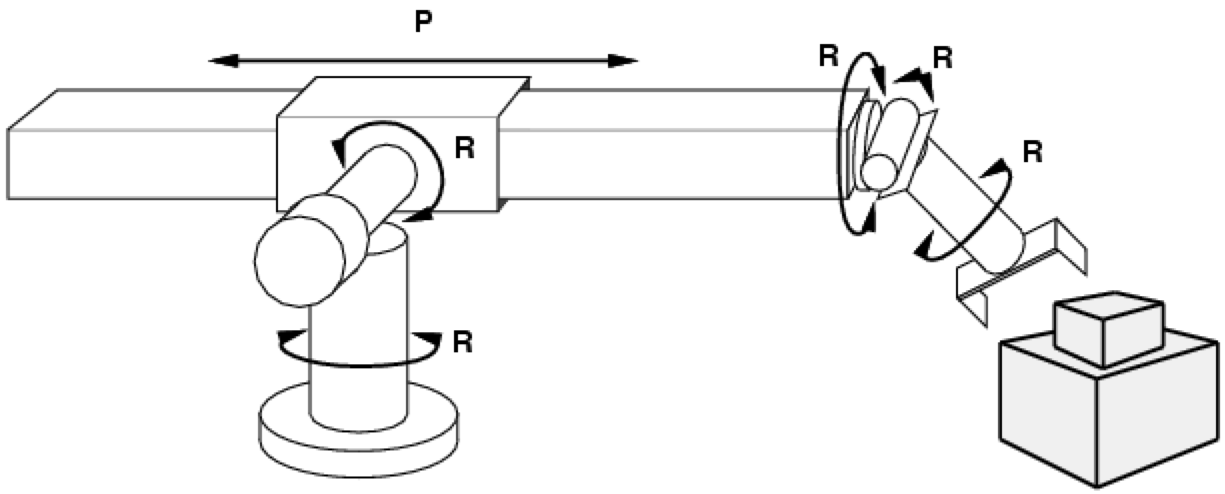


Start State



Goal State

Example: Robot motion planning



- **States**
  - Real-valued joint parameters (angles, displacements).
- **Actions**
  - Continuous motions of robot joints.
- **Goal state**
  - Configuration in which object is grasped.
- **Path cost**
  - Time to execute, smoothness of path, etc.

Example: Tic-Tac-Toe

- Nodes(N): all the different configuration of Xs and Os that the game can have.
- Arcs (A): generated by legal moves by placing an X or an O in unused location.
- Start state (S): an empty board.
- Goal states (GD): a board state having three Xs in a row, column, or diagonal.
- The arcs are directed, then no cycles in the state space, *directed acyclic graph* (DAG).
- Complexity: 9! Different paths can be generated.

Example: Traveling Salesperson

A sales person has five cites to visit and ten must return home.

- Nodes(N): represent 5 cites.
- Arcs(A): labeled with weight indicating the cost of traveling between connected cites.
- Start state(S): a home city.
- Goal states(GD): an entire path contains a complete circuit with minimum cost.
- Complexity: (n-1)! Different cost-weighted paths can be generated.

State Space Search Strategies

There are two distinct ways for searching a state space graph:

❖**Data-Driven Search:** (Forward chaining)
Start searching from the given data of a problem instance toward a goal.

❖**Goal-Driven Search:** (Backward chaining)
Start searching from a goal state to facts or data of the given problem.

State Space Search Strategies ..
Selecting Search Strategy

Data-Driven Search is suggested if:
- The data are given in the initial problem statement.
- There are few ways to use the given facts.
- There are large number of potential goals.
- It is difficult to form a goal or hypothesis.

Goal- Driven Search is appropriate if:
- A goal is given in the problem statement or can easily formulated.
- There are large number of rules to produce a new facts.
- Problem data are not given but acquired by the problem solver.

Search

Given:

- **Initial state**
- **Actions**
- **Transition model**
- **Goal state**
- **Path cost**

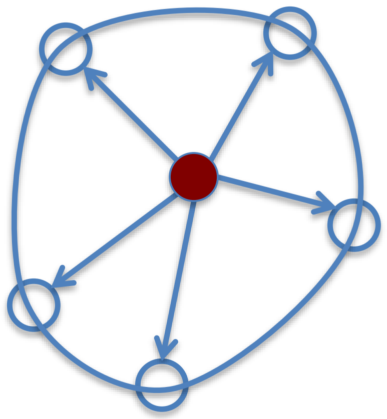
How do we find the optimal solution?

Search: Basic idea

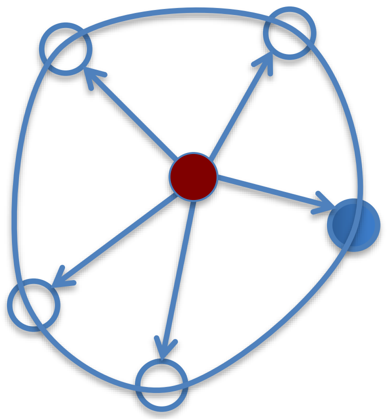
- Let’s begin at the start state and **expand** it by making a list of all possible successor states.
- Maintain a **frontier** or a list of unexpanded states.
- At each step, pick a state from the frontier to expand.
- Keep going until you reach a goal state.
- Try to expand as few states as possible.

