CS4246/CS5446 AI Planing and Decision Making

Classical Planning

Lee Wee Sun
School of Computing
National University of Singapore
leews@comp.nus.edu.sg

Semester 1, 2020/21

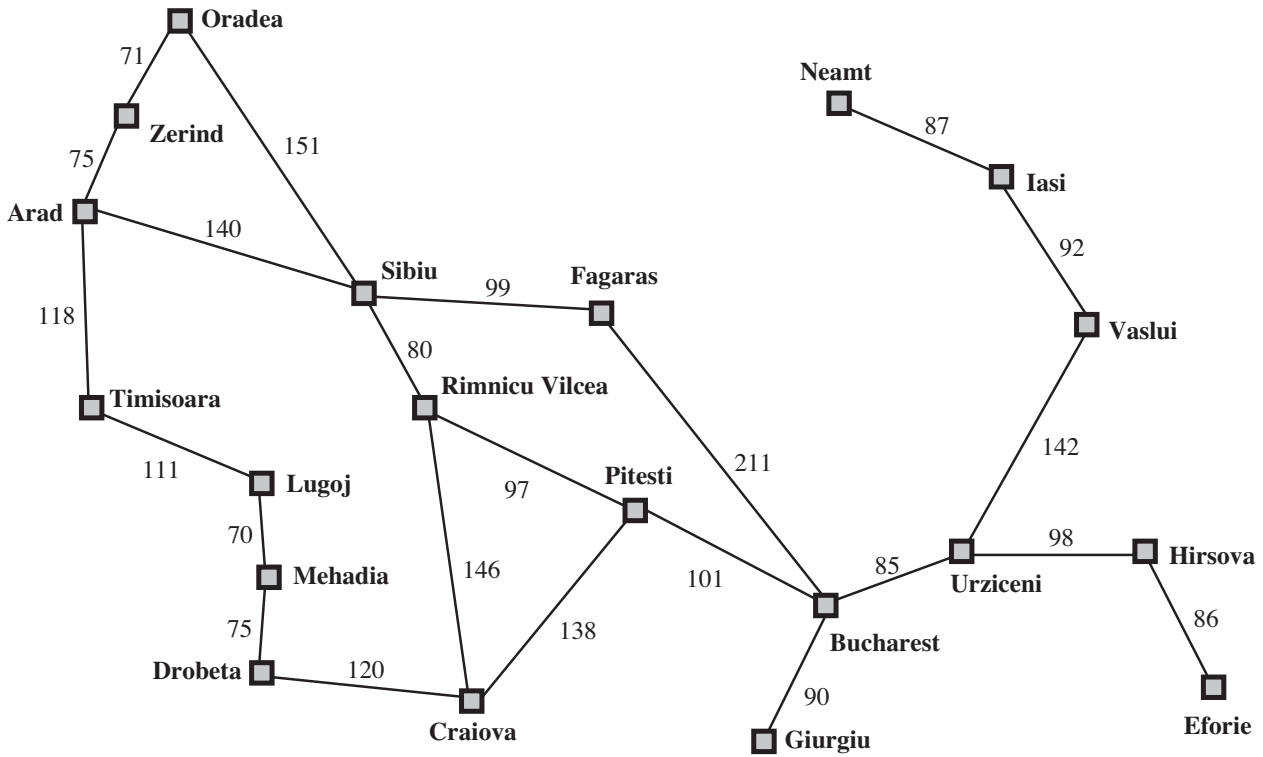
Classical Planning

- When we have to make a sequence of decisions, we call that planning.
- In this section, we consider environments that are fully observable, deterministic, finite and discrete: **classical planning**.

Outline

Problem Solving (Revision)

Recall the problem solving agent from AIAMA Chapter 3.



Navigation in Romania as example.

Exercise:

What is the fastest runtime known for solving the single source shortest path problem for a graph with V vertices and E edges with non-negative weights?

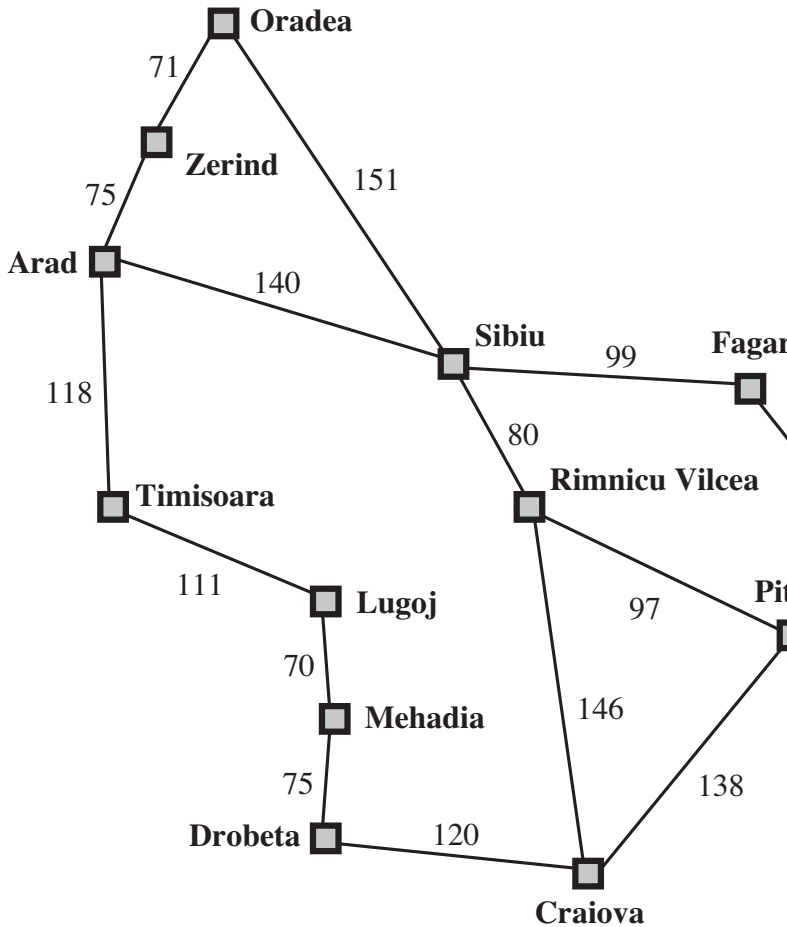
A. $O(V + E)$
B. $O(V^2)$
C. $O(E \log V)$
D. $O(V \log V + E)$

