CS440/ECE448: Intro to Artificial Intelligence

Lecture 13: Review for midterm

Prof. Julia Hockenmaier
juliahmr@illinois.edu

http://cs.illinois.edu/fa11/cs440

Planning

Classical planning: assumptions

The environment is fully observable, deterministic, static, known and finite.

A plan is a linear sequence of actions;

Planning can be done off-line

Representations for planning: key questions

How do we represent states?

- What information do we need to know?
- What information can we (safely) ignore?

How do we represent actions?

- When can we perform an action?
- What changes when we perform an action?
- What stays the same?
- What level of detail do we care about?

Operators, actions and fluents

Operator: *carry(x)*
General knowledge of one kind of action: preconditions and effects

Action: *carry(BlockA)*
Ground instance of an operator

Fluent: *on(BlockA, BlockB, s)*
may be true in current state, but not after the action *move(A,B,T)* is performed.

Representations for operators

Operator name (and arity): move x from y to z

$move(x,y,z)$

Preconditions: when can the action be performed

$clear(x) \wedge clear(z) \wedge on(x,y)$

Effects: how does the world change?

$\underbrace{clear(y) \wedge on(x,z)}_{new} \wedge \underbrace{clear(x)}_{persist} \wedge \underbrace{\neg clear(z) \wedge \neg on(x,y)}_{retract}$

=> main differences between languages

Representations for states

We want to know what state the world is in:

– What are the current properties of the entities?

– What are the current relations between the entities?

Logic representation:

Each state is a conjunction of ground predicates:

$Block(A) \land Block(B) \land Block(C) \land Table(T)$
$\land On(A,B) \land On(B,T) \land On(C,T) \land Clear(A) \land Clear(C)$

Representations for planning

Situation Calculus	Strips
Specify fluents	Specify fluents
Add-set	*Add*-set
Persist-set	*Delete*-set
By default fluents are deleted	By default fluents persist

Planning algorithms

State space search (DFS, BFS, etc.)

Nodes = states; edges = actions;

Heuristics (make search more efficient)

Compute h() using relaxed version of the problem

Plan space search (refinement of partial plans)

Nodes = partial plans; edges: fix flaws in plan

SATplan (encode plan in propositional logic)

Solution = *true* variables in a model for the plan

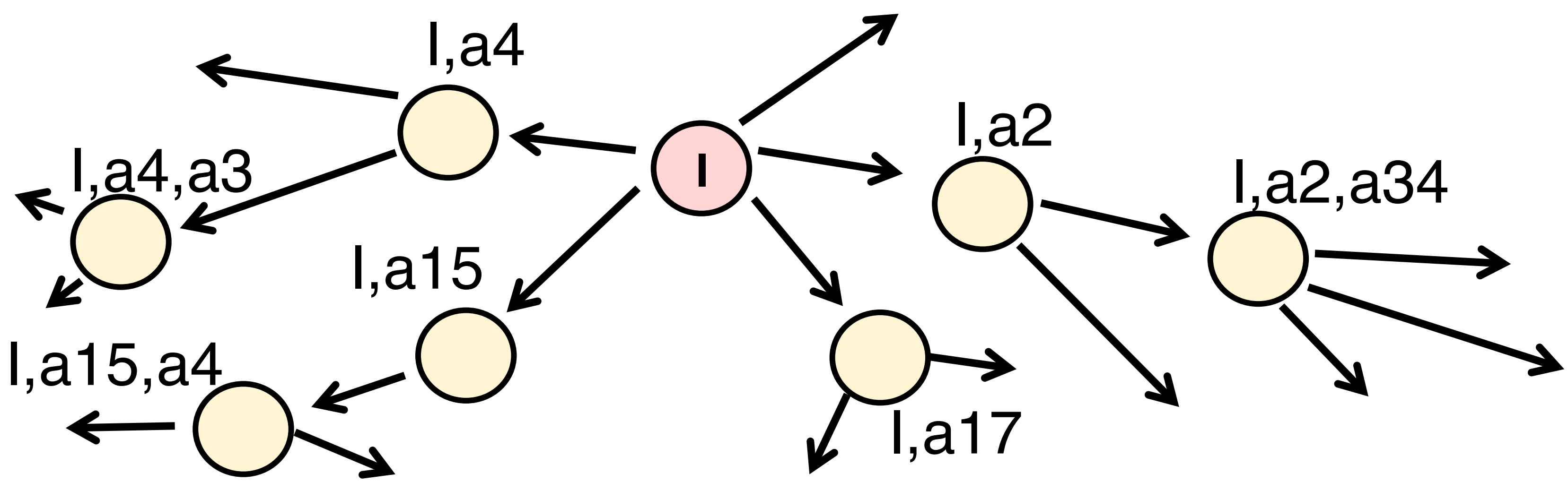
Graphplan (reduce search space to planning graph)

Planning graph: levels = literals and actions

Planning as state space search

Search tree:

– Nodes: states
– Root: initial state
– Edges: actions (ground instances of operators
– Solutions: paths from initial state to goal.



Searching plan-space

1. Start with the empty plan
   = *{start state, goal state}*

2. Iteratively refine current plan to resolve flaws
   (refine = add new actions and constraints)
   – Flaw type 1: open goals (require more actions)
   – Flaw type 2: threats (require ordering constraints)

3. Solution = a plan without flaws

SATplan

Represent a plan of fixed length *n*
as a ground formula in predicate logic.
Translate this formula into propositional logic.