Search: Basic idea

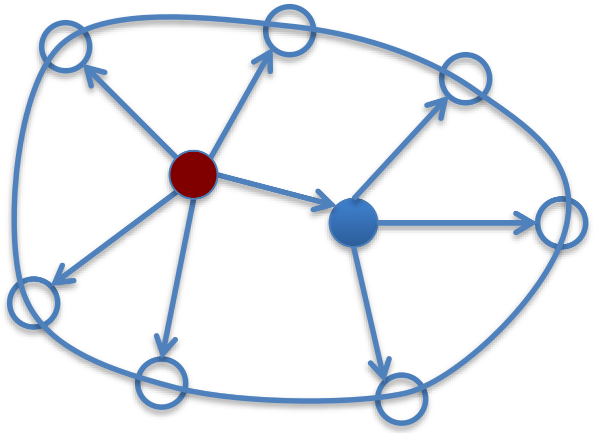
Search: Basic idea



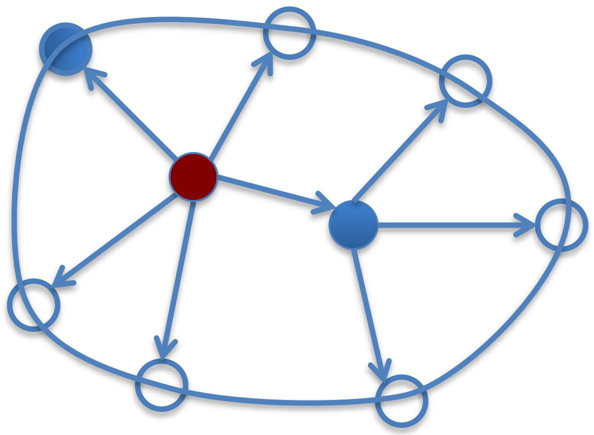
Search: Basic idea



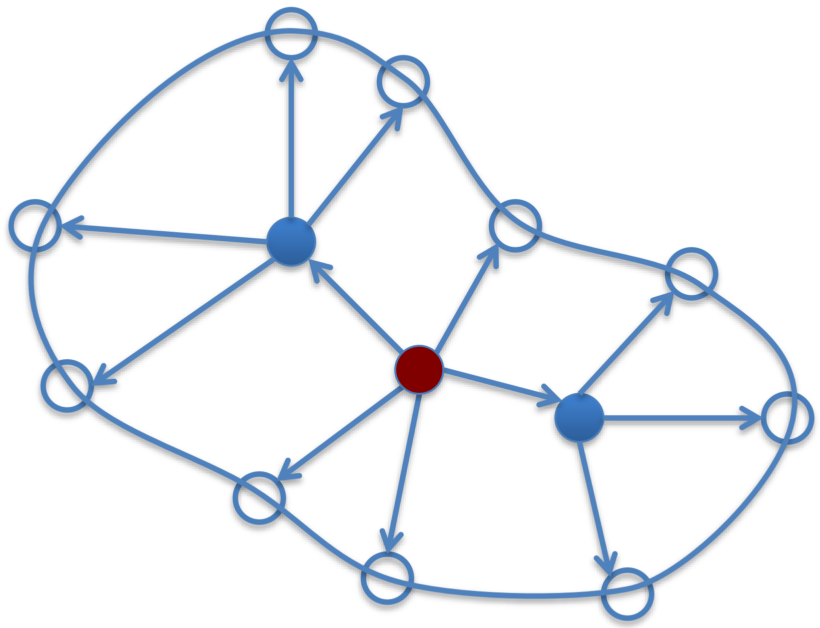
Search: Basic idea



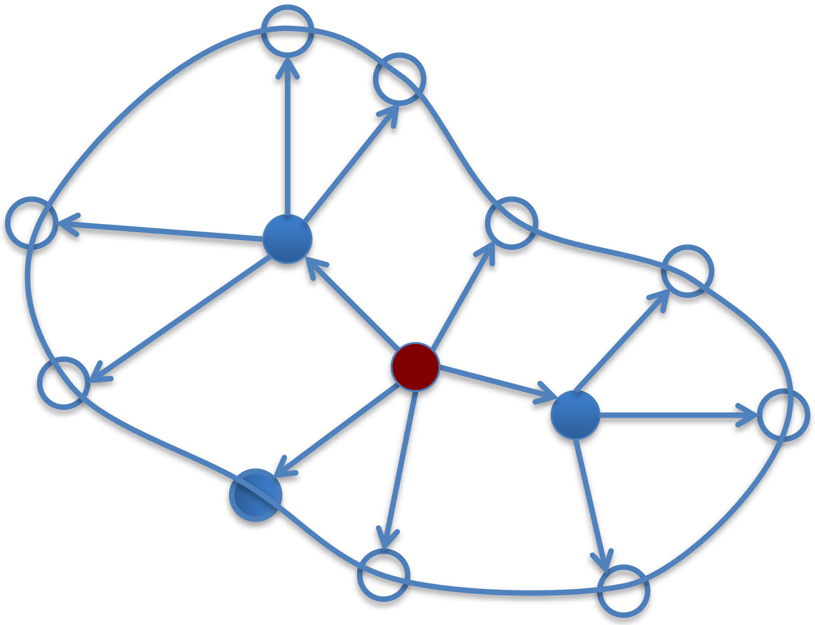
Search: Basic idea



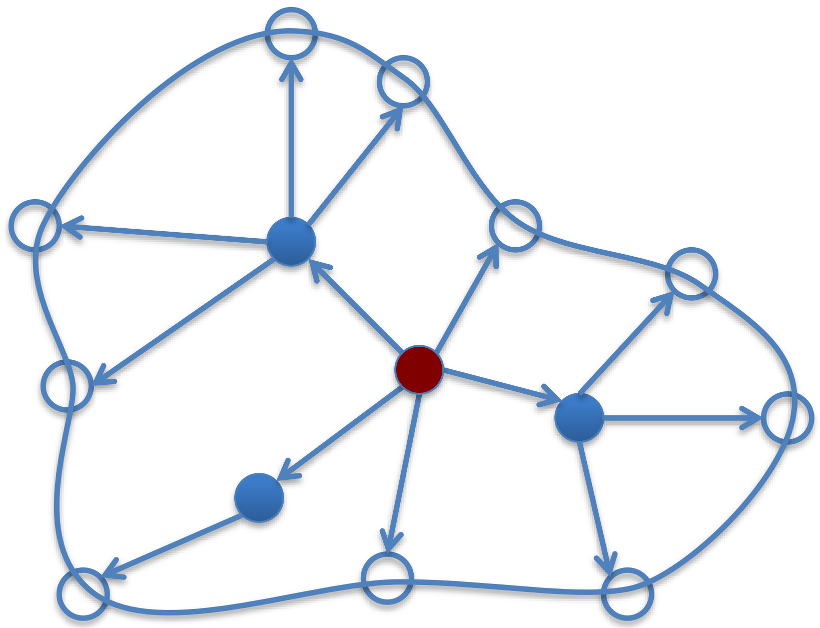
Search: Basic idea



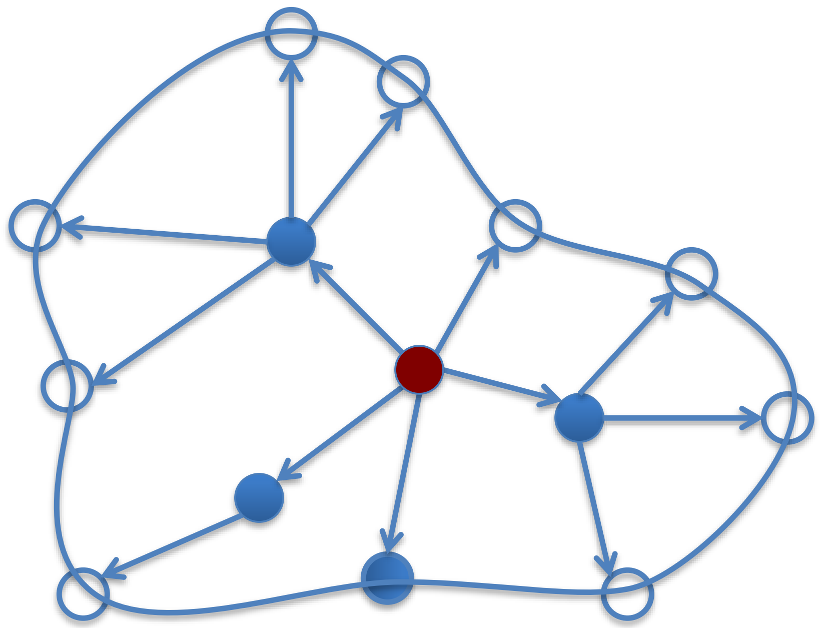
Search: Basic idea



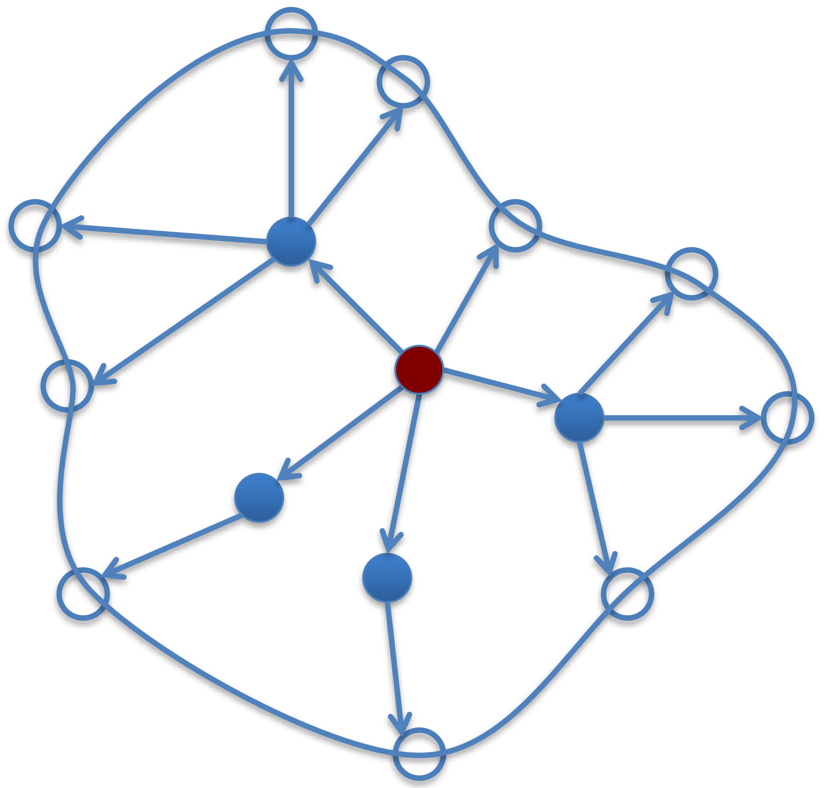
Search: Basic idea



