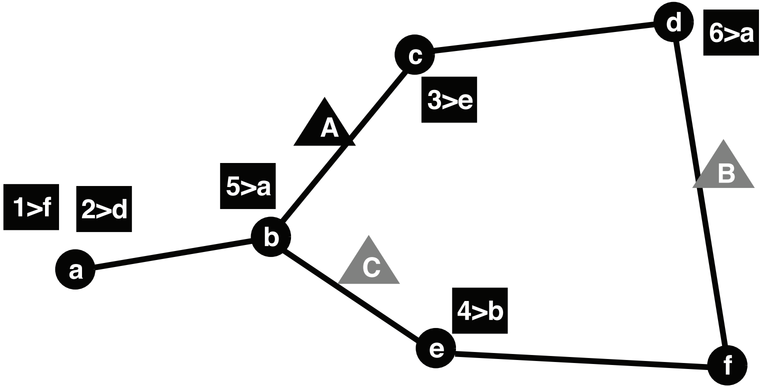
Factored Representations

Boi Faltings

Laboratoire d'Intelligence Artificielle
boi.faltings@epfl.ch
http://moodle.epfl.ch/

Fall 2019

Recall: delivery problem



Agent A delivers packages 1..6 to their destinations

Similar real life planning problems

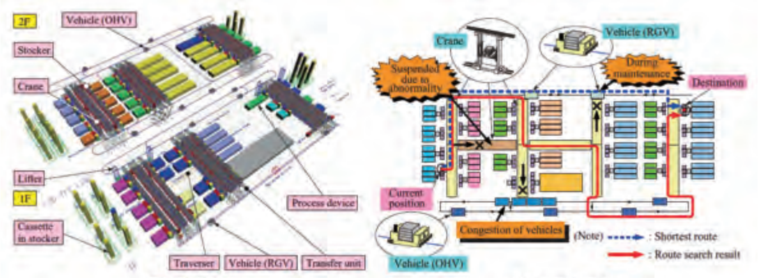
Airport ground-traffic control

- Guide aircrafts between runway and terminals
- Keep safe distances, minimize taxi and wait times

More real life planning problems

Material control system for LCD manufacturing

- Transfer LCD cassettes to different locations in the plant
- Handle hardware failure and minimize delays



(a) High-level view (b) Sample route !"#$%&'()$*+,-$

State space explosion

Multiple features $\Rightarrow$ combinatorial explosion of state space:

pos(A)=a/b/c/d/e/f	pos(1) = a/b/c/d/e/f	holding(1)=t/f
	pos(2) = a/b/c/d/e/f	holding(3)=t/f
	...	...
	pos(6) = a/b/c/d/e/f	holding(6)=t/f

$\Rightarrow$ 17'915'904 states

Only few states are reachable

Successor states of
$\{pos(A) = a, pos(1) = a, pos(2) = a, pos(3) = c, ...\}$:

- $\{pos(A) = a, ..., holding(1), ...\}$
- $\{pos(A) = a, ..., holding(2), ...\}$
- $\{pos(A) = b, ...\}$

few actions $\Rightarrow$ few successors.

$\Rightarrow$ construct states dynamically as combination of features.

Many states are equivalent

Can agent move package 1 from a to b:

- pos(A) = a/elsewhere?
- pos(1) = a/elsewhere?
- holding(1)?

Differences in other features ⇒ equivalent states.
Only $2^3 = 8$ possibilities to distinguish!
⇒ drop features that don't matter!

Factored representations

Idea:

- model each feature by separate predicates (factors).
- represent only those that are important for the current goal.

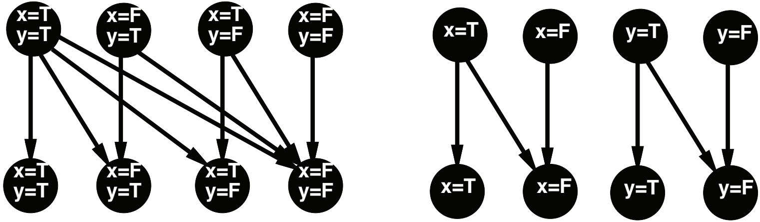
Construct states as needed for planning.

$\Rightarrow$ equivalent states treated as one in the planning process.

$\Rightarrow$ dramatic reduction in complexity.

Factored representations (2)

Factored state transitions:



Treat factors independently $\Rightarrow$ reduce combinatorics.