A **problem** is defined by 5 components:

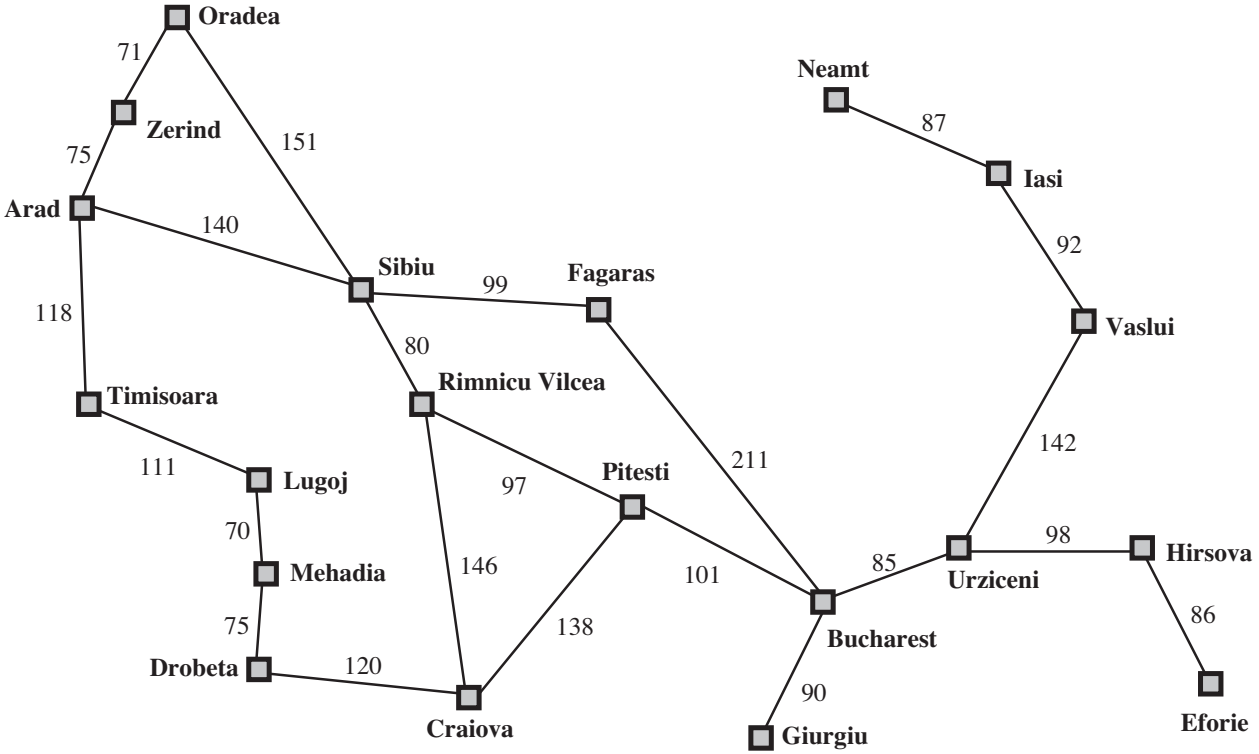
1. **Initial state**.

2. Possible **actions**. ACTION(s) returns set of actions that can be executed in s

3. **Transition model**. RESULT(s,a) returns state that results from doing a in s.



The set of all states reachable from the initial state is called the **state space**. Forms a directed graph.



Continued ...

4. A **goal test**.
5. A **path cost**. Often this is the sum of **step cost**, $c(s, a, s')$.

When the number of states is not too large, can solve using shortest path algorithms, such as UNIFORM-COST-SEARCH (Dijkstra's algorithm).