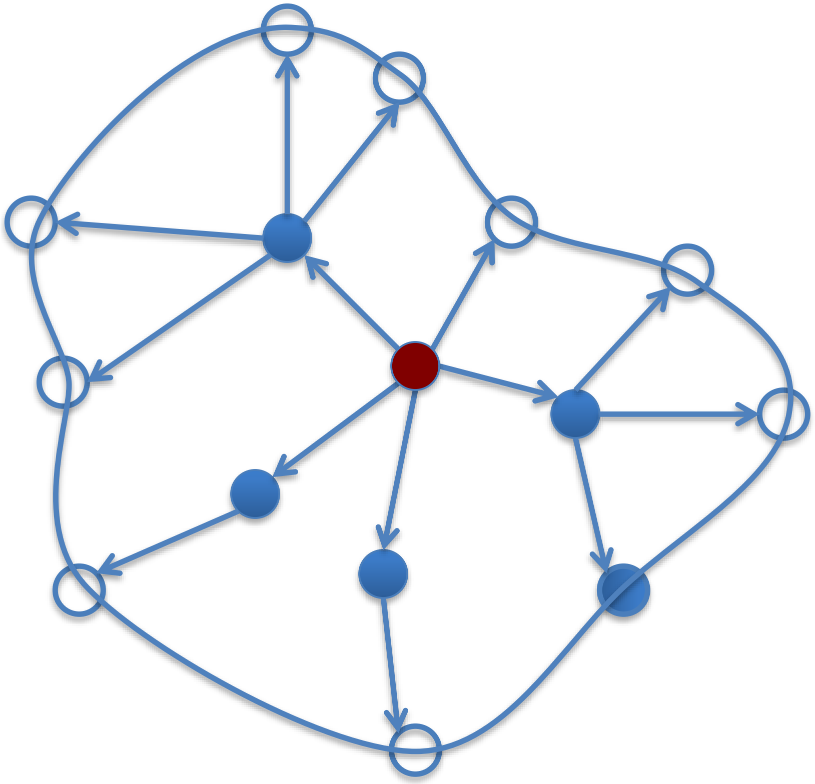
Search: Basic idea



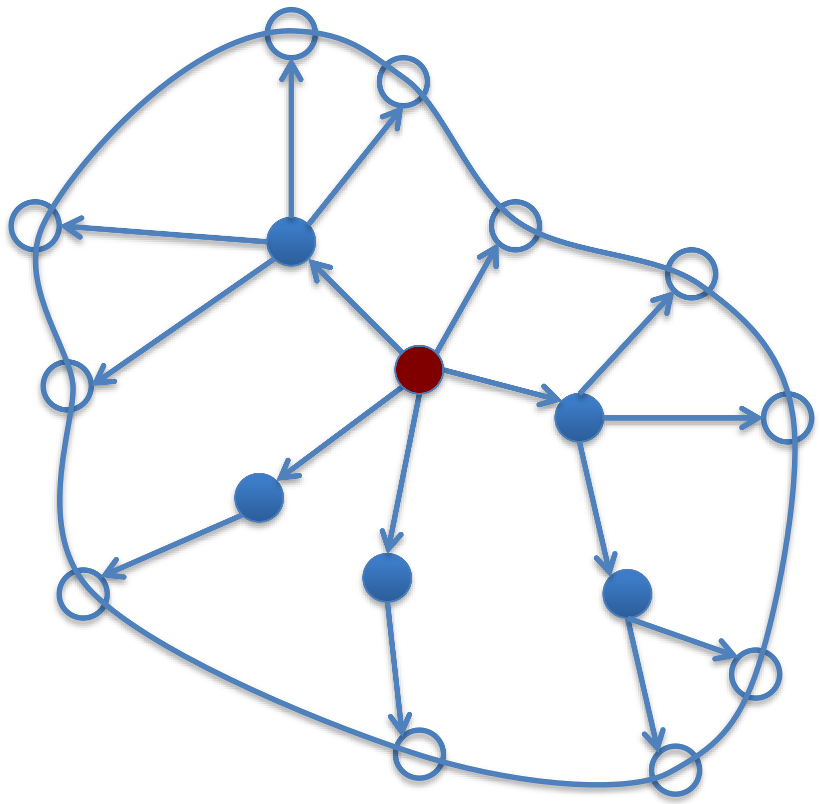
Search: Basic idea



Search: Basic idea



Search: Basic idea



Search Tree

"What if" tree of sequences of actions and outcomes

- ❖ When we are searching, we are not acting in the world, merely "thinking" about the possibilities

The root node corresponds to the starting state

The children of a node correspond to the **successor states** of that node's state