There is a solution if this formula is satisfiable.

Plan = sequence of 'true' actions.

If there is no solution of length *n*, try *n+1*.

From plans to predicate logic

Fluents are ground literals: $clear(B, t)$

Actions are ground implications:

$(preconditions^t \wedge action^t) \rightarrow effect^{t+1}$

Action $move(A,B,C, 23)$

$(on(A,B, 23) \wedge clr(C, 23) \wedge move(A,B,C, 23))$

$\rightarrow (on(A,C, 24) \wedge clr(B, 24))$

From plans to propositional logic

Fluents

$clear(B, 23) \; = clear\text{-}B\text{-}23$

Actions

$$((on(A,B, 23) \land clear(C,23) \land move(A,B,C, 23)) \rightarrow (on(A,C, 24) \land clear(B, 24))$$

$$((on\text{-}A\text{-}B\text{-}23 \land clear\text{-}C\text{-}23) \land move\text{-}A\text{-}B\text{-}C\text{-}23)) \rightarrow (on\text{-}A\text{-}C\text{-}24 \land clear\text{-}B\text{-}24)$$

Intelligent agents

Key concepts

Agents:

- Different kinds of agents
- The structure and components of agents

Describing and evaluating agents:

- Performance measures
- Task environments

Rationality:

- What makes an agent intelligent?

Agents

1. What is the **task environment** of:
   - a chess computer?
   - a Mars rover?
   - a spam detector?

2. What is the advantage of model-based agents over reflex-based agents?

Task environments

The task environment specifies the problem that the agent has to solve.

It is defined by:

1. the objective **P**erformance measure
2. the external **E**nvironment
3. the agent's **A**ctuators
4. the agent's **S**ensors

Simple reflex agents

Action depends *only* on current percept.
Agent has no memory.

Last percept	**Action**
[Clean]	Right
[*cat*]	RUN!

May choose actions stochastically
to escape infinite loops.

Last percept	**Action**
[Clean]	Right (p=0.8) Left(p=0.2)

Model-based reflex agents

Agent has an **internal model**
of the current state of the world.
Examples: the agent's previous location; current locations of all objects it has seen;

Last percept	Last location	Action
[Clean]	Left of current	Right
[Clean]	Right of current	Left

Systematic search

Key concepts

Problem solving as search:

Solution = a finite sequence of actions

State graphs and search trees

Which one is bigger/better to search?

Systematic (blind) search algorithms

Breadth-first vs. depth-first; properties?

Systematic/blind search: assumptions

The environment is:

1. observable
   (Agent perceives all it needs to know)
2. known
   (Agent knows the effects of each action)
3. deterministic
   (Each action always has the same outcome)

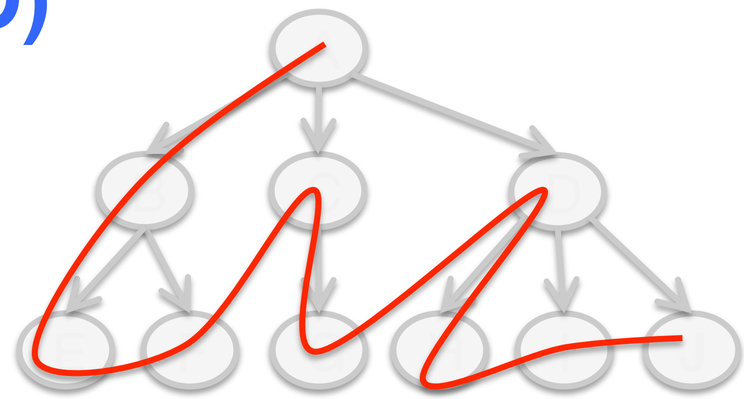
In such environments, the solution to any problem is a fixed sequence of actions.

The queuing function defines the search order

Depth-first search (LIFO)

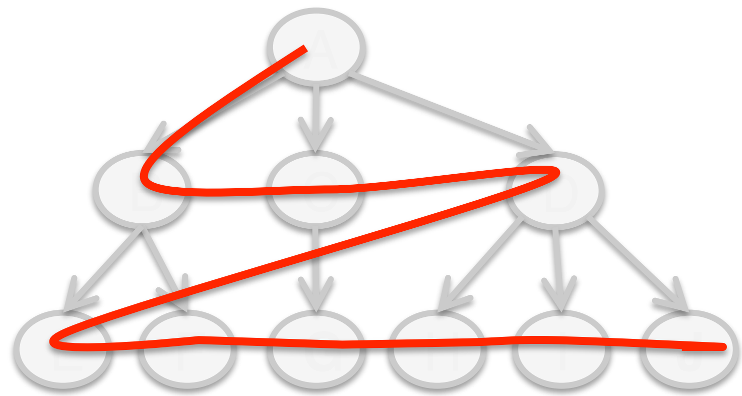
Expand deepest node first

```
QF(old, new):
    Append(new, old)
```