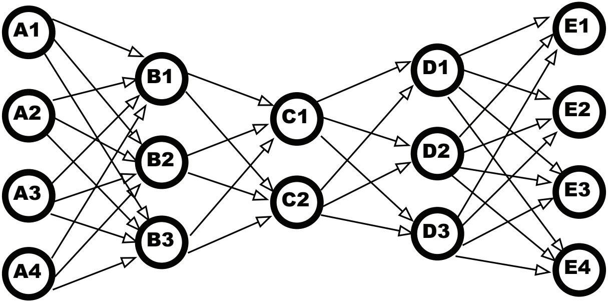
Techniques for factored representations

- State X = vector of k state variables $(x_1, x_2, ..., x_k)$.
- Represent all states by domains of state variables.
- Formulate successor functions and rewards as functions on the vector of state variables.

Logic-based factored representations

- Basis: predicate logic.
- Each state is modelled by a set of true/false predicates.
- De-facto standard: *situation calculus*.
- Most suitable for deliberative agents (planning).

Neural Networks



- Neurons in each layer represent state variables.
- Train autoencoder net to discover a compact factorization: A/E = "raw" state variables, C = factorization (embedding).

Factored Decision Processes

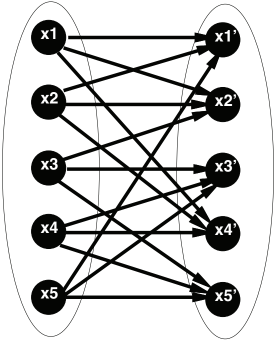
- State $X = (x_1, .., x_k)$
- Each x_i is chosen from a finite domain d_i.
- All combinations are allowed $\Rightarrow$ exponential state space.
- Delivery problem: positions of agents, packages, goals.

Bayesian Networks

Bayesian network =

- nodes = events, for example $x_3 = c$.
- arcs = causation, for example $(x_3 = c) \rightarrow (x_7 = d)$.
- causation is uncertain, expressed by probability distribution: for arc $(x_i \rightarrow x_j)$, express $P(x_j | x_i)$ for all $x_j \in d_j, x_i \in d_i$.

Dynamic Bayesian Networks



- Nodes $= (x_1, .., x_k) \cup (x'_1, .., x'_k)$
- X = states at time t
- X' = states at time t+1
- No arcs between x_is or x'_is
- Arc from x_i to x'_j if x_i influences x'_j

Model transitions by a separate DBN for each action
(more efficient representations possible)

Probability distributions

Every variable x_i' with inbound nodes $x_j, .., x_k$ has a probability distribution:

$$p_i(x_i' | x_j, .., x_k)$$

Transition probability between states given as product of distributions:

$$\begin{aligned} & pr(x_1' = v_1, .., x_n' = v_n | x_1 = w_1, .., x_n = w_n) \\ = & \prod_{l=1}^{n} p_l(x_l' = v_l | x_j = w_j, .., x_k = w_k) \end{aligned}$$

Best if relations are as local as possible!

Example: Delivery Problem

DBN for agent A moving package 1 from a to b

- $x_1 = \text{pos}(1)$, $x_2 = \text{pos}(A)$, ...
- x_1' influenced by x_1 and $x_2 \Rightarrow$ arcs
- Probability distributions:

$$\begin{aligned} p(x_1' = y | x_1 = x, x_2 = x) &= 1, \qquad x = a, y = b \\ p(x_1' = y | x_1 = x, x_2 = x) &= 0, \qquad x \neq a, y \neq b \\ \text{same for } p(x_2' = y | x_1 = x, x_2 = x) & \end{aligned}$$

Frame axioms:

$$\begin{aligned} p(x_i' = x | x_i = x) &= 1 \qquad i \neq 1, 2 \\ p(x_i' = x | x_i = y) &= 0 \qquad x \neq y, i \neq 1, 2 \end{aligned}$$

Factoring the policy

- Policy should be formulated on factored state representation.
- Generally, number of possible actions is small $\Rightarrow$ many states have the same optimal action.
- Assumption: states with same action have a compact description.

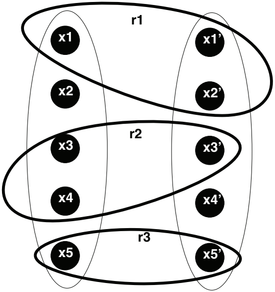
$\Rightarrow$ Policy = mapping (optimal action) $\Rightarrow$ set of states.

- Examples: combinations of features, decision tree on features.