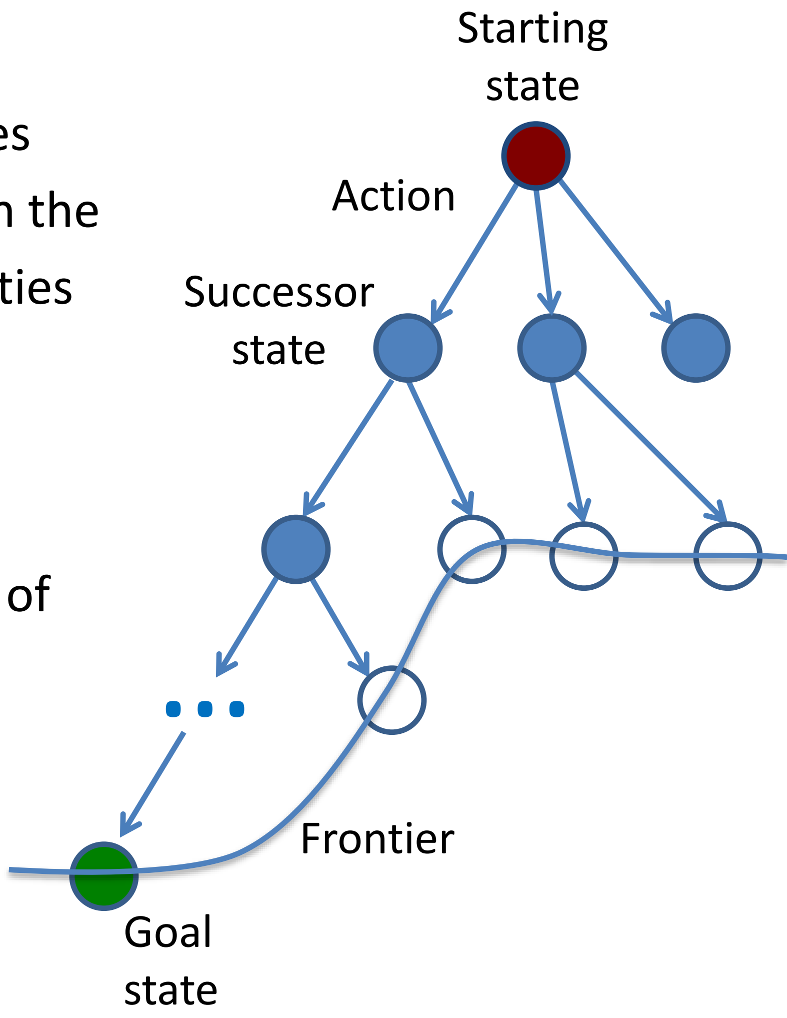
A path through the tree corresponds to a sequence of actions

- ❖ A solution is a path ending in the goal state

Nodes vs. **states**

A state is a representation of the world, while a node is a data structure that is part of the search tree

Node has to keep pointer to parent, path cost, possibly other info



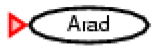
Tree Search Algorithm Outline

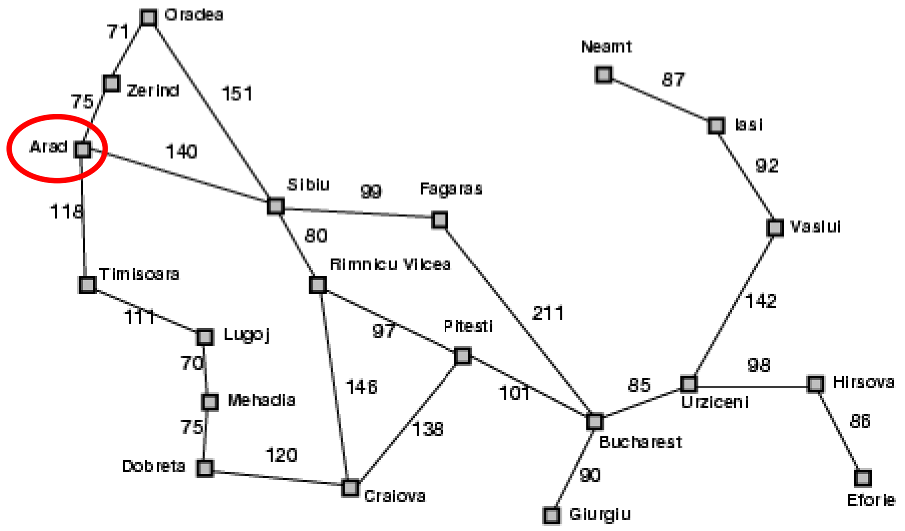
Initialize the **frontier** using the **starting state.**

While the frontier is not empty:

- Choose a frontier node according to **search strategy** and take it off the frontier.
- If the node contains the **goal state**, return solution.
- Else **expand** the node and add its children to the frontier.