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the lowest-cost node in frontier */
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        frontier ← INSERT(child, frontier)
      else if child.STATE is in frontier with higher PATH-COST then
        replace that frontier node with child
```

A* Search

- **A* search** is the same as UNIFORM-COST-SEARCH except that it uses $f(n) = g(n) + h(n)$ as the cost in the priority queue.
  - $g(n)$ is the cost to reach node n
  - $h(n)$, called a **heuristic** is the estimated cheapest cost from n to the goal.

Admissible Heuristic

- A* is optimal if
  - $h(n)$ is an **admissible heuristic**, i.e. it never overestimates the cost to the goal (optimistic).
    - The straight line distance from n to the goal is an admissible heuristic for the navigation problem.

- Continued ...
  - A stronger condition **consistency** or **monotonicity** is required for graph search (to avoid repeating the nodes): $h(n)$ is consistent if

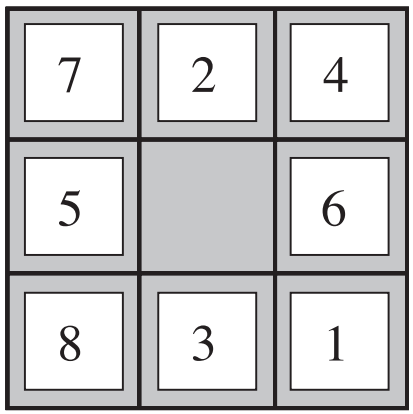
$$h(n) \leq c(n, a, n') + h(n')$$

where $c(n, a, n')$ is the cost for going from n to n' using a.

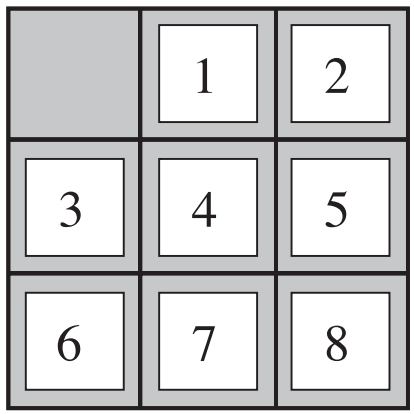
Relaxation

Admissible heuristics often comes from a **relaxed problem**: a problem with fewer restrictions on the actions.

- A straight line distance is not restricted to travelling on roads.



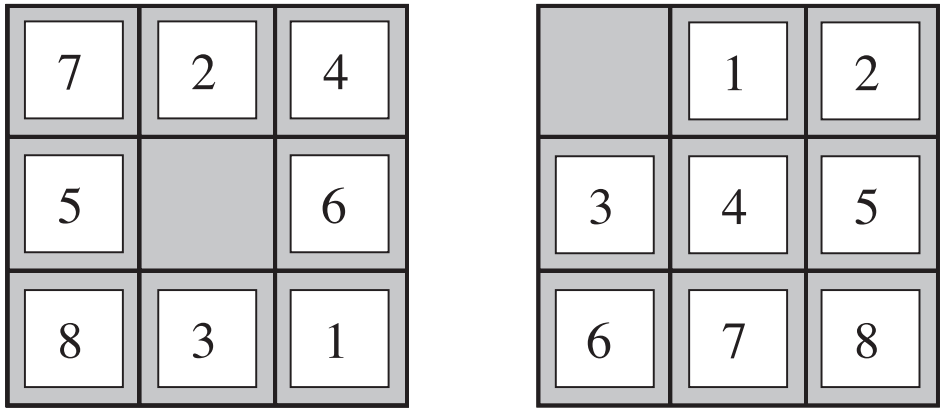
Start State



Goal State

Admissible heuristics for 8 puzzle:

- h_1: number of misplaced tiles



- h_2: sum of distances to goal for each tile, where distance is sum of horizontal and vertical distances.

- In this case, $h_2(n) \geq h_1(n)$, i.e. h_2 **dominates** h_1.
- If h_2 dominates h_1, h_2 will always explore fewer nodes with A*.
- In general, max over multiple admissible heuristic is admissible and dominates the individual heuristics.

Exercise:

A. Show how to use the minimum spanning tree (MST) problem to construct an admissible heuristic for the traveling salesman problemn (TSP).

Exercise: Consider the following three admissible heuristics for the traveling salesman problem.

MST Minimum spanning tree through the current node, target node, and unvisited nodes

SP Shortest path from current node to target node through subgraph consisting of current node, target node, and all unvisited nodes

SL Straight line distance from current node to target node

Which of the following is true?

A. MST dominates SP which dominates SL

B. SL dominates SP which dominates MST

C. None of the heuristic dominates any other heuristic

Large State Space

- When state space defined in terms of state variables, **factored representation**, it is typically exponentially large,
  - E.g. the state space for n puzzle, where we have n instead of 8 tiles grows exponentially with n
- Run out of memory if store all states, e.g. in A* algorithm.
- In such cases, **depth-first search** type algorithm often used.
  - $O(m)$ memory used, where m is depth of search
  - Optimal solution, or failure, when whole space has been exhausted (may be exponential time)
  - When depth-first path may be very long, use **iterative deepening search**
    - With heuristic function, use **iterative deepening A*** (IDA*).

Domain Independent Heuristics

- Without good heuristics, planning can quickly become very difficult as the problem size grows.
- Deriving heuristics for each individual problem can be time consuming.
- It turns out that there are domain independent heuristics that tend to work well for certain representations of planning problems.
- We will look at specifying planning problems using
  - STRIPS
  - Satisfiablity
  - Planning graph (CS5446)

  and some of the domain independent heuristics that are useful for the representations.

Outline

A^*, Shakey, and STRIPS

Video from `https://media.ed.ac.uk/media/Artificial+Intelligence+Planning+-+Nils+Nilsson+-+A-Star+and+STRIPS/1_uhxvxo4a/74328231`.

Factored Representation

- To scale up planning, a **factored representation** is usually used.
  - Instead of **atomic** representation of state (e.g. node in a graph), a state is represented using a set of attribute/value (variable/value) pairs.

- In planning problems, state variables (attributes that can change through time) are often referred to as **fluents**.