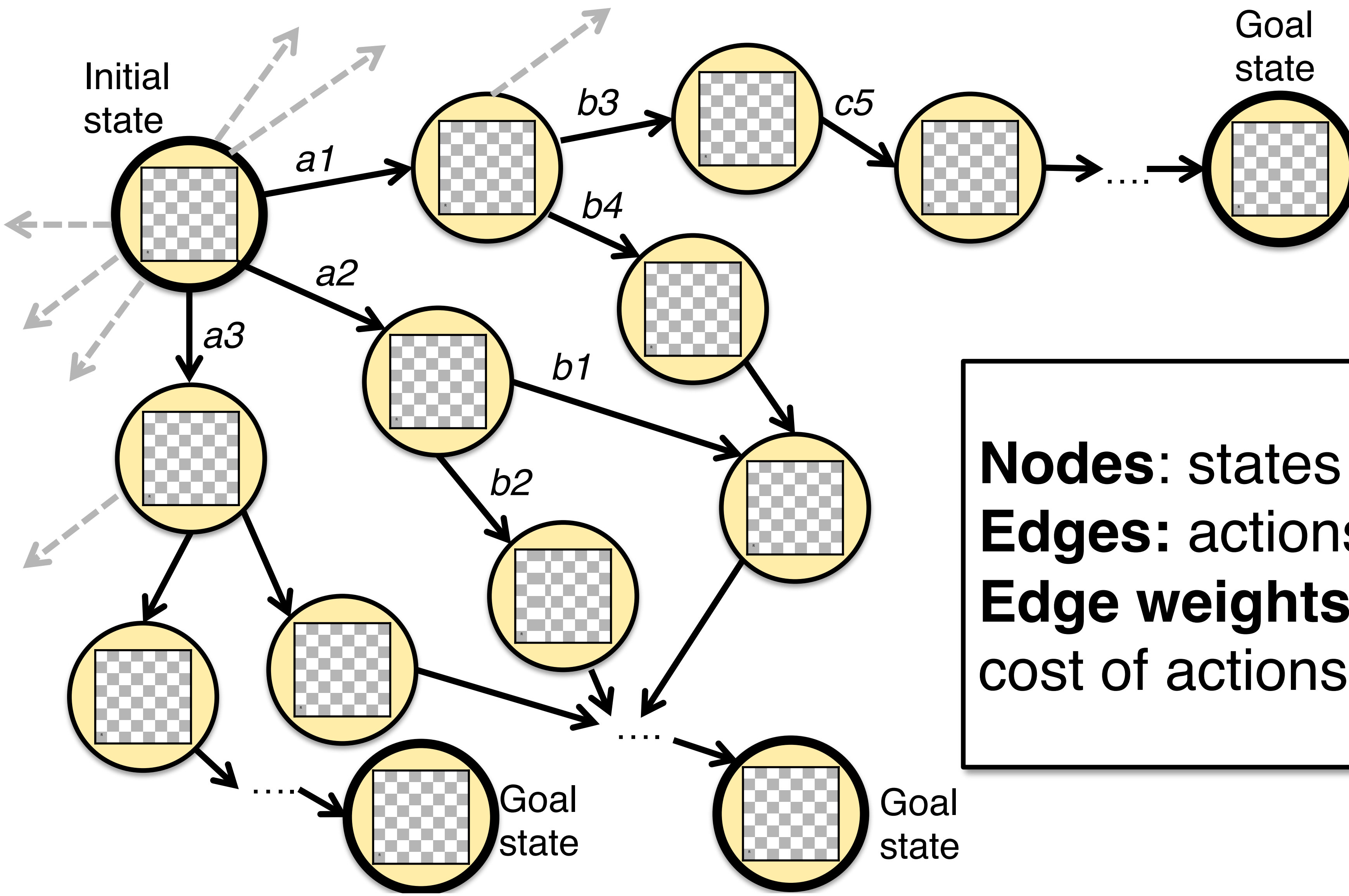
Breadth-first (FIFO)