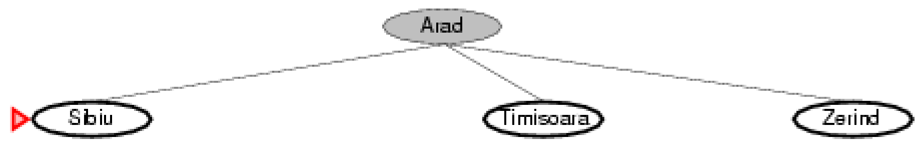
Tree Search Example

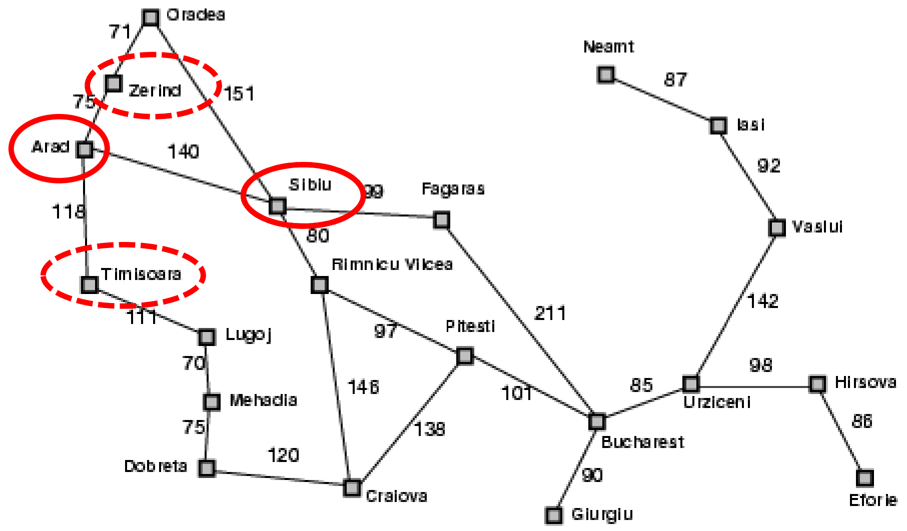




Start: Arad
Goal: Bucharest

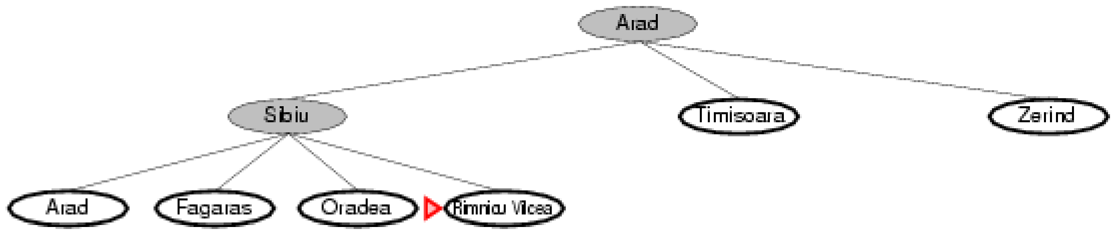
Tree Search Example



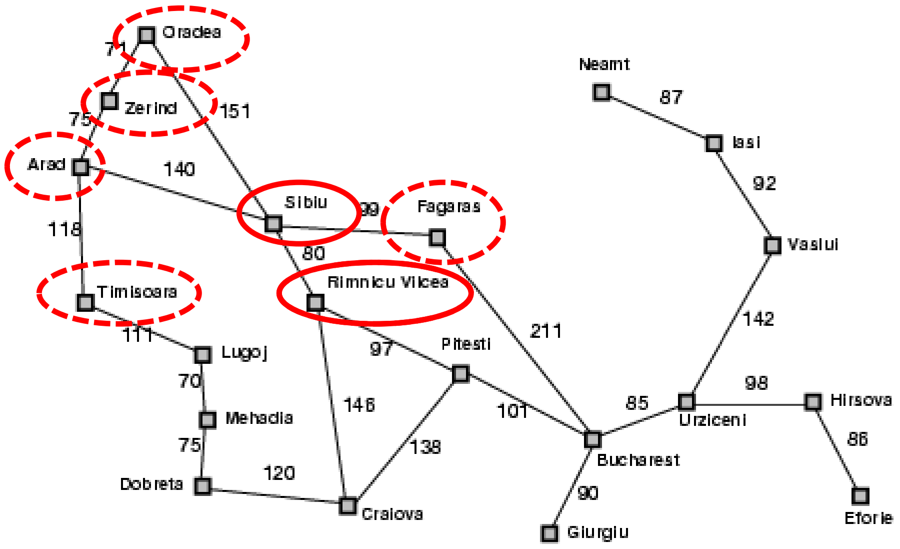


Start: Arad
Goal: Bucharest

Tree Search Example



Start: Arad
Goal: Bucharest



Handling Repeated States

Initialize the **frontier** using the **starting state**
While the frontier is not empty
    Choose a frontier node according to **search strategy** and take it off the frontier.
    If the node contains the **goal state**, return solution
    Else **expand** the node and add its children to the frontier.

To handle repeated states:
    Every time you expand a node, add that state to the **explored set**; do not put explored states on the frontier again.
    Every time you add a node to the frontier, check whether it already exists with a higher path cost, and if yes, replace that node with the new one.

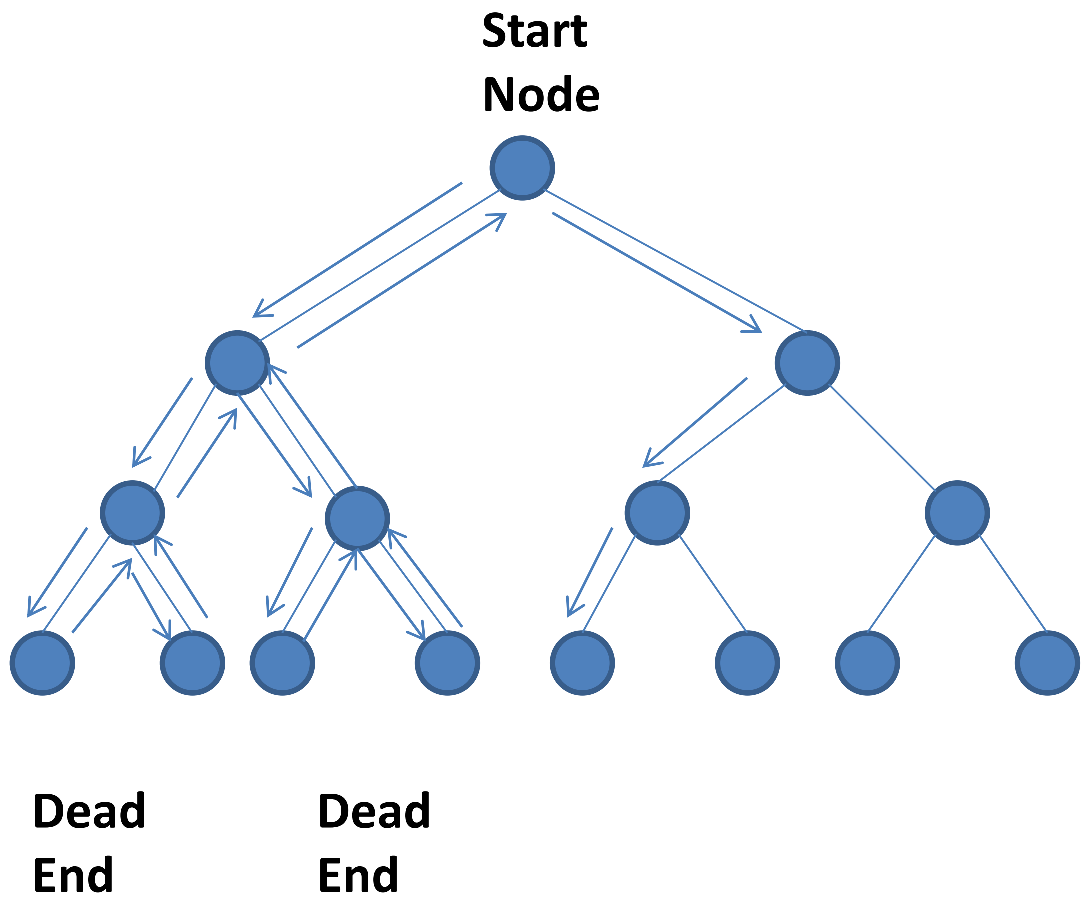
Backtracking Search

“Backtracking is a technique for systematically trying all paths through a state space”

It begins at the start state and pursues a path until:

- Finding a goal, then quit and return the solution path.
- Finding a dead end, then backtrack to the most recent unexamined node and continue down one of its branches.

Backtracking Search (Cont.)



Backtracking Algorithm Data Structure

- State List (SL): lists the states in the current path being tried. If the goal is found, then it contains the solution path.
- New State List (NSL): contains nodes a waiting evaluation.
- Dead Ends (DE): lists states whose descendants have failed to contain a goal node.
- Current State (CS): a state currently under consideration.

The Backtracking Algorithm

```
Function backtrack;
Begin
        SL:=[Start]; NSL:=[Start]; DE:= Start;
        while NSL# [ ] do
        begin
                if CS=goal(or meets goal description) then return(SL);
                if CS has no children(excluding nodes already on DE, SL, and NSL)
                        then begin
                                while SL is not empty and CS = the first element of SL do
                                        begin
                                          add CS to DE;
                                          remove first element from SL;
                                          remove first element from NSL;
                                           CS:= first element of NLS;
                                        end;
                                        add CS to SL;
                                end
                        else begin
                                place children of CS on NSL; (except nodes on DE, SL, or NSL)
                                CS:= first element of NSL;
                                add CS to SL;
                        end;
        end;
        return FAIL;
end;
```

Thanks! … *Questions?*

CS361 *(Artificial Intelligence)*

Lecture 3
Knowledge Representation via Propositional & Predicate Calculi

Dr. Hala Abdel-Galil & Dr. Amr S. Ghoneim
(Computer Science Dept.)

Helwan University
Fall 2019

Lectures are based on their counterparts in the following courses:
- *Intelligent Systems*, **University of British Columbia** (Dept. of Computer Science)
- *Introduction to Artificial Intelligence*, **University of Wisconsin-Madison**
- *Artificial Intelligence*, **University of Illinois at Urbana-Champaign**
- *Artificial Intelligence*, **University of California, Berkeley**

Resources for this lecture

- This lecture covers the following chapters:
  - Part II (pages 35 to 41) and Chapter 2 *from* George F. Luger, "Artificial Intelligence: Structures and strategies for complex problem solving, " Sixth edition (2009), Pearson Education Limited.