Factoring the reward function

- Instantaneous rewards depend on action and state variables.
- Factored reward function: reward depends on action and *subset* of state variables.
- e.g. delivery problem: agent A moves package 1 from a to b: reward = 1 if successful (as in transition) and b is the goal.
- $\Rightarrow$ dependency on: pos(1), pos(A), destination(1)
- Separate reward function for each possible action.
- Select depending on action actually executed.

- Reward functions depend on nodes in dynamic Bayes net.
- Total reward = sum of rewards.
- Can we also factor the value of a state?



Factoring the value function

- Factored reward function $\not\Rightarrow$ factored value function!
- Value of pos(A)=b also depends on positions of agents and other packages; if also pos(B)=b then value increment is lower because agent B can also carry the package from b.
- Factor value function into basis functions $b_i(U), U \subset X$, such that $V(X) = \sum w_i b_i$
- Optimal policy is computed using the same factoring as the value function.

Value function factored into basis functions

- $V(X) = \sum w_i b_i$
- Basis functions b_i are programmed by analysis of the system (ex deliveries: analyze dependencies).
- Weights w_i are determined so that the overall mean square error is minimized.
- Disadvantage: approximation depends on the quality of the basis functions.
- Advantage: often just few basis functions $\Rightarrow$ manageable complexity.

Value iteration?

- As value function $V(S) = \sum w_i b_i(S)$, value iteration recurrence becomes:

$$\begin{aligned} V(S) &= R(S,\pi(S)) + \gamma \sum_{S' \in \mathcal{S}} T(S,\pi(S),S')V(S') \\ \sum w_i b_i(S) &= R(S,\pi(S)) + \gamma \sum_{S' \in \mathcal{S}} T(S,\pi(S),S') \sum w_i b_i(S') \end{aligned}$$

  with one equation for each state.
- Many more states than basis functions $\Rightarrow$ more equations than unknowns w_i.

$\Rightarrow$ solve approximately for least-squares approximation of b_i.

- However, far too many states: intractable to solve.

Policy iteration

Factored policy $\Rightarrow$ factored state space; use for approximating value function in policy iteration:

```
choose an arbitrary policy π'
loop
  π ← π'
  compute the value function V_π of policy π:
   solve the linear equations:
    V_π(S) = R(S, π(S)) + γ Σ_{S'∈𝒮} T(S, π(S), S') V_π(S')
  improve the policy at each state:
   π'(s) ← argmax_a (R(s, a) + γ Σ_{s'∈S} T(s, a, s') V_π(s'))
until π = π'
```

$$V_\pi(S) = R(S,\pi(S)) + \gamma \sum_{S' \in \mathcal{S}} T(S,\pi(S),S') V_\pi(S')$$

$$\pi'(s) \leftarrow \operatorname*{argmax}_a \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_\pi(s')\right)$$

Alternative: Deep Reinforcement Learning

Neural networks can be seen as a kind of factored representation:

- input units = factors of state representation
  (state = *combination* of activation levels)
- output units = factors of action representation.

Deep learning finds a complex policy mapping *state* $\Rightarrow$ *action*.

.

Factors of reward function a good starting point for neural units.

Factored representations for deliberative agents

- Factoring helps to reduce complexity of decision processes.
- But still generates policies for all imaginable states $\Rightarrow$ unnecessary complexity.
- Often, need to use deliberative agents.

A simple delivery world

Consider a robot moving packages among a network of locations.
We use the following predicates to model the world in state S:

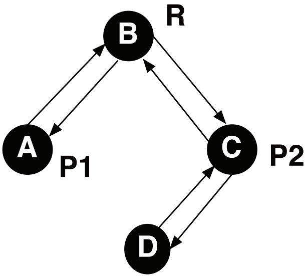
- `AT(p,l,`S`)`: object `p` is at location `l`
- `LOC(l,`S`)`: the robot is at location `l`
- `CONN(l1,l2,`S`)`: there is a connection from location `l1` to `l2`

Argument S allows different truth values in different states.

Modeling world states

States modelled by a set of propositions, e.g. state S_0:

AT(P1,A,S_0)
AT(P2,C,S_0)
LOC(B,S_0)
CONN(A,B,S_0)
CONN(B,A,S_0)
CONN(B,C,S_0)
CONN(C,B,S_0)
CONN(C,D,S_0)
CONN(D,C,S_0)



Situations and states

States contain unnecessary details:

$\mathtt{AT(P2,C,}S_0\mathtt{)}$ *not important for plan* $\Rightarrow$ *drop*

$\Rightarrow$ partial models = *Situations*

- Situations allow specifying goals, more general plans.
- Operators model agent actions.
- In *situation calculus*, operators transform situations.