Logic Notation

We will use some notations and concepts from first order logic.

- A term refers to an object. It can be a variable, a constant, or a function
- An **atomic sentence** or **atom** is a predicate symbol optionally followed by a parenthesized list of terms
- A **literal** is an atom or its negation
- A **ground literal** is a literal with no variables

- A **sentence** or **formula** is formed from atoms together with quantifiers ($\forall$, $\exists$), logical connectives, equality symbnol.
- Sentences can take values **true** or **false**.
- A **substitution** replaces variables by terms, e.g. $\{x/John, y/z\}$ replaces x with *John* and y with z.
- A **unifier** takes two sentences and returns a substitution that makes the sentences look identical, if one exists, e.g. $\text{UNIFY}(Brother(John,x), Brother(y,James))=\{x/James, y/John\}$

- For every pair of unifiable sentences, there is a **most general unifier** that is unique up to renaming and substitution of variables
  - Let σ is a most general unifier and θ is a unifier. After applying σ, we can find another unifier ω such that applying ω to the output of σ gives the same effect as applying θ.

STRIPS

- The **Stanford Research Institute Problem Solver** (STRIPS) is a highly influential planning language that uses a factored representation.
- STRIPS defines 4 things: initial state, actions available in a state, result of applying an action, the goal.

STRIPS State and Goal

- In STRIPS, a **state** is a represented by a **conjunction** of **positive** literals that are **ground** and **function-free**.
  - In STRIPS, literals are ground boolean variables, e.g. for a package delivery problem:
    - literals are variables like $At(Truck_1, Melbourne)$
    - a state is a conjunction of fluents like $At(Truck_1, Melbourne) \wedge At(Truck_2, Sydney)$.
  - **Database semantics** are used
    - Literals not mentioned are false because of closed world assumption
    - $Truck_1$ and $Truck_2$ are distinct because of unique names assumption

- Continued ...
  - The following are not allowed in the state description:
    - $At(x, y)$ because not grounded
    - $\neg Poor$ because it is a negation
    - $At(Father(Fred), Sydney)$ because STRIPS does not allow functions.
  - Also useful to think of state as a *set* of positive literals, manipulated using set operation.

- In STRIPS, a **goal** is a partially specified state, represented as a **conjunction** of **positive ground** literals.
  - A state s satisfies a goal g if s contains all the literals in g
  - Example: $Rich \wedge Famous \wedge Miserable$ satisfies the goal $Rich \wedge Famous$.

STRIPS Action Schema

- STRIPS specifies the result of an action in terms of what changes.
- What is left unmentioned remains the same.
- A set of actions is specified by an **action schema**, a **lifted** representation, lifted from propositional logic to a restricted subset of first order logic. Example:

$$Action(Fly(p, from, to),$$
$$\text{PRECOND} : At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$$
$$\text{EFFECT} : \neg At(p, from) \wedge At(p, to))$$

A schema consist of an action name, a list of all variables used, a **precondition** and an **effect**.

- By instantiating the variables, a schema can give ground actions, e.g.

  $Action(Fly(P_1, SFO, JFK)$,
  PRECOND : $At(P_1, SFO) \wedge Plane(P_1) \wedge Airport(SFO) \wedge Airport(JFK)$
  EFFECT : $\neg At(P_1, SFO) \wedge At(P_1, JFK))$

- The preconditions and effects are conjunctions of literals (positive or negative atomic sentences).
  - Precondition specifies states where the action can be executed: action a is **applicable** in state s if the preconds are satisfied by state s
  - In STRIPS, the precondition is a conjunction of **function-free positive** literals.

Exercise:

For an action a with v variables in a domain with k objects, how many distinct ground actions are there in the worst case?

Example action: $Fly(p, from, to)$.

A. $O(v + k)$

B. $O(v^k)$

C. $O(k^v)$

- In STRIPS, the effects is a conjunction of **function-free** (positive or negative) literals.
- The **result** of executing action a in state s is a state s' which is a set of fluents formed as follows:
  - Start from s