Outline

- AI Problem Solving Technique
  - Reminder … What is AI & Thinking Rationally
  - AI Representation Languages
    - Allow Representation of General Principles As Well As Specific Situations
    - Capture Complex Semantic Meaning
    - Allow for Meta-level Reasoning
- Propositional Calculus
  - Propositional Calculus Symbols
  - Propositional Calculus Sentences (WFFs)
    - Well Formed Formulas (WFFs)
  - Propositional Calculus Semantics
    - Truth Tables
    - Truth Tables Example
    - Common Identities
  - Limitations of propositional Calculus
- Predicate Calculus?
  - Predicate Calculus Symbols and Terms
  - Predicates and Atomic Sentences
    - Predicate Calculus Sentences
    - Predicate Calculus Sentences Examples
  - Predicate Calculus Semantics
    - Predicate Calculus Expressions Truth Value
    - English Sentences Represented in PC
    - Inference Rules
    - Common Inference Rules
  - Unification
    - Unification Rules
    - Unification Examples

Reminder … AI Problem Solving Technique

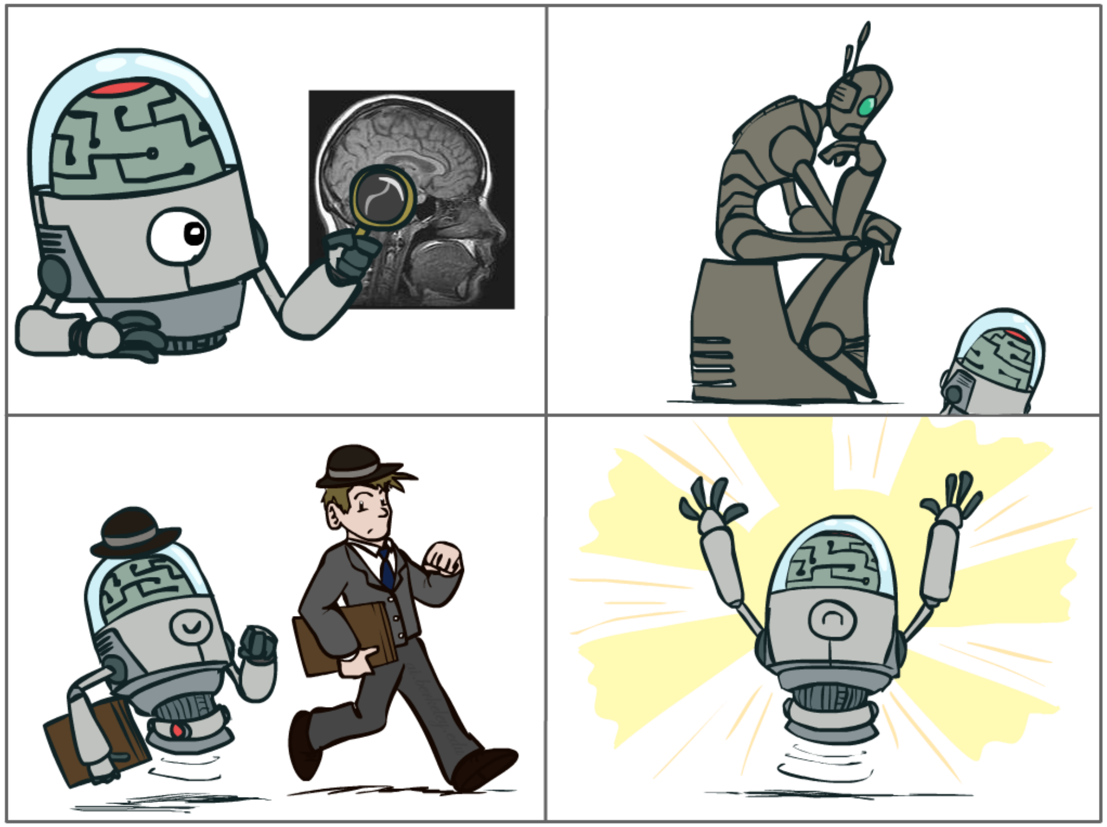
Intelligence activity is achieved through the following steps:

1. Formulate the symbol patterns to represent significant aspects of the problem domain.
2. Define the operations on these patterns to generate potential solutions to problems.
3. Searching among different possible solutions to select an acceptable solution.

Reminder . . . What is Artificial Intelligence?

Thinking humanly

Acting humanly



Thinking rationally

Acting rationally

Reminder . . . What is Artificial Intelligence?

Systems that act like humans	Systems that think rationally
"The study of how to make computers do things at which, at the moment, people are better" (Rich and Knight, 1991) "The art of creating machines that perform functions that require intelligence when performed by people." (Kurzweil, 1990)	"The study of mental faculties through the use of computational models" (Charniack and McDermott, 1985). "The study of the computations that make it possible to perceive, reason, and act." (Winston, 1992)
Systems that think like humans	**Systems that act rationally**
"The automation of activities that we associate with human thinking, such as decision making, problem solving, learning" (Bellman, 1978) "The exciting new effort to make computers think … machines with minds, in the full and literal sense." (Haugeland, 1985)	"AI .. is concerned with intelligent behavior in artifacts (Nilsson, 1998) "Computational Intelligence is the study of the design of intelligent agents." (Poole et al., 1998)

Reminder . . . Systems that Think Rationally

•***Logic*****:** formalize *idealized* or *right* thinking, i.e. irrefutable reasoning processes. That is; patterns of argument that always yield correct conclusions when supplied with correct premises:

"Socrates is a man; all men are mortal; therefore Socrates is mortal."

- Logistic tradition in AI aims to build computational frameworks based on logic, that is, describe problem in formal logical notation and apply general deduction procedures to solve it.
- Then use these frameworks to build intelligent systems.

- Some examples are (Propositional Logic) and (Logic Programming).
- More advanced logic-based representations:
  - Semantic Networks

Reminder . . . Systems that Think Rationally

- Main Research Problems/Challenges
  - Describing real-world problems and knowledge in logical notation.
  - Proving *Soundness* and *Completeness* of various formalisms.
  - How to represent often *informal* and *uncertain* domain knowledge and formalize it in logic notation *(i.e., dealing with Uncertainty)*.
  - Computational Complexity of finding a solution.
  - A lot of “rational” behavior has nothing to do with logic.

AI Representation Languages

Representation languages must be characterized by the following features:

- Handle qualitative knowledge.
- Allow new knowledge to be inferred from a basic set of facts.
- Allow representaion of general principles as well as specific situations.
- Capture complex semantic meaning.
- Allow for meta-level reasoning.

Allow Representation of General Principles As Well As Specific Situations

An intelligent system must be general as possible. Therefore, the variables are used in the representation language.

Good knowledge representation languages handle bindings of variable names, objects and values in higly dynamic fashion.

- For example, in a Blocks World, beside the given facts, we should be allowed to represent general principles, such as:

$$\forall x \sim\exists Y\ on(Y,x) \Rightarrow clear(x)$$

Capture Complex Semantic Meaning

Many intelligent domains require:

- Representing structured interrelated knowledge like taxonomic information.
- Described causal relationships between events occuring over time.

Some higher-level of structure is desirable to deal with complex concepts in a coherent fashion.