Expand nodes level by level



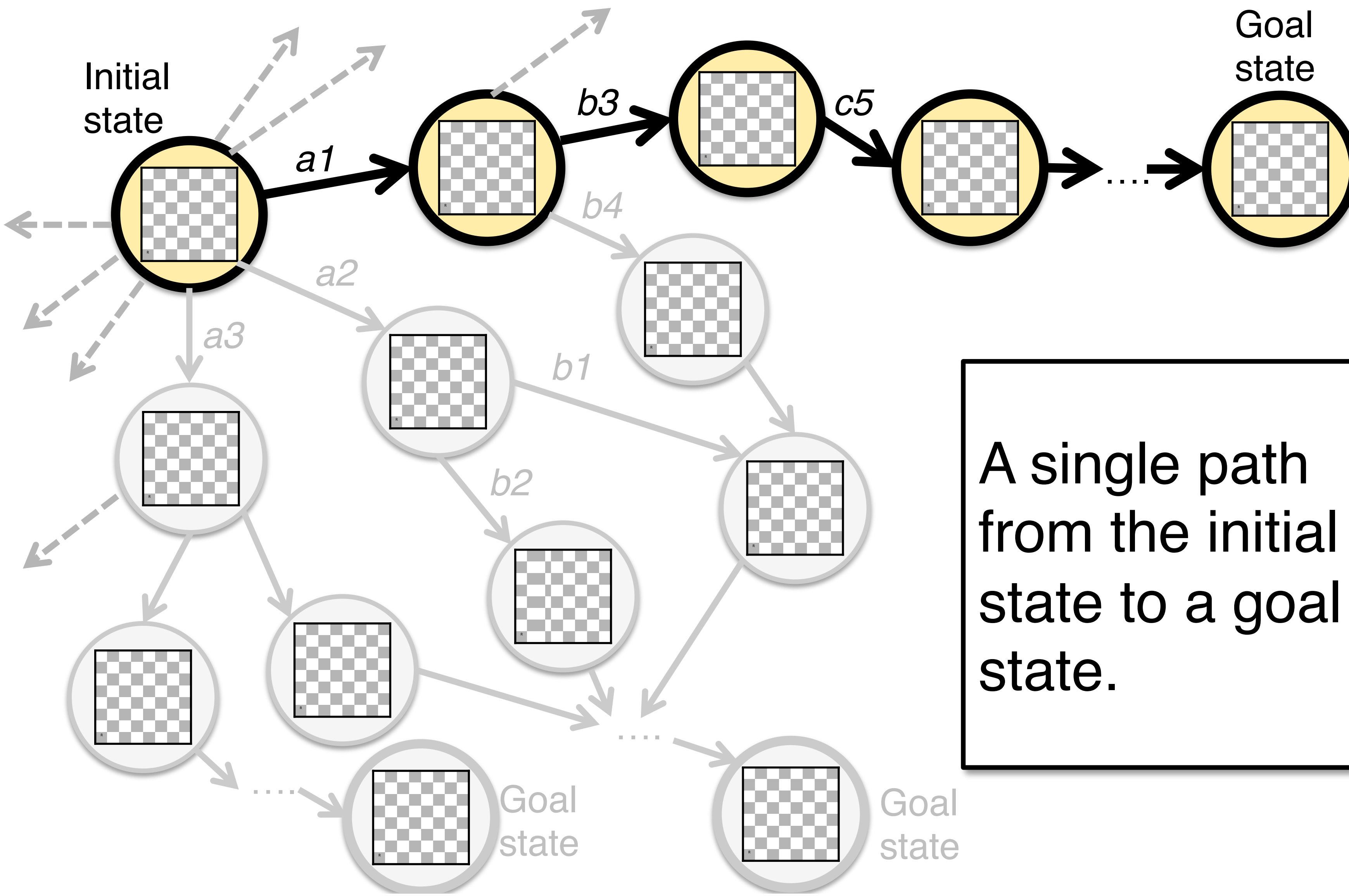
```
QF(old, new):
    Append(old, new);
```

State-space graph



Nodes: states
Edges: actions
Edge weights: cost of actions

Solution



A single path from the initial state to a goal state.

Properties of search algorithms

A search algorithm is **complete**
if it will find any goal whenever one exists.

A search algorithm is **optimal**
if it will find the cheapest goal.

Time complexity: how long does it take to find a solution?

Space complexity: how much memory does it take to find a solution?

Informed (heuristic) search

Informed search: key questions/concepts

How can we find the *optimal* solution?
We need to assign values to solutions

Values = cost.
We want to find the *cheapest* solution.