Operators (situation calculus = STRIPS)

An operator $S_i \Rightarrow S_{i+1}$ is characterized by:

- *preconditions*: propositions which must be true in S_i for the operator to be applicable.
- *postconditions*: propositions which will be true in S_{i+1} as a result of the action. Also called ADD-LIST.
- *DELETE-LIST*: propositions which will no longer be true in S_{i+1}. Often identical with preconditions.

Example: CARRY

$\texttt{CARRY(p,l1,l2,}S_i\texttt{)}$ models the action of the robot carrying object p from location l1 to l2.

It is defined as follows:

- preconditions (P): $\texttt{CONN(l1,l2,}S_i\texttt{)}$, $\texttt{LOC(l1,}S_i\texttt{)}$, $\texttt{AT(p,l1,}S_i\texttt{)}$
- add-list (A): $\texttt{LOC(l2,}S_{i+1}\texttt{)}$, $\texttt{AT(p,l2,}S_{i+1}\texttt{)}$
- delete-list (D): $\texttt{LOC(l1,}S_{i+1}\texttt{)}$, $\texttt{AT(p,l1,}S_{i+1}\texttt{)}$

Example: MOVE

MOVE(l1,l2,S_i) models the action of the robot moving from location l1 to l2 without carrying anything.
It is defined as follows:

- preconditions (P): CONN(l1,l2,S_i), LOC(l1,S_i)
- add-list (A): LOC(l2,S_{i+1})
- delete-list (D): LOC(l1,S_{i+1})

Situation calculus as a factored representation

- Operations in delivery world are independent of the positions of irrelevant objects.
- Situation calculus does not represent these
  $\Rightarrow$ automatically groups equivalent states.
- Situation calculus operators
  $\Rightarrow$ basis for state-based planning by search.

Least committment

- In general, planning problems solved by search.
- Assume actions A,...,Z can be carried out in any order.

$\Rightarrow$ 26! different (ordered) plans:

$$(A, B, ..., Z), (B, A, ..., Z), ...$$

Search treats all of them separately!

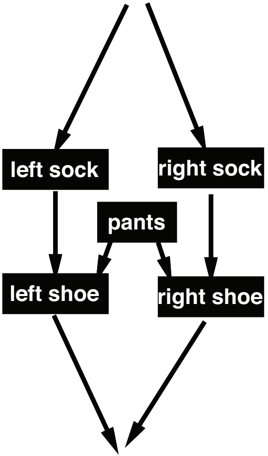
- Delay committment on order $\Rightarrow$
  1 plan with 26 parallel actions:

$$(..., \{A, B, ..., Z\}, ...)$$

Less to enumerate $\Rightarrow$ more efficient search.

Non-linear (partial-order) planning

Plan indicates partial orders:



Boxes = operations
Arrows = precedence relations

Making nonlinear planning efficient

- Problem: complexity of establishing causal links in a non-conflicting fashion.
- Idea: explicitly generate sets of actions that can be executed in parallel.

$\Rightarrow$ plan = sequence of (potentially parallel) action groups.

- Generate the groups to avoid conflicts in causal links.

$\Rightarrow$ no need for backtracking on this link structure.

Constructing a finite decision space

- Assumptions:
  - finite universe ⇒
    finite set of propositions
  - plan has at most n steps (use iterative deepening)
- Each proposition becomes a state variable ⇒ universe of state variables for each of the n states.
- Only reachable propositions are represented!

Example

5 state variables:

- `ls`,`rs`: wearing left/right sock
- `lh`,`rh`: wearing left/right shoe
- `p`: wearing pants

Initial state: $\neg$ `ls`, $\neg$ `rs`, $\neg$ `lh`, $\neg$ `rh`, $\neg$ `p`

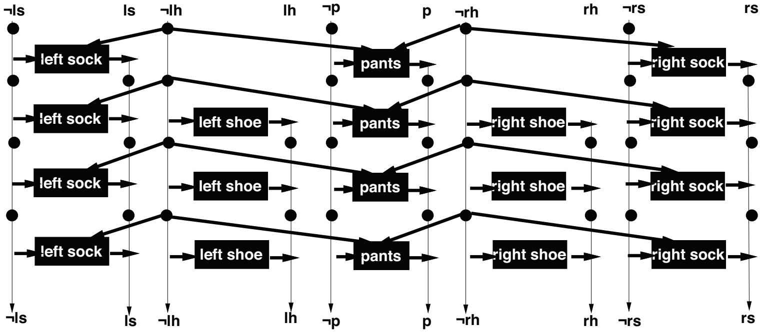
5 operators:

- `left/right sock`:
  `P={¬ls/rs, ¬lh/rh}, D={¬ls/rs}, A={ls/rs}`
- `left/right shoe`:
  `P = {¬lh/rh, ls/rs}, D = {¬lh/rh}, A = {lh/rh}`
- `pants`:
  `P = {¬p, ¬lh, ¬rh}, D = {¬p}, A = {p}`

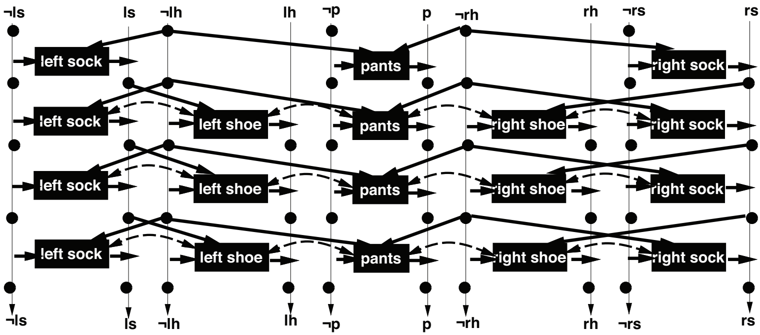
Planning graphs

- Idea: consider what actions can be carried out simultaneously.
- Graph has *layers*. Each layer contains nodes
  - for each possible proposition that could hold
  - for each possible action
- First layer: initial state + actions possible in initial state.
- Following layers: propositions and actions that might be possible.
- Construction: polynomial time.

Planning graph example



Exclusion relations



- Interference, Competing needs = constraints!
- Result: identify what actions can take place in parallel.

Backward search

- Goals at time t: `ls,rs,lh,rh,p`
- Select set of non-exclusive actions at time t:
  1. `{pants, left sock, right sock}`
  2. `{left shoe, right shoe}`

⇒ Goals at time t-1:
  1. ¬ `p`, ¬ `ls`, ¬ `rs`, ¬ `lh`, ¬ `rh`, `lh`, `rh`
  2. ¬ `lh`, ¬ `rh`, `ls`, `rs`, `p`

- Goals (1) contradictory ⇒ backup.

Search (continued)

- Goals (2) $\Rightarrow$ non-exclusive actions:

  {`left sock`, `right sock`, `pants`}

$\Rightarrow$ Goals at time t-2:

  ¬ `p`, ¬ `ls`, ¬ `rs`, ¬ `lh`, ¬ `rh`

  Satisfied by initial conditions!

- Plan:
  1. {`left sock`, `right sock`, `pants`}
  2. {`left shoe`, `right shoe`}

Testing for unsolvability

- After n levels, all levels of the graph will become identical.
- If the goal state is not contained in this state, or ruled out by mutual-exclusion constraints, the problem is unsolvable.
- Extension of this criterion for general unsolvability test.

Power of Graphplan

- Every feasible plan of t or less steps exists in graph.
- Graph has only polynomially many nodes.
- Exclusion constraints can propagate.
- Considers goal/action sets $\Rightarrow$ small search space.
- Can combine with search heuristics for constraint satisfaction.

Does Graphplan give all solutions?

No - plan not included:

`left sock, pants, left shoe, right sock, right shoe`

However, complete:

no solution in Graphplan

$\Rightarrow$ *no plan exists*

Using Graphplan as a heuristic

- Simplification: ignore all delete lists.

$\Rightarrow$ after forming feasible sets, can find a plan without backtracking.

- Use as a heuristic for planning with A* algorithm: **F**ast **F**orward algorithm.
- FF often much faster than Graphplan itself.

Graphplan ⇒ Satisfiability

- Consider collection of *clauses*:

$$l1 \lor l2 \lor l3 \lor \ldots$$

  for example:

$$(AT(P1, A, S_0) \land CARRY(P1, A, B, S_0)) \Rightarrow AT(P1, B, S_1)$$

  ⇒ clause:

$$\neg AT(P1, A, S_0) \lor \neg CARRY(P1, A, B, S_0) \lor AT(P1, B, S_1)$$

- Satisfiability (SAT): what truth assignment will make a collection of clauses simultaneously true.
- Research has obtained efficient search algorithms for this problem.