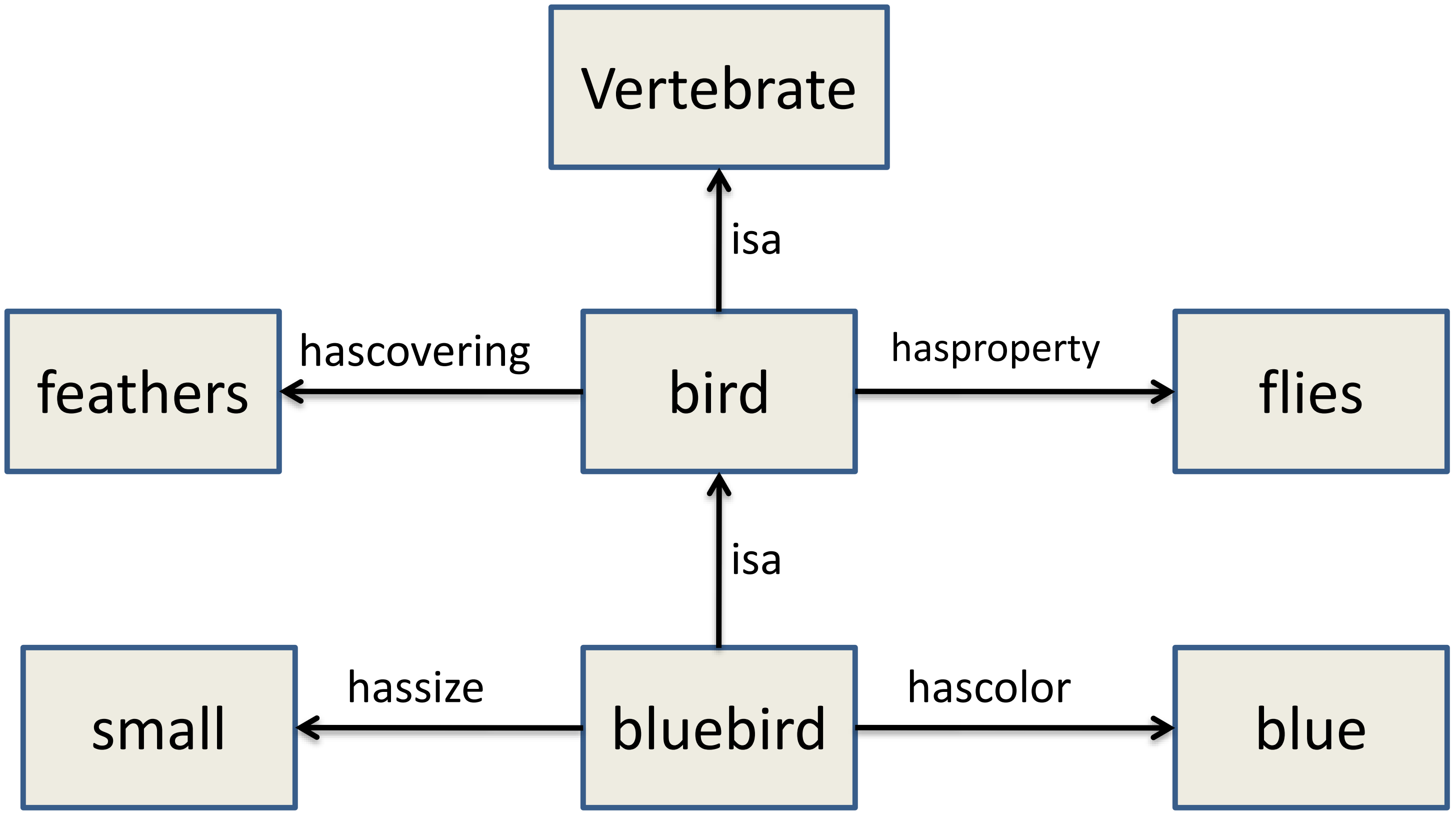
Capture Complex Semantic Meaning (Cont.)

“ A bluebird is a small blue-colored bird and a bird is a feathered flying vertebrate”

```
hassize(bluebird,small).
hascolor(bluebird,blue).
hascovering(bird,feathers).
hasproperty(bird,files).
isa(bluebird,bird).
isa(bird,vertebrate).
```

Predicate Calculus Representation

Capture Complex Semantic Meaning (Cont.)



Semantic network representation

Allow for Meta-level Reasoning

An intelligent system should know what it knows to be able:

- To explain how it solved the problems.
- To recognize its limitation of knowledge.
- To learn from its interactions with the world.

These level of knowledge *called meta-knowledge(knowledge about knowledge).*

The ability of capturing this type of knowledge and allowing meta-level reasoning must be achieved in the representation language.

Propositional Calculus

“A formal language that can be used in representing and reasoning about properties and relationships in the world”

Language Primitives:

- √ Symbols.
- √ Sentences.
- √ Semantics.

Propositional Calculus Symbols

“Propositional symbols denote propositions, or statements about the world that may be either true or false”

The symbols of propositional calculus are:

Propositional symbols: P, Q, R, S, T, …

Truth symbols: true, false.

Connectives: ^, V, ~, ⟹, =

Propositional Calculus Sentences (WFFs)

Sentences in Propositional Calculus are formed from Propositional Calculus symbols according to the following rules:

- Every Propositional or truth symbol is a sentence.
- The *negation* ($\sim$) of a sentence is a sentence.
- The *conjunction* ($\wedge$) of two sentences is a sentence.
- The *disjunction* ($\vee$) of two sentences is a sentence.
- The *implication* ($\Longrightarrow$) of one sentence for another is a sentence.
- The *equivalence* ($=$) of two sentences is a sentence.

Well Formed Formulas (WFFs)

“Only expressions that are formed of legal symbols through some sequence of sentence composition rules are propositional calculus sentences or WFFs.”

For Example: ((P^Q)⟹R) = ~P V ~Q V R

- P, Q and R are the propositions.
- P^Q is the conjunction.
- (P^Q)⟹R, is the implications.
- ~P and ~Q are the negations.
- ~P V ~Q is the disjunction.
- ~P V ~Q V R is the disjunction.
- ((P^Q)⟹R) = ~P V ~Q V R is the equivalence.

Propositional Calculus Semantics

A Proposition symbol corresponds to a statement about the world that may be either true or false, given some state of the world, this truth value assignment is called *interpretation.*

The *interpretation* is a mapping from the prpositional symbols into the set {T, F}.

Each possible mapping corresponds to a possible world of interpretation.

If P denotes “it is raining”, and Q denotes “I’m at work”, then the set of propositions {P,Q} has four different mappings into the truth values {T, F}, which means four possible interpretations.

Propositional Calculus Semantics (Cont.)

An *interpretation* of a set of propositions is the assignment of a truth value, either T or F, to each propositional symbol.

The symbol true is assigned T, and the symbol false is assigned F.

The interpretation or truth value for sentences, for example:

- The truth assignment of *negation*, ~P where P is any propositional symbol, is F if the assignment to P is T. And T if the assignment to P is F.

Propositional Calculus Semantics (Cont.)

- The truth assignment of *conjunction*, ^ is T only when both conjuncts have truth value T; otherwise it is F.
- The truth assignment of *disjunction*, v is F only when both disjuncts have truth value F; otherwise it is T.
- The truth assignment of *implication*,⟹, is F only when the premise or symbol before the implication is T and the truth value of the consequent or symbol after implication is F; otherwise it is T.
- The truth assignment of *equivalance,=,* is T only when both expressions have the same truth assignment for all possible

Truth Tables

Used to describe the truth assignments of compound propositions.

It lists all possible value assignments to the atomic propositions of an expression and gives the truth value of the expression for each assignment.

Two expressions in the propositional calculus are equivalant if they have the same value under all truth value assignments in their truth tables.