  - Remove fluents that appear in the action's effect as negative literals: **delete list** or $\textsc{Del}(a)$.
  - Add fluents that appear in the action's effect as positive literals: **add list** or $\textsc{Add}(a)$.

$$\textsc{Result}(s, a) = (s - \textsc{Del}(a)) \cup \textsc{Add}(a).$$

Exercise: What is the result of executing the action $Fly(P_1, SFO, JFK)$ from the following states?
$At(P_1, SFO) \land Plane(P_1) \land Airport(SFO) \land Airport(JFK)$

$Action(Fly(p, from, to),$

PRECOND : $At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$

EFFECT : $\neg At(p, from) \land At(p, to))$

A. $At(P_1, SFO) \land Plane(P_1) \land Airport(SFO) \land Airport(JFK)$
B. $At(P_1, JFK) \land Plane(P_1) \land Airport(SFO) \land Airport(JFK)$
C. Not applicable.

Exercise: What is the result of executing the action $Fly(P_1, SFO, JFK)$ from the following states?
$At(P_1, JFK) \land Plane(P_1) \land Airport(SFO) \land Airport(JFK)$

$Action(Fly(p, from, to),$
PRECOND : $At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$
EFFECT : $\neg At(p, from) \land At(p, to))$

A. $At(P_1, SFO) \land Plane(P_1) \land Airport(SFO) \land Airport(JFK)$
B. $At(P_1, JFK) \land Plane(P_1) \land Airport(SFO) \land Airport(JFK)$
C. Not applicable.

- A set of action schemas defines a planning *domain*.
- A specific problem is defined by adding an initial state and a goal.
- STRIPS summary:
  - **State:** Conjunction of ground positive function-free literals. Closed world assumption.
  - **Goal:** Conjunction of ground positive function-free literals. Satisfied by states that contain all literals.
  - **Action Precond:** Conjunction of positive function-free literals
  - **Action Effects:** Conjunction of function-free literals. Positive literals for *add* list, negative literals for *delete* list.

PDDL

- Restrictions in STRIPS makes it more efficient to solve but makes problems harder to describe.
- Various extensions makes it more expressive, e.g. Action Description Language (ADL) allows
  - positive and negative literals in states, open world assumption
  - quantified variables at goal, conjunctions and disjunctions at goal,
  - conditional effects
  - equality predicate, typed variables.
- Planning Domain Description Language (PDDL) incorporates all these and other extensions and is used at the International Planning Competition (IPC).

Air Cargo Problem

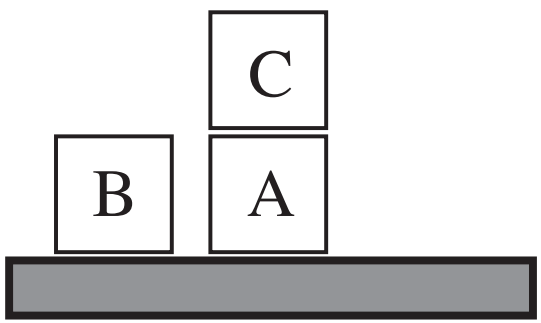
$Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK)$
$\land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2)$
$\land Airport(JFK) \land Airport(SFO))$
$Goal(At(C_1, JFK) \land At(C_2, SFO))$
$Action(Load(c, p, a),$
PRECOND: $At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $\neg At(c, a) \land In(c, p))$
$Action(Unload(c, p, a),$
PRECOND: $In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $At(c, a) \land \neg In(c, p))$
$Action(Fly(p, from, to),$
PRECOND: $At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$
EFFECT: $\neg At(p, from) \land At(p, to))$

- Problem involving loading, unloading and flying from place to place. One solution:

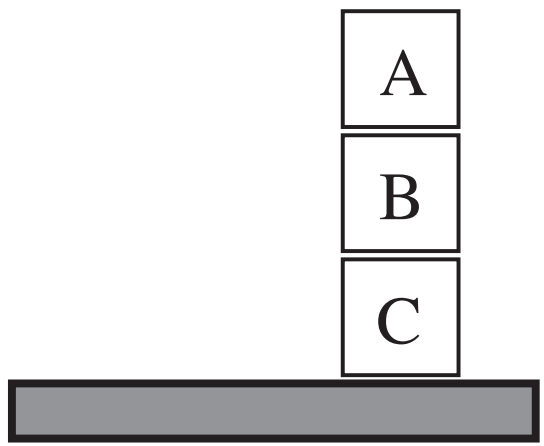
$[Load(C_1, P_1, SFO), Fly(P_1, SFO, JFK), Unload(C_1, P_1, JFK),$
$Load(C_2, P_2, JFK), Fly(P_2, JFK, SFO), Unload(C_2, P_2, SFO)]$

Blocks World

$Init(On(A, Table) \land On(B, Table) \land On(C, A)$
$\quad \land Block(A) \land Block(B) \land Block(C) \land Clear(B) \land Clear(C))$
$Goal(On(A, B) \land On(B, C))$
$Action(Move(b, x, y),$
$\quad$ PRECOND: $On(b, x) \land Clear(b) \land Clear(y) \land Block(b) \land Block(y) \land$
$\quad\quad (b \neq x) \land (b \neq y) \land (x \neq y),$
$\quad$ EFFECT: $On(b, y) \land Clear(x) \land \neg On(b, x) \land \neg Clear(y))$
$Action(MoveToTable(b, x),$
$\quad$ PRECOND: $On(b, x) \land Clear(b) \land Block(b) \land (b \neq x),$
$\quad$ EFFECT: $On(b, Table) \land Clear(x) \land \neg On(b, x))$



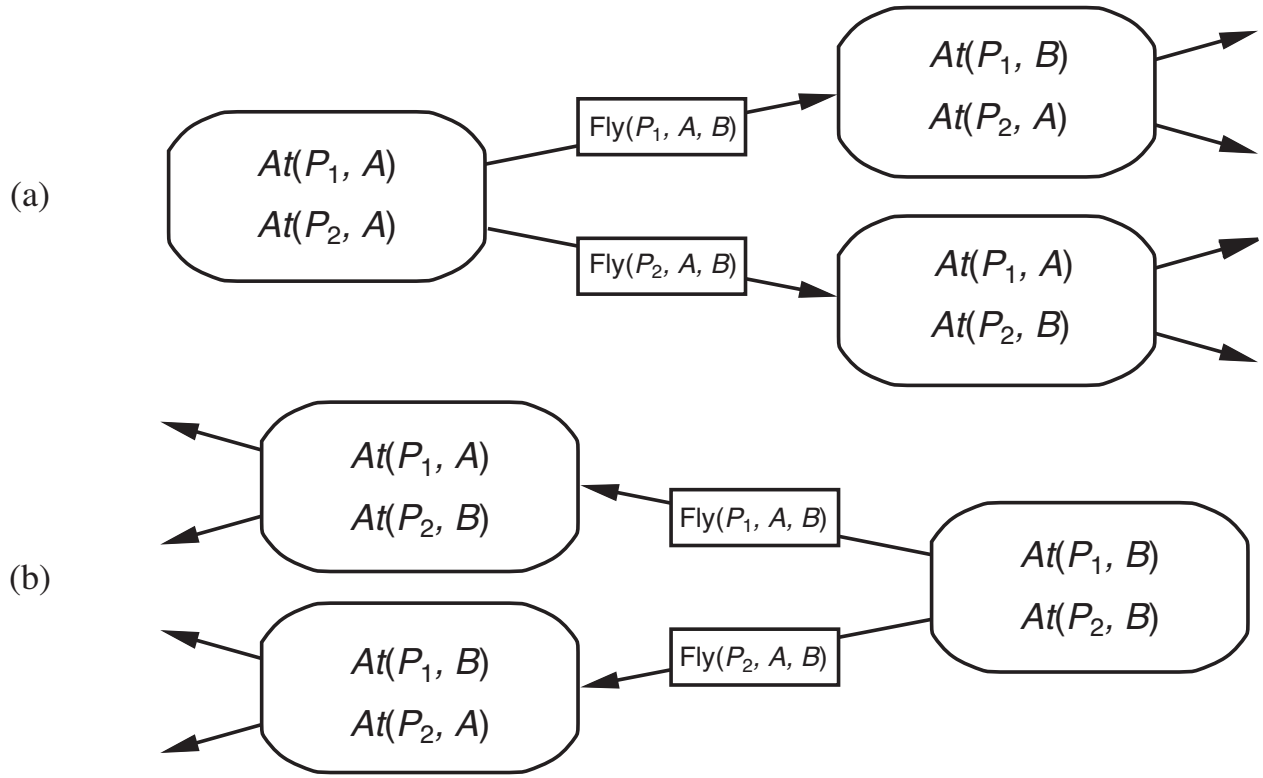
Start State



Goal State

- Here we have used inequality predicates that are not in STRIPS but in PDDL, otherwise need to explicitly add such a predicate.
- To move a block b, we need a precondition that there is no block on top of it
  - $\neg\exists x\ On(x, b)$ or $\forall x\ \neg On(x, b)$.
