Sessions 1 & 2
Introduction to AI;
Planning & Search

CSE 592

Applications of Artificial Intelligence

Henry Kautz

Winter 2003

What is Intelligence?
What is Artificial Intelligence?

What is Artificial Intelligence?

- The study of the principles by which natural or artificial machines manipulate knowledge:
  - how knowledge is acquired
  - how goals are generated and achieved
  - how concepts are formed
  - how collaboration is achieved

. . . Exactly what the computer provides is the ability not to be rigid and unthinking but, rather, to behave conditionally. That is what it means to apply knowledge to action: It means to let the action taken reflect knowledge of the situation, to be sometimes this way, sometimes that, as appropriate. . . .

-Allen Newell

- Classical AI
  Disembodied Intelligence
- Autonomous Systems
  Embodied Intelligence

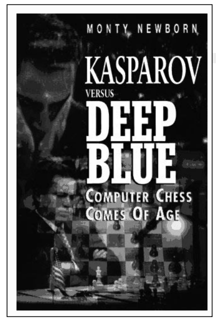
Classical AI

- The principles of intelligence are separate from any hardware / software / wetware implementation
  - logical reasoning
  - probabilistic reasoning
  - strategic reasoning
  - diagnostic reasoning
- Look for these principles by studying how to perform tasks that require intelligence

Success Story: Medical Expert Systems

- Mycin (1980)
  - Expert level performance in diagnosis of blood infections
- Today: 1,000's of systems
  - Everything from diagnosing cancer to designing dentures
  - Often outperform doctors in clinical trials
  - Major hurdle today – non-expert part – doctor/machine interaction

Success Story: Chess



I could feel – I could smell – a new kind of intelligence across the table
- Kasparov

•Examines 5 billion positions / second

•Intelligent behavior emerges from brute-force search

Autonomous Systems

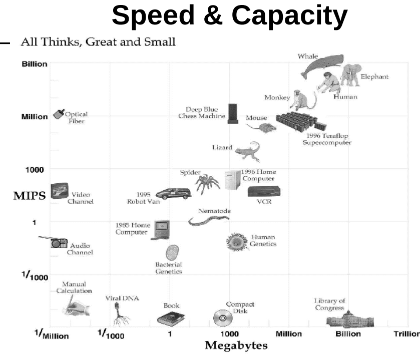
- In the 1990's there was a growing concern that work in classical AI ignored crucial scientific questions:
  - How do we integrate the components of intelligence (*e.g.* learning & planning)?
  - How does perception interact with reasoning?
  - How does the demand for real-time performance in a complex, changing environment affect the architecture of intelligence?

RoboCup

- Provide a standard problem where a wide range of technologies can be integrated and examined
- By 2050, develop a team of fully autonomous humanoid robots that can win against the human world champion team in soccer.

Experiment: May 17-21 courtesy JPL

Speed & Capacity



Not Speed Alone…

- Speech Recognition
  - "Word spotting" feasible today
  - Continuous speech – rapid progress
  - Turns out that "low level" signal not as ambiguous as we once thought
- Translation / Interpretation / Question-answering
  - Very limited progress

  The spirit is willing but the flesh is weak. (English)

  The vodka is good but the meat is rotten. (Russian)

Varieties of Knowledge

What kinds of knowledge required to understand –

- Time flies like an arrow.
- Fruit flies like a banana.
- Fruit flies like a rock.

Historic Perspective

1940's - 1960's: Artificial neural networks
- McCulloch & Pitts 1943

1950's - 1960's: Symbolic information processing
- General Problem Solver – Simon & Newell
- "Weak methods" for search and learning
- 1969 - Minsky's *Perceptrons*

1940's – 1970's: Control theory for adaptive (learning) systems
- USSR – Cybernetics – Norbert Weiner
- Japan – Fuzzy logic

1970's – 1980's: Expert systems
- "Knowledge is power" – Ed Feigenbaum
- Logical knowledge representation
- AI Boom

1985 – 2000: A million flowers bloom
- Resurgence of neural nets – backpropagation
- Control theory + OR + Pavlovian conditioning = reinforcement learning
- Probabilistic knowledge representation – Bayesian Nets – Judea Pearl
- Statistical machine learning

2000's: Towards a grand unification
- Unification of neural, statistical, and symbolic machine learning
- Unification of logic and probabilistic KR
- Autonomous systems

… In sum, technology can be controlled especially if it is saturated with intelligence to watch over how it goes, to keep accounts, to prevent errors, and to provide wisdom to each decision.