Truth Tables Example

$((P \wedge \sim Q) \vee R)$

P	Q	R	$\sim Q$	$P \wedge \sim Q$	$((P \wedge \sim Q) \vee R)$
T	T	T	F	F	T
T	T	F	F	F	F
T	F	T	T	T	T
T	F	F	T	T	T
F	T	T	F	F	T
F	T	F	F	F	F
F	F	T	T	F	T
F	F	F	T	F	F

Common Identities

- ~(~P) = P
- (P v Q) = (~P ⟹ Q)
- De Morgan’s law: ~(P v Q) = (~P ^ ~Q)
- De Morgan’s law: ~(P ^ Q) = (~P v ~Q)
- Distributive law: P v (Q ^ R) = (P v Q) ^ (P V R)
- Distributive law: P ^ (Q v R) = (P ^ Q) v (P ^ R)
- Commutative law: (P ^ Q) = (Q ^ P)
- Commutative law: (P v Q) = (Q v P)
- Associative law: ((P ^ Q) ^ R) = (P ^ (Q ^R))
- Associative law: ((P v Q) v R) = (P v (Q v R))
- Contrapositive law: (P => Q) = (~Q ⟹ ~P)

Limitations of propositional Calculus

Limited expressive power

- Does not break down the proposition into objects and their relationships.
- Does not allow generalization of these relationships over class of objects.

Predicate Calculus?

Predicate Calculus is superior to propositional calculus in two points:

- Predicate calculus can describe the relation between objects.
- Predicate calculus allows expressions to contain variables.

Predicate Calculus Symbols and Terms

Predicate caculus symbols include:
Truth symbols, true or false.
Constant symbols, which are symbol expressions having the first character lowercase.
Variable symbols, which are symbol expressions beginning with an uppercase character.
Function symbols, which are symbol expressions having the first character lowercase and attached arity indicating the number of elements of the domain mapped onto each element of the range.
A function expression, consists of a function constant of arity n, followed by n terms, t1, t2, …, tn enclosed in parentheses and seperated by commas.

Predicates and Atomic Sentences

- Predicate symbols are symbols beginning with a lowercase letter.
- Predicates have an associated possitive integer referred to as the arity, number of arguments, for the predicate.
- Predicates with the same name but different arities are considered distinct.
- An atomic sentence is a predicate constant of arity *n*, followed by *n* terms enclosed in parentheses and seperated by commas. Atomic sentences are delimited by a period.

- The truth values, true and false, are also atomic sentences.

Predicate Calculus Sentences

1. Every atomic sentence is a sentence
2. If S is a sentence, then so is its negation, ~S.
3. If S1 and S2 are sentences, then so is their conjunction, S1 ^ S2.
4. IF S1 and S2 are sentences, then so is their disjunction, S1 v S2.
6. If S1 and S2 are sentences, then so their implication, S1$\Longrightarrow$ S2. IF S1 and S2 are sentences, then so their equivalence, S1= S2.
7. If X is a variable and S a sentence. Then ∀X S is a sentence.
8. IF X is a variable and S a sentence, then ∃X S is a sentence.

Predicate Calculus Sentences Examples

- mother(eve,abel).
- mother(eve,cain).
- father(adam,abel).
- father(adam,cain).
- $\forall X\ \forall Y$ father(X,Y) v mother(X,Y) $\Longrightarrow$ parent(X,Y).
- $\forall X\ \forall Y\ \forall Z$ parent(X,Y) ^ parent(X,Z) $\Longrightarrow$ sibling(Y,Z).

Predicate Calculus Semantics

Let the domain D be a nonempty set.

An *interpretation* over D is an assignment of the entities of D to each of the constant, variable, predicate, and function symbols of a predicate calculus expression, such that:

1. Each constant is assigned as element of D.
2. Each variable is assigned to a nonempty subset of D; these are the allowable substitutions for that variable.
3. Each function f of arity m is defined on m arguments of D and defines a mapping from Dm into D.
4. Each predicate p of arity n is defined on n arguments from D and defines a mapping from Dn into {T,F}.

“Given an interpretation, the meaning of an expression is a truth value assignment over the interpretation”

Predicate Calculus Expressions Truth Value

Assume an expression E and an interpretation I for E over a nonempty domain D. The truth value for E is determined by:

1. The value of a constant.
2. The value of a variable.
3. The value of a function expression.
4. The value truth symbol “true” is T and “false” is F.
5. The value of an atomic sentence is either T or F.
6. Negation, Conjunction, Disjunction, Implication, and equivalence.
7. The value of $\forall X$ S is T if S is T for all assignments to X and it is F otherwise.
8. The value of $\exists X$ S is T if there is an assignment to X under which S is T; otherwise it is F.

English Sentences Represented in PC

- If it doesn’t rain tomorrow, Tom will go to the mountains.

  ~weather(rain, tomorrow) ⟹ go(tom, mountains).

- Emma is a Doberman pinscher and good dog.

  gooddog(emma) ^ isa(emma, doberman).

- All basketball players are tall.

  ∀X(basketball_palyer(X) ⟹tall(X)).

English Sentences Represented in PC (Cont.)

- If wishes were horses, beggars would ride.

  $equal(wishes, horse) \Longrightarrow ride(beggars).$

- Some people likes anchovies.

  $\exists X(person(X) \wedge likes(X, anchovies)).$

- Nobody likes taxes.

  $\sim\exists X\ likes(X,taxes).$

Inference Rules

“A mechanical means of producing new predicate calculus sentences from other sentences”

•The concept “logically follows” provides a formal basis for proofing the soundness and completeness of inference rules.

•If every sentences X produced by an inference rule logically follows from a set S of logical expressions, then the inference rule is *sound*.

•If the inference rule is able to produce every expressions that logically follows from S, then it is *complete*.

Common Inference Rules

Modus Ponens: If the sentences P and P $\Longrightarrow$ Q are true, then by modus ponens inference rule we can infer Q.

Modus Tolens: If the sentences (P $\Longrightarrow$ Q) is true and Q is known to be false then by modus tolens inference rule we can infer ~P.

Universal Instantiation: If any universal quantified variable in a true sentence is replaced by any appropriate term from the domain, the result is a true sentence. If a is from the domain of X, $\forall$X P(X) lets us infer P(a).

Unification

“ An algorithm for detemining the substitutions needed to make two predicate calculus expressions match”

- Unification and inference rules allow to make inferences on a set of logical assertions.
- Variables of the unified expressions are bounded to the values substituted for them by special rules.
- Unification requires that all variables be universally quantified. Existentially quantified variables are eliminated by replacing them with the constants that make the expression true.

Unification Rules

- A variable is substituted by unification to:
  - A constant and may not be replaced.
  - Another unbounded variable.
  - A term not containing the same variable.
- Two different constants can not be substituted for one variable.
- Once a variable has been bound future unification and inferences must take the value of this binding into account.
- The *composition* of two unification substitution sets S and S′ is obtained by applying S to the elements of S′ and adding the results to S.

Unification Examples

foo(X,a,goo(Y)).

- ❖foo(fred,a,goo(Z)).
- ❖foo(W,a,goo(jack)).
- ❖foo(Z,a,goo(moo(Z))).

$\Rightarrow$

{fred/X, Z/Y}
{W/X, jack/Y}
{Z/X,moo(Z)/Y}

- ∀X(man(X) ⟹ mortal(X))
  man(socrates).

$\Rightarrow$ mortal(socrates)

- {X/Y, W/Z},
  {V/X},
  {a/V, f(b)/W}

$\Rightarrow$ {a/Y, f(b)/Z}

Thanks! … *Questions?*