  - STRIPS does not allow quantifiers, so we introduce $Clear(b)$ which is true when there is nothing on b.
- The table is different from a block – it does not have to be clear for something to be put on top of it.
  - We introduce the action $MoveToTable(b, x)$ to allow the different behaviour.

Planning as State-Space Search



Can do both (a) forward search or (b) backward search.

Forward Search

- Difficulties for forward search include:
  - State space can be very large – exponential with the number of state variables
  - Action space can be very large. Example:
    - Buying a book: action schema $Buy(isbn)$ with effect $Own(isbn)$.
    - ISBNs have 10 digits, so action schema represents 10 billion ground actions.
    - Uninformed forward search will enumerate all the actions.
- Forward search hopeless without good heuristics.
- Turns out that strong domain independent heuristics can be automatically derived, making forward search feasible for many problems.

Backward (Regression) Relevant-State Search

- Called **relevant-state** search because only consider actions that are relevant to the goal, or current state.
- There is a *set* of relevant states to consider at each step.
- We distinguish between a state and a *description*
  - In a state, every variable is assigned a value. For n ground fluents, there are 2^n ground states.
  - For n ground fluents, there are 3^n descriptions: each fluent can be positive, negative, or not mentioned.
    - E.g. for goal $\neg Poor \wedge Famous$ describes states where *Poor* is false and *Famous* is true, but other unmentioned fluents can have any value.
    - A description represent a set of states.
- In backward search, we regress from a state description to a predecessor state description.

- STRIPS is designed to make regression easy. Given goal g and action a, regression from g over a gives description g':

$$g' = (g - \textsc{Add}(a)) \cup \mathit{Precond}(a).$$

  - Preconditions must have held before the action, so is added to g'
  - The add list need not be true before the action, so may be removed from g'.
  - The delete list need not be true before the action, so do not need to be added to g'.

- Using *description* allows a formula to represent a set of propositional variables. Can use a subset of first order representation to enlarge what a description can represent:
  - For goal of having cargo C_2 at SFO, $At(C_2, SFO)$, we do not care which plane is used, so use action $Unload(C_2, p', SFO)$ where p' is a variable.
  - Implicitly represent $\exists p'\ Unload(C_2, p', SFO)$.

- For backward search, we want actions that are **relevant**, can be the previous step in the plan.
  - At least one of the action's effect must unify with an element of the current goal
  - Must not have any effect that negate an element of the current goal – otherwise it cannot be the previous step in the plan.
- We substitute the most general unifier to keep the branching factor down without ruling out any solution.

Exercise: What is the regressed state description when action $Unload(C_2, p', SFO)$ is used with the goal state description $At(C_2, SFO)$?

$Action(Unload(c, p, a),$

PRECOND : $In(c, p) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$

EFFECT : $\neg At(c, a) \wedge \neg In(c, p))$

A. $In(c, p) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
B. $In(C_2, p') \wedge At(p', SFO) \wedge Cargo(C_2) \wedge Plane(p') \wedge Airport(SFO)$
C. The action is not relevant.

- Example: consider goal $Own(0125234534)$, initial state with 10 million ISBNs and a single action schema

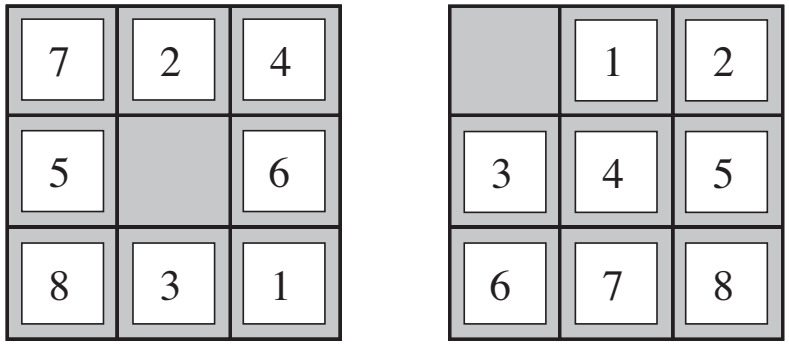
$$Action(Buy(i), \text{PRECOND} : ISBN(i), \text{EFFECT} : Own(i)).$$

  - Unify $Own(0125234534)$ with $Own(i')$, yielding substitution $\theta = \{i'/0125234534\}$.
  - Applying the substitution gives predecessor state description $ISBN(0125234534)$, which is part of the initial state, so we are done.
- Backward search has lower branching factor compared to forward search for most problem domains.
- But backward search use sets of states, and harder to come up with good heuristics for that.

Heuristics for Planning

- Factored represention for states and actions makes good domain independent heuristics possible.
- **Ignore pre-conditions heuristic** drops all pre-conditions.