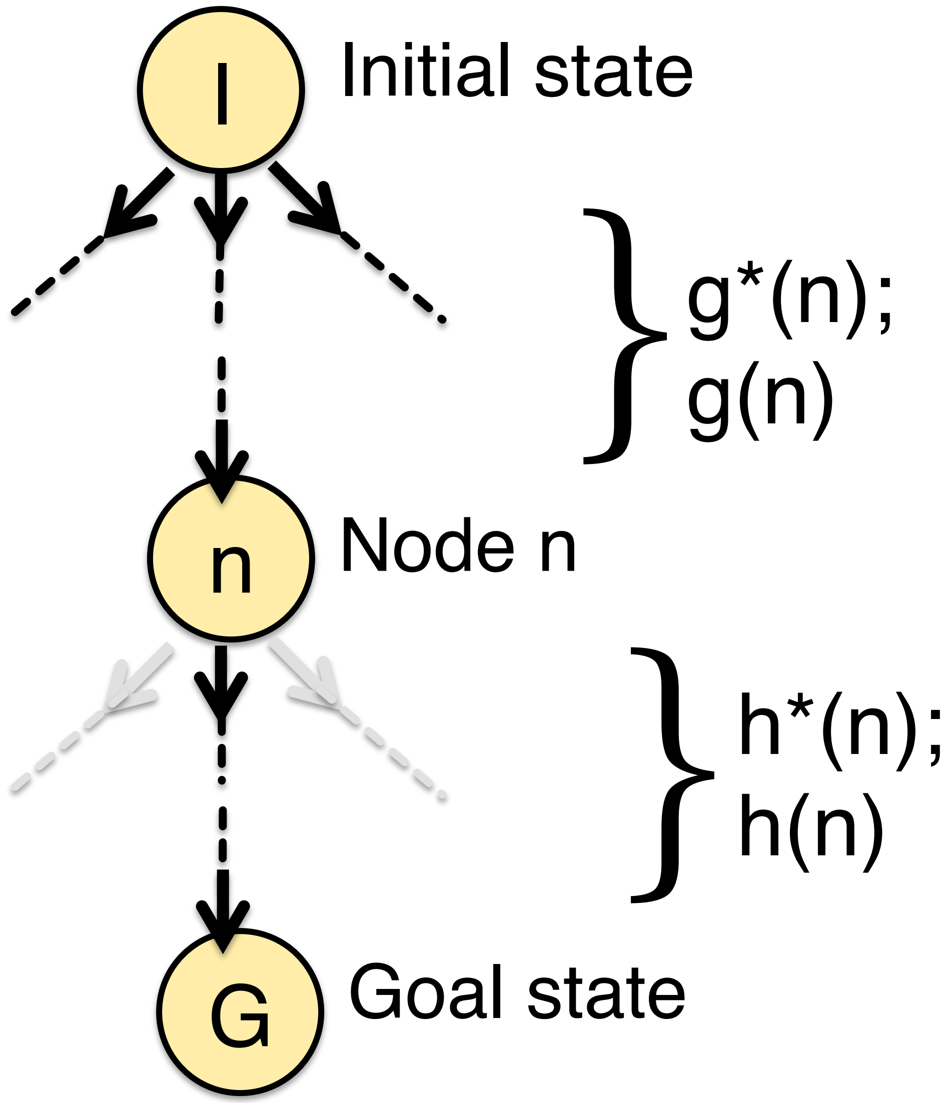
Heuristic search: priority queue

Heuristic search algorithms sort the nodes on the queue according to a **cost function**:

```
QF(a,b):
    sort(append(a,b), CostFunction)
```

The cost function is an estimate of the true cost. Nodes with the lowest estimated cost have the highest priority.

Cost functions



Heuristic search algorithms

	Uniform cost search	Greedy best-first search	A* search
Cost	g(n)	h(n)	f(n)
Optimal?	yes	no	if h(n) admissible
Complete?	with positive non-zero costs	graph-search: yes tree-search: no	with positive non-zero costs

Admissible heuristics

Task: Find the shortest path to X.

- **Heuristic 1:** $h(n) = 1000$.
- **Heuristic 2:** $h(n) = 0$.
- **Heuristic 3:** $h(n)$ = length of line from n to X.

Admissible heuristic:
never overestimate true cost.

Local search

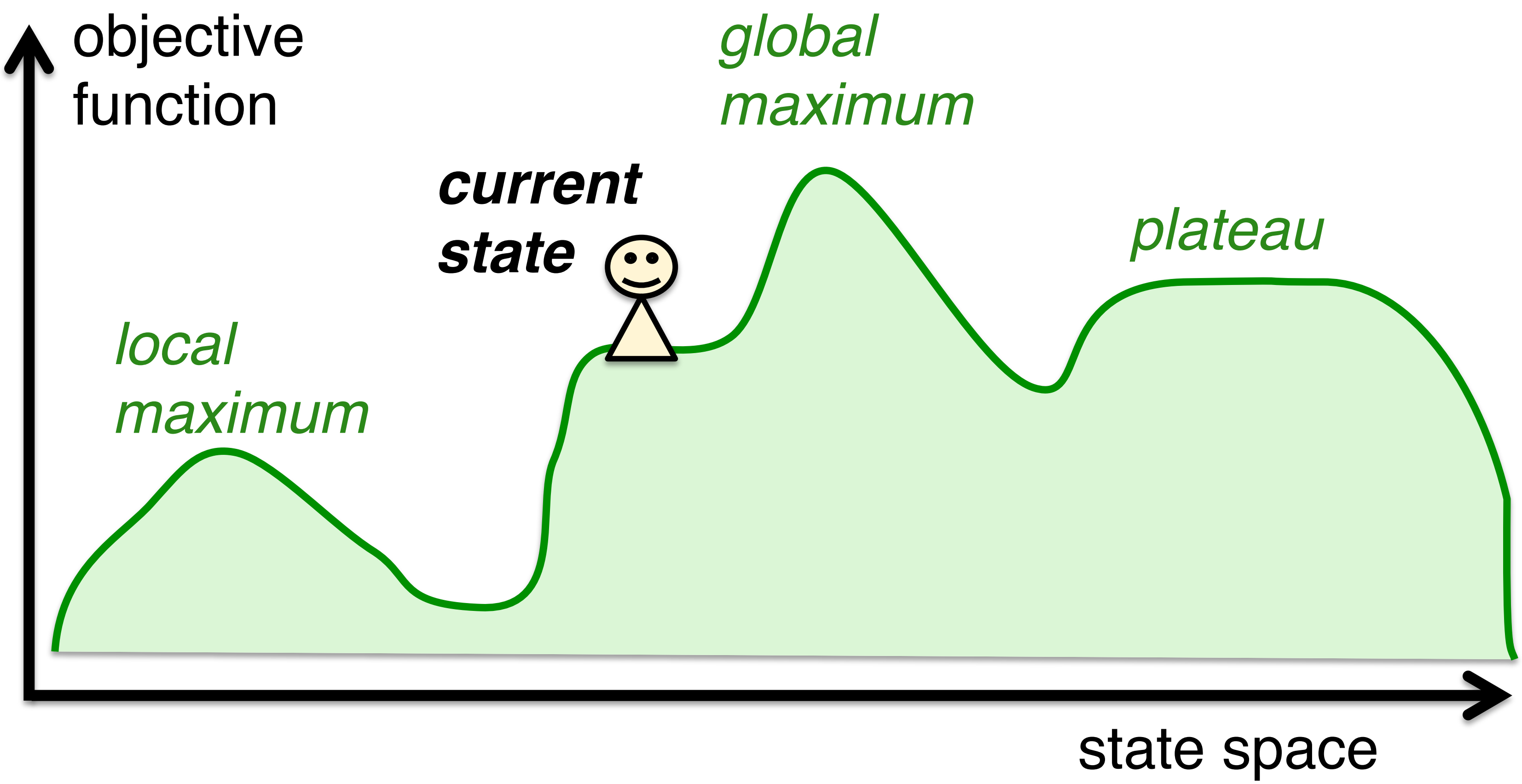
Local search

How can we find the goal when:

- we can’t keep a queue?
  (because we don’t have the memory)
- we don’t want to keep a queue?
  (because we just need to find the goal state,)
- we can’t enumerate the next actions?
  (because there’s an infinite number)

Local search = consider only the current node and the next action

The state space landscape



Dealing with local maxima

Random restart hill-climbing:
k trials starting from random positions

k-best (beam) hill climbing:
Pursue k trials in parallel; only keep the k best successors at each time step

Simulated annealing:
Accept downhill moves with non-zero prob.

Constraint satisfaction problems

Constraint satisfaction problems are defined by…

- **a set of variables** X:

  $\{WA, NT, QLD, NSW, VA, SA, TA\}$

- **a set of domains** D_i
  (possible values for variable x_i):

  $D_{WA} = \{red, blue, green\}$

- **a set of constraints** C:

  $\{\langle (WA,NT), WA \neq NT \rangle, \langle (WA,QLD), WA \neq QLD \rangle, \ldots\}$
  scope *relation*

Unary constraints: Node consistency

A unary constraint:
Western Australia is blue.
The final inspection takes 10 minutes

Expressed as constraint:
WA = blue; FI + 10 $\leq$ *5:00pm*

Expressed as restriction on the domain:
$D_{WA} = \{blue\}$, $D_{FI} = \{8{:}00am \ldots 4{:}50pm\}$

A single variable is node-consistent if all the values in its domain satisfy all its unary constraints.

Binary constraints: Arc consistency

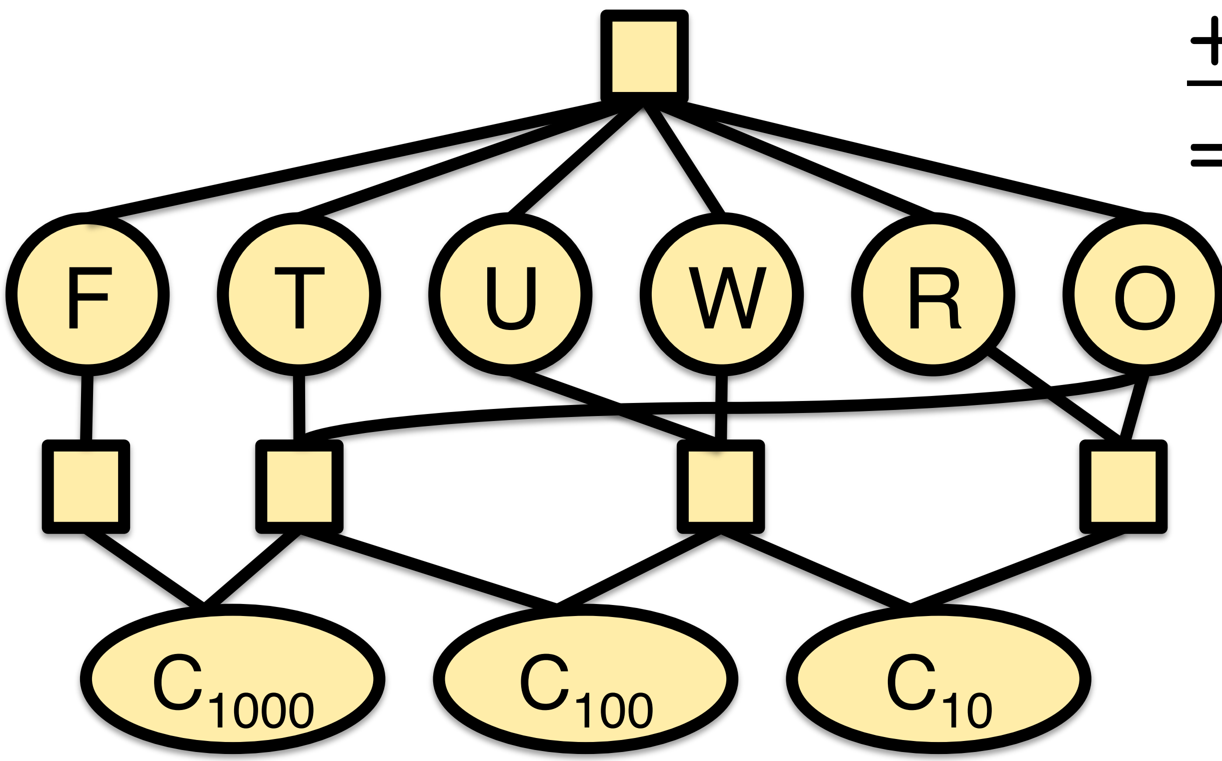
A variable x_i is **arc-consistent** if and only if for *every* value $d_i \in D_i$ in its own domain and for every binary constraint $C(x_i, x_j)$, there is a value $d_j \in D_j$ in x_j's domain such that C is satisfied.