-Allen Newell

Course Mechanics

Topics
- What is AI?
- Search, Planning, and Satisfiability
- Bayesian Networks
- Statistical Natural Language Processing
- Decision Trees and Neural Networks
- Data Mining: Pattern Discovery in Databases
- Planning under Uncertainty and Reinforcement Learning
- Autonomous Systems Case Studies
- Project Presentations

Assignments
- 4 homeworks
- Significant project & presentation

Information
- http://www.cs.washington.edu/education/courses/592/03wi/

Planning & Search

Search – the foundation for all work in AI
- Deduction
- Probabilistic reasoning
- Perception
- Learning
- Game playing
- Expert systems
- Planning

R&N Ch 3, 4, 5, 11

Planning

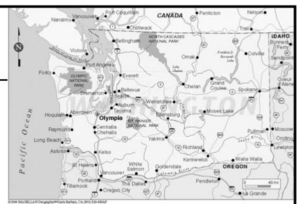
- Input
  - Description of set of all possible states of the world *(in some knowledge representation language)*
  - Description of initial state of world
  - Description of goal
  - Description of available actions
    - May include costs for performing actions
- Output
  - Sequence of actions that convert the initial state into one that satisfies the goal
  - May wish to minimize length or cost of plan

19

Classical Planning

- ***Simplifying assumptions***
  - Atomic time
  - Actions have deterministic effects
  - Agent knows complete initial state of the world
  - Agent knows the effects of all actions
  - States are either goal or non-goal states, rather than numeric utilities or rewards
  - Agent is sole cause of change
- All these assumptions can be relaxed, as we will see by the end of the course…

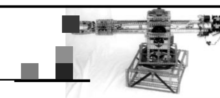
Example: Route Planning



Input:
- **State set**
- **Start state**
- **Goal state test**
- **Operators**

Output:

Example: Robot Control (Blocks World)



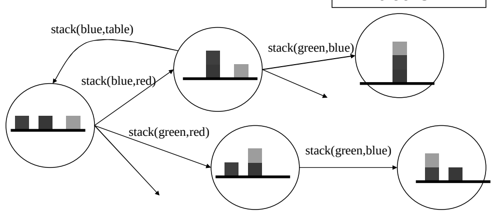
Input:
- **State set**
- **Start state**
- **Goal state test**
- **Operators (and costs)**

Output:

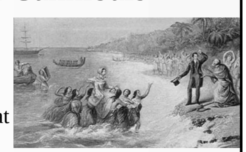
Implicitly Generated Graphs

- Planning can be viewed as finding paths in a graph, where the graph is implicitly specified by the set of actions
- Blocks world:
  - vertex = relative positions of all blocks
  - edge = robot arm stacks one block

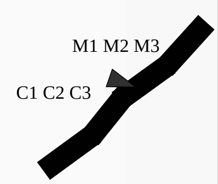
How many states for K blocks?



Missionaries and Cannibals



- 3 missionaries M1, M2, M3
- 3 cannibals C1, C2, C3
- Cross in a two person boat, so that missionaries never outnumber cannibals on either shore
- What is a state? How many states?



24

STRIPS Representation

- *Description of initial state of world*

  Set of propositions that completely describes a world

  { (block a) (block b) (block c) (on-table a) (on-table b) (clear a) (clear b) (clear c) (arm-empty) }
- *Description of goal (i.e. set of desired worlds)*

  Set of propositions that partially describes a world

  { (on a b) (on b c) }
- *Description of available actions*

How Represent Actions?

- World = set of propositions true in that world
- Actions:
  - Precondition: conjunction of propositions
  - Effects: propositions made true & propositions made false (deleted from the state description)

operator: **stack_B_on_R**

precondition: **(on B Table) (clear R)**

effect: **(on B R) (:not (clear R))**

Action Schemata

- Compact representation of a large set of actions

```
(:operator pickup
    :parameters ((block ?ob1))
    :precondition (:and (clear ?ob1) (on-table ?ob1)
                        (arm-empty))
    :effect (:and (:not (on-table ?ob1))
                  (:not (clear ?ob1))
                  (:not (arm-empty))
                  (holding ?ob1)))
```

Search Algorithms

Backtrack Search

1. **DFS**
2. **BFS / Dijkstra's Algorithm**
3. **Iterative Deepening**
4. **Best-first search**
5. **A***

Constraint Propagation

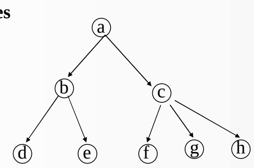
1. **Forward Checking**
2. **k-Consistency**
3. **DPLL & Resolution**

Local Search

1. **Hillclimbing**
2. **Simulated annealing**
3. **Walksat**

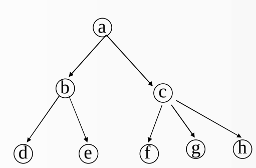
Depth First Search

- Maintain stack of nodes to visit
- Evaluation
  - Complete?
    Not for infinite spaces
  - Time Complexity?
    O(b^d)
  - Space Complexity?
    O(d)



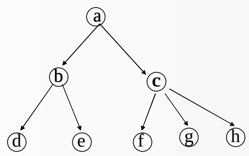
Breadth First Search

- Maintain queue of nodes to visit
- Evaluation
  - Complete?
    Yes
  - Time Complexity?
    O(b^d)
  - Space Complexity?
    O(b^d)



Iterative Deepening Search

- DFS with limit; incrementally grow limit
- Evaluation
  - Complete?
    Yes
  - Time Complexity?
    O(b^d)
  - Space Complexity?
    O(d)