  - Solves relaxed problem. Admissible.
  - Any single goal fluent achievable in one step.
  - Number of steps is roughly number of unsatisfied goals, except some actions can satisfy multiple goals, some may undo some goals.
  - If relaxed further to remove all effects except literals in the goal, problem becomes minimum number of actions such that the union of effects satisfies the goal: **set cover problem**.
    - Unfortunately, set cover is NP-hard, although greedy algorithm that selects action that covers the largest number of remaining goals returns the number of steps within $\log n$ factor of optimal, where n is number of literals in goal.

Exercise: Consider the 8 puzzle with the Slide schema



Start State Goal State

$Action(Slide(t, s_1, s_2),$
PRECOND: $On(t, s_1) \wedge Tile(t) \wedge Blank(s_2) \wedge Adjacent(s_1, s_2)$
EFFECT: $On(t, s_2) \wedge Blank(s_1) \wedge \neg On(t, s_1) \wedge \neg Blank(s_2))$

What happens if we ignore only parts of the precond?

1. What heuristic do we get when we ignore the part $Blank(s_2) \wedge Adjacent(s_1, s_2)$?

2. What heuristic do we get if we only ignore the part $Blank(s_2)$?

- **Ignore delete lists heuristic**.
  - Ignoring delete lists allows monotonic progress towards goal.
  - This is a relaxed problem, so optimal length can be used as heuristic.
  - Unfortunately relaxed problem still NP-hard
    - But hill climbing gives an approximate solution.

Exercise: Describe how the problem becomes easier when we apply ignore delete lists heuristic to this schema.

$$Action(Slide(t, s_1, s_2),$$
$$\text{PRECOND:}\, On(t, s_1) \wedge Tile(t) \wedge Blank(s_2) \wedge Adjacent(s_1, s_2)$$
$$\text{EFFECT:}\, On(t, s_2) \wedge Blank(s_1) \wedge \neg On(t, s_1) \wedge \neg Blank(s_2))$$

- Problem **decomposition** is another technique: divide into subgoals, solve subgoals, then combine.

  - If each subproblem uses an admissible heuristic, taking max is admissible.

  - Assume **subgoal independence**: sum the cost of solving each subgoal
    - Solution optimistic when there is negative interaction: action in one subplan deletes goal in another subproblem.
    - Solution pessimistic (not admissible) when there is positive interaction: action in one subplan achieves goal in another subproblem.
    - If admissible, sum is better than max.

- Example system with effective heuristics is FF (FASTFORWARD): forward state-space searcher using ignore-delete-list with hill-climbing search and also planning graph (later for CS5446).
  - Winner of International Planning Competition 2000

Outline

SATPLAN

One way to do planning is to transform the planning problem into a **Boolean satisfiability** (SAT) problem, and solve using a SAT solver.

- SAT is a NP-complete problem.
- But SAT solvers are effective in practice, able to solve some problems with millions of variables and constraints.
  - Use various domain independent heuristics that exploit structure of SAT problems to search for a solution.
  - See Section 7.6 of AIAMA.

- Solving SAT requires finding an assignment to variables that will make a Boolean formula TRUE, or to declare that no such assignment exists.
  Examples:

- SAT solvers usually takes input in **Conjunctive Normal Form** (CNF) formulas
  - A CNF formula is a conjunction of clauses
  - A clause is a disjunction of literals
  - A literal is a variable or its negation

  Examples:

CNF

Any Boolean formula can be converted into CNF. We illustrate using $A \Leftrightarrow (B \vee C)$.

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$

2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$

3. CNF requires $\neg$ to appear only on literals. Move $\neg$ inwards by repeated applications of the following:

$$\neg(\neg\alpha) \equiv \alpha \qquad \text{(double-negation elimination)}$$
$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta) \qquad \text{(De Morgan)}$$
$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta) \qquad \text{(De Morgan)}$$

4. Now have sentence with nested $\wedge$ and $\vee$. Apply distributive law, distributing $\vee$ over $\wedge$ whenever possible.

Classical Planning as Boolean Satisfiability

Translate STRIPS into SAT

- Propositionalize the actions: replace each action schema with set of ground actions by substituting constants for each variable.
  Example:

- Define initial state: assert F^0 for every fluent F in initial state and $\neg F^0$ for every fluent not in initial state.
  Example:

- Propositionalize the goal: Each goal is a conjunction – construct a disjunction over all possible ground conjunctions obtained by replacing the variables with constants.
  Example: for goal of having block A on another block at time T, $On(A, x) \wedge Block(x)$ in world with blocks A, B, and C

$$(On(A, A)^T \wedge Block(A)^T) \vee (On(A, B)^T \wedge Block(B)^T) \vee (On(A, C)^T \wedge Block(C)^T)$$

- Add **successor-state axioms**. For each fluent F, add axiom of the form

$$F^{t+1} \Leftrightarrow \mathit{ActionCausesF}^t \vee (F^t \wedge \neg \mathit{ActionCausesNotF}^t)$$

where $\mathit{ActionCausesF}^t$ is a disjunction of all ground actions that have F in their add list and $\mathit{ActionCausesNotF}^t$ is a disjunction of all ground actions that have F in their delete list.
Example:

- (Continued)
  The successor-state axioms is a clever encoding that handles the **frame problem** well.
  - The frame problem refers to the fact that most actions leave most fluents unchanged.
  - A natural way to handle this is to write **frame axioms**: for each action, assert an axiom for each fluent the action leaves unchange.
    Example:

    For m actions and n fluents, need $O(mn)$ such axioms
  - In contrast, with successor-state axioms, only need $O(n)$ axioms: each axiom is longer, but only involves actions that has an effect on the fluent.

- Add precondition axioms: For each ground action A, add axiom $A^t \Rightarrow \text{PRE}(A)^t$, i.e. if an action is taken at time t, its preconds must have been true.
  Example:

- Add action exclusion axioms: say that every action is distinct from every other actions, i.e. only one action is allowed at each time step.
  - To do this, for every pair of actions A_i^t and A_j^t, add mutual exclusion constraint $\neg A_i^t \vee \neg A_j^t$.
    Example:

If we want to allow parallel actions, add mutual exclusion only if the pair of action really interfere with each other.

The number of steps required is not known in advance: try every value for T up to $T_{\max}$.
The first value that is successful gives the shortest plan.

function SATPLAN($init$, $transition$, $goal$, $T_{\max}$) **returns** solution or failure
  inputs: $init$, $transition$, $goal$, constitute a description of the problem
    $T_{\max}$, an upper limit for plan length

  for $t = 0$ **to** $T_{\max}$ **do**
    $cnf \leftarrow$ TRANSLATE-TO-SAT($init$, $transition$, $goal$, t)
    $model \leftarrow$ SAT-SOLVER(cnf)
    if $model$ is not null **then**
      return EXTRACT-SOLUTION($model$)
  return *failure*

Instead of Boolean SAT, can also encode problem as constraint satisfaction problem (CSP). Similar, but variables need not be binary.

- SATPlan was the winner in the 2004 and 2006 International Planning Competition
  - `http://www.cs.rochester.edu/users/faculty/kautz/satplan/index.htm`
- IPC 2014 was won by *SymBA** which based on bidirectional search using heuristics and abstraction.
  - `https://fai.cs.uni-saarland.de/torralba/software.html`
- IPC 2018 was won by Delfi, which uses deep learning to select a planner from a collection of planners which include Fast Downward (uses forward search) planners with various heuristics and *SymBA**.

Complexity of Planning

- **PlanSAT** is the question of whether there is any plan that solves a planning problem.
- **Bounded PlanSAT** asks whether there is a solution of length k or less.
- Both problems are decidable for classical planning as the number of states is finite.
  - If function symbols added, number of states becomes infinite and PlanSAT becomes semidecidable (exists algorithms that will terminate for solvable problems but may not terminate for unsolvable problems). Bounded PlanSAT remains decidable.
- PlanSAT and bounded PlanSAT are in PSPACE: solvable with polynomial amount of space.
  - Still NP-hard, even if negative effects disallowed.
  - But PlanSAT in P if negative preconds also disallowed.

Reading

- AIAMA Chapter 3 (revision)
- AIAMA Chapter 10.1, 10.2, 10.4.1
- AIAMA Section 7.7.4

Acknowledgement: Figures from AIAMA