$D_X = D_Y = \{0,1,2,3,4,5,6,7,8,9\}$,
Constraint: $C(X, Y)$: $Y = X^2$
Arc-consistency ➔ $D_X = \{0, 1, 2, 3\}$, $D_Y = \{0,1,4,9\}$

Interactions of binary constraints: Path consistency

A pair of variables {X,Y} is **path consistent with respect to variable Z** if and only if for every consistent $x \in D_X$ and $y \in D_X$ there is a $z \in D_Z$ that satisfies C(X=x,Z=z) and C(Y=y,Z=z)

Global (n-ary) constraints: Constraint Hypergraph




```
   TWO
+  TWO
= FOUR
```

Propositional logic

Propositional logic: key concepts

Syntax of propositional logic:

– propositional variables, connectives, well-formed formulas

Semantics of propositional logic:

– interpretations, models, truth tables

Inference with propositional logic:

– model-checking, resolution

Syntax: the building blocks

Variables: $\{p, q, r, \ldots\}$

Constants: $\top$ (true) , $\bot$(false)

Well-formed formulas:

WFF $\longrightarrow$ Atomic | Complex

Atomic $\longrightarrow$ Constant | Variable

WFF' $\longrightarrow$ Atomic | (Complex)

Complex $\longrightarrow$ $\neg$ WFF' | WFF' $\wedge$ WFF'

| WFF' $\vee$ WFF' | WFF' $\rightarrow$ WFF'

Semantics: truth values

The interpretation $[\![\alpha]\!]^v$ of a well-formed formula α under a model v is a truth value:

$$[\![\alpha]\!]^v \in \{true, false\}.$$

A model (=valuation) v is a complete* assignment of truth values to variables:

$$v(p) = true \quad v(q) = false, \; \ldots$$

*each variable is either true or false

With n variables, there are 2^n different models

Models of α ('$M(\alpha)$'): set of models where α is true

Validity and satisfiability

α is valid in a model m ('$m \models \alpha$') iff $m \in M(\alpha)$
= the model m satisfies α
(α is true in m)

α is valid ('$\models \alpha$') iff $\forall m: m \in M(\alpha)$
(α is true in all possible models. α is a tautology.)

α is satisfiable iff $\exists m: m \in M(\alpha)$
(α is true in at least one model, $M(\alpha) \neq \varnothing$)

Entailment

Definition:

α entails β (‘$\alpha \vDash \beta$’) *iff* $M(\alpha) \subseteq M(\beta)$

Entailment is monotonic:

If $\alpha \vDash \beta$, then $\alpha \wedge \gamma \vDash \beta$ for any γ

Proof: $M(\alpha \wedge \gamma) \subseteq M(\alpha) \subseteq M(\beta)$

We also write $\alpha, \gamma \vDash \beta$ or $\{\alpha, \gamma\} \vDash \beta$ for $\alpha \wedge \gamma \vDash \beta$

Inference in propositional logic

How do we know whether α is valid or satisfiable given KB?

Model checking: (semantic inference)
Enumerate all models for KB and α.

Theorem proving: (syntactic inference)
Use inference rules to derive α from KB.

Inference: soundness and completeness

A procedure P that derives α from KB…

$$KB \vdash_P \alpha$$

…is **sound** if it only derives valid sentences:

if $KB \vdash_P \alpha$, then $KB \models \alpha$ (soundness)

…is **complete** if it derives any valid sentence:

if $KB \models \alpha$, then $KB \vdash_P \alpha$ (completeness)

The resolution rule

Unit resolution:

$$\frac{p_1 \vee \ldots \vee p_{i-1} \vee p_i \vee p_{i+1} \vee \ldots \vee p_n \qquad \neg p_i}{p_1 \vee \ldots \vee p_{i-1} \vee p_{i+1} \vee \ldots \vee p_n}$$

Full resolution:

$$\frac{\mathbf{p_1} \vee \ldots \vee \mathbf{p_i} \vee \ldots \vee \mathbf{p_n} \qquad \mathbf{q_1} \vee \ldots \vee \neg \mathbf{p_i} \vee \ldots \vee \mathbf{q_m}}{\mathbf{p_1} \vee \ldots \vee \mathbf{p_n} \vee \mathbf{q_1} \vee \ldots \vee \mathbf{q_m}}$$

Final step: factoring (remove any duplicate literals from the result $A \vee A \equiv A$)

A resolution algorithm

Goal: prove $\alpha \models \beta$' by showing that $\alpha \wedge \neg \beta$ is not satisfiable (*false*)

Observation:
Resolution derives a contradiction (*false*) if it derives the empty clause:

$$\frac{\mathbf{p_i} \qquad \neg\, \mathbf{p_i}}{\varnothing}$$

First-order predicate logic

Richer models

Domain: a set of entities *{ann, peter, ...,book1,...}*

Entities can have **properties**.
A property defines a (sub)set of entities:
blue= {book5, mug3, ...}

There may be **relations** between entities.
An *n*-ary relation defines a set of *n*-ary tuples of entities: *belongsTo ={<book5, peter>, <mug3, ann>,...}*

A richer language

Terms: refer to entities
– Variables, constants, functions,

Predicates: refer to properties of entities or relations between entities

Sentences (= propositions)
can be true or false

The building blocks

A (finite) set of **variables** VAR:

$$VAR = \{x, y, z, \ldots\}$$

A (finite) set of **constants** $CONST$:

$$CONST = \{john, mary, tom, \ldots\}$$

For $n=1\ldots N$:

A (finite) set of n-place **function** symbols $FUNC$

$$FUNC_1 = \{fatherOf, \; successor, \ldots\}$$

A (finite) set of n-place **predicate** symbols $PRED_n$:

$$PRED_1 = \{student, \; blue, \ldots\}$$

$$PRED_2 = \{friend, \; sisterOf, \ldots\}$$

Clauses, literals, and CNF

Conjunctive normal form: a conjunction of clauses.