Dijkstra's Shortest Path Algorithm

- Like breadth-first search, but uses a priority queue instead of a FIFO queue:
  - Always select (expand) the vertex that has a lowest-cost path from the initial state
- Correctly handles the case where the lowest-cost path to a vertex is not the one with fewest edges
  - Handles actions planning with costs, with same advantages / disadvantages of BFS

Pseudocode for Dijkstra

- Initialize the cost of each *vertex* to ∞
- cost[s] = 0;
- heap.insert(s);
- While (! heap.empty())

```
n = heap.deleteMin()
For (each vertex a which is adjacent to n along edge e)
  if (cost[n] + edge_cost[e] < cost[a]) then
     cost [a] = cost[n] + edge_cost[e]
     previous_on_path_to[a] = n;
     if (a is in the heap) then heap.decreaseKey(a)
                    else heap.insert(a)
```

Important Features

- Once a vertex is removed from the head, the cost of the shortest path to that node is known
- While a vertex is still in the heap, another shorter path to it might still be found
- The shortest path itself from s to any node a can be found by following the pointers stored in previous_on_path_to[a]

Edsger Wybe Dijkstra
(1930-2002)

- **Invented concepts of structured programming, synchronization, weakest precondition, and semaphores**
- **1972 Turing Award**
- *"In their capacity as a tool, computers will be but a ripple on the surface of our culture. In their capacity as intellectual challenge, they are without precedent in the cultural history of mankind."*

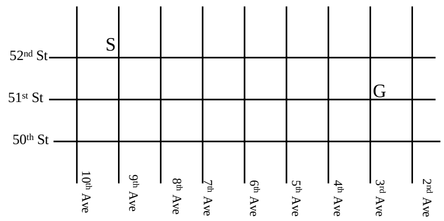
Heuristic Search

- A heuristic is:
  - Function from a state to a real number
    - Low number means state is close to goal
    - High number means state is far from the goal

Designing a good heuristic is very important!
(And often hard! Later we will see how
some heuristics can be created *automatically*)

An Easier Case

- Suppose you live in Manhattan; what do you do?

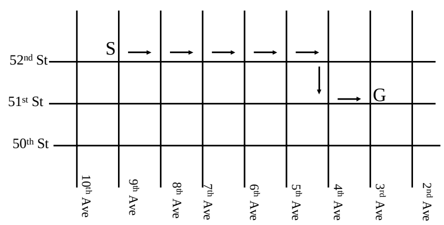


Best-First Search

- The *Manhattan distance* (Δ x+ Δ y) is an estimate of the distance to the goal
  - a heuristic value
- Best-First Search
  - Order nodes in priority to minimize estimated distance to the goal h(n)
- Compare: BFS / Dijkstra
  - Order nodes in priority to minimize distance from the start

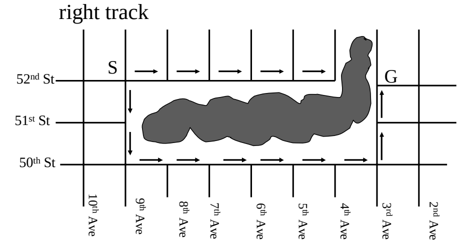
Best First in Action

- Suppose you live in Manhattan; what do you do?



Problem 1: Led Astray

- Eventually will expand vertex to get back on the right track

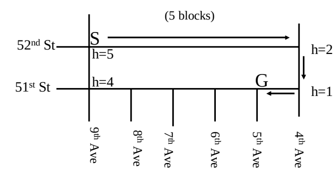


Problem 2: Optimality

- With Best-First Search, are you *guaranteed* a shortest path is found when
  - goal is first seen?
  - when goal is removed from priority queue (as with Dijkstra?)

Sub-Optimal Solution

- No! Goal is by definition at distance 0: will be removed from priority queue immediately, even if a shorter path exists!



Synergy?

- Dijkstra / Breadth First guaranteed to find *optimal* solution
- Best First often visits *far fewer* vertices, but may not provide optimal solution
  - *Can we get the best of both?*

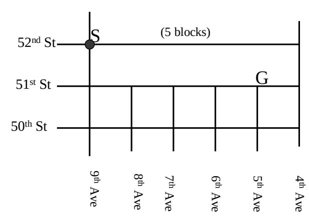
A* ("A star")

- Order vertices in priority queue to minimize
- (distance from start) + (estimated distance to goal)
- f(n) = g(n) + h(n)
- f(n) = priority of a node
- g(n) = true distance from start
- h(n) = heuristic distance to goal

Optimality

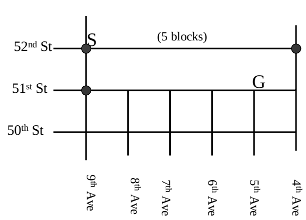
- Suppose the estimated distance (h) is *always* less than or equal to the true distance to the goal
  - heuristic is a *lower bound on true distance*
  - *heuristic is admissible*
- Then: when the goal is removed from the priority queue, we are guaranteed to have found a shortest path!

Problem 2 Revisited