Graphplan as SAT

Variables:

- each operator at each time instant.
- each proposition at each time instant.

Constraints (clauses):

- each operator with its preconditions in preceeding state.
- each operator with its postconditions in following state.
- exclusions among propositions in each state.
- exclusions among operators using the same resource.

Assumptions

Same as for Graphplan:

- finite set of propositions (fluents).
- plan has at most n steps.

Time broken up into $2n$ points:

- even: states.
- uneven: actions.

Encoding as literals

Literals of the SAT problem =

- all propositions describing states at every even time, e.g.:
  $AT(P1, A, 0), AT(P1, A, 2), AT(P1, A, 4), ...AT(P1, A, 2n)$
  $AT(P1, B, 0), AT(P2, A, 0), AT(P2, B, 0), ...$
- all possible actions at every odd time, e.g.:
  $MOVE(A, B, 1), MOVE(A, B, 3), ..., MOVE(A, B, 2n-1)$
  $CARRY(P1, A, B, 1), CARRY(P1, A, B, 3), ...$

Axioms:

- `INIT`, `GOAL`: for initial and final conditions.
- for every action *op* and every odd time *t*, an implication

$$op(t) \Rightarrow P(t-1), A(t+1), \neg D(t+1)$$

  (Notation: $E = A \cup \neg D$)
- frame axioms.

Action representation

- Problem:
  operators with k arguments, l values have l^k possible instantiations $\Rightarrow$ l^k literals
- Operator splitting:
  $$op(A, B, C, t) \rightarrow$$
  $$op1(A, t) \wedge op2(B, t) \wedge op3(C, t)$$
- Overloaded operator splitting:
  $$op1(A, B, C, t) \rightarrow$$
  $$act(op1, t), arg1(A, t), arg2(B, t), arg3(C, t)$$
- Bitwise representation:
  $$act(\{op1, op2, op3, op4\}, t) \rightarrow$$
  $$act-bit1(\{0, 1\}, t), act-bit2(\{0, 1\}, t)$$

Frame Axioms

- Frame axioms: ensure that propositions not affected by actions remain true.
- Classical frame axioms: associated with every operator

$$AT(P2, C, t-1) \wedge CARRY(P2, A, B, t) \Rightarrow AT(P2, C, t+1)$$

$\Rightarrow$ must require ≥ 1 action at every time.

- Explanatory frame axioms:

$$AT(P2, C, t-1) \wedge \neg AT(P2, C, t+1) \Rightarrow$$
$$CARRY(P2, C, D, t) \vee CARRY(P2, C, B, t) \vee ...$$

- Exclusion constraints: actions with conflict between precondition/effect cannot be executed in parallel.

Factoring

- Some literals are irrelevant, e.g. in:

$$MOVEarg1(A, t) \wedge MOVEarg2(B, t) \Rightarrow AT(P1, B, t + 1)$$

  we can drop $MOVEarg1(A, t)$:

$$MOVEarg2(B, t) \Rightarrow AT(P1, B, t + 1)$$

- Can be applied to frame, exclusion and at-least-one axioms.
- But need to make sure that all parts of an operator are instantiated, otherwise action may not be executable!

Comparison with MDP

MDP:

- any state can be sucessor to another state with some probability.
- $\Rightarrow$ need to immediately plan for all states...

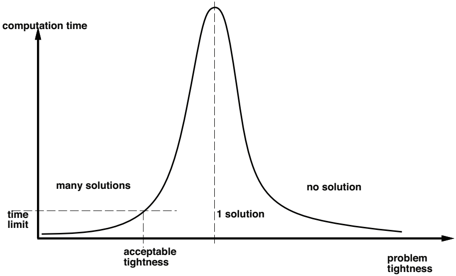
$\Rightarrow$ focus instead on limited uncertainty:

- uncertain effects (but with limited set of choices)
- conditional effects (predictable)
- measurement actions

can be integrated particularly well with SAT and constraint satisfaction techniques.
Active research area.

Phase transition behavior

- SAT expected computation time depends on *tightness*:



⇒ agent has to operate with sufficient liberty.

Summary

- Factored, logical representations can greatly reduce complexity of planning.
- Can improve efficiency of reactive agents (but complex).
- Widely used in deliberative agents.
- Efficiency gains through least-committment principle:
  - operator order: non-linear, partial-order planning
  - operator choice: graphplan, SATplan
- No solution with neural nets (so far).