$(\varphi_1 \vee \ldots \vee \varphi_n) \wedge \ldots \wedge (\varphi'_1 \vee \ldots \vee \varphi'_m)$

Clause: a disjunction of an arbitrary number of positive or negative literals.

$(\varphi_1 \vee \neg\varphi_2 \vee \varphi_3 \vee \neg\varphi_4)$

Literal: a (negated) predicate and its arguments

likes(x, John')

¬likes(John', motherOf(Mary))

¬student(x)

Different kinds of clauses

Definite clause: exactly one positive literal
Facts: ψ
Implications: $\neg\varphi_1 \vee \ldots \vee \neg\varphi_n \vee \psi \quad \equiv [(\varphi_1 \wedge \ldots \wedge \varphi_n) \rightarrow \psi]$

Horn clause: at most one positive literal.
All definite clauses, plus (resolution) queries: $\neg\psi$

Clauses: arbitrary number of positive literals
All Horn clauses, plus any disjunction, e.g $\varphi \vee \psi$

Model $M=(D,I)$

The **domain** D is a nonempty set of objects:

$$D = \{a1, b4, c8, \ldots\}$$

The **interpretation function** I maps:

- each constant c to an element c^I of D: $John^I = a1$
- each n-place function symbol f to an (total) n-ary function $f^I\ D^n \rightarrow D$: $fatherOf^I(a1) = b4$
- each n-place predicate symbol p to an n-ary relation $p^I \subseteq D^n$: $child^I = \{a1, c8\}$ $likes^I = \{\langle a1, b4\rangle, \langle b4, a1\rangle\}$

Interpretation of variables

A variable assignment v over a domain D is a (partial) function from variables to D.

The assignment $v = [a21/x, b13/y]$ assigns object $a21$ to the variable x, and object $b13$ to variable y.

We recursively manipulate variable assignments when interpreting quantified formulas.
Notation: $v[b/z]$ is just like v, but it also maps z to b.
We will make sure that v is undefined for z.

Inference in FOPL

Challenge:
How do we deal with quantifiers and variables?

Solution 1: **Propositionalization**
Ground all the variables.

Solution 2: **Lifted inference**
Convert to prenex NF, then skolemize existentially quantified variables, use unification for universally quantified variables.

Propositionalization

If we have a finite domain, we can use UI and EI to eliminate all variables.

KB: $\forall x\ [(student(x) \wedge inClass(x)) \longrightarrow sleep(x)]$

$student(Mary)$ $\quad$ $professor(Julia)$

Propositionalized KB:

$(student(Mary) \wedge inClass(Mary)) \longrightarrow sleep(Mary)$

$(student(Julia) \wedge inClass(Julia)) \longrightarrow sleep(Julia)$

etc.

…this is not very efficient....

Prenex normal form

Every well-formed formula in FOL has an equivalent **prenex normal form,** in which all the quantifiers are at the front.

$\forall x \exists y \exists z \forall w \, \varphi(x,y,z,w)$

$\varphi(x,y,z,w)$ (the ‘matrix’) does not contain any quantifiers.

Skolemization: remove existentially quantified variables

Replace any existentially quantified variable $\exists x$ that is in the scope of universally quantified variables $\forall y_1 \ldots \forall y_n$ with a new function $F(y_1,\ldots,y_n)$ (a **Skolem function**)

Replace any existentially quantified variable $\exists x$ that is not in the scope of any universally quantified variables with a new constant c (a **Skolem term**)

Unification, MGU

A set of sentences $\varphi_{1,}\ldots\varphi_n$ unify to σ if for all $i \neq j$: $\mathrm{S{\scriptsize UBST}}(\sigma,\varphi_i) = \mathrm{S{\scriptsize UBST}}\,(\sigma, \varphi_j)$.

σ is the most general unifier (MGU) of φ and ψ if it imposes the fewest constraints.

$\mathrm{MGU}(P(x,y,z), P(w,w,Fred)) = \sigma = \{x=w, y=w, z=Fred\}$

yields $P(w,w,Fred)$

Equivalently, $\sigma = \{x=u, y=u, w=u, z=Fred\}$ yields the alphabetic variant $P(u,u,Fred)$

Inference in FOL: using Modus ponens

$$\frac{\begin{array}{c}\varphi \longrightarrow \psi \\ \varphi\end{array}}{\psi}\ (\mathrm{MP})$$

Forward: I know that φ implies ψ. I also know φ. Hence, I can conclude that ψ is true as well.

Backward: I want to know whether ψ is true. I know that φ implies ψ. Hence, if I can prove φ, I can conclude that ψ is true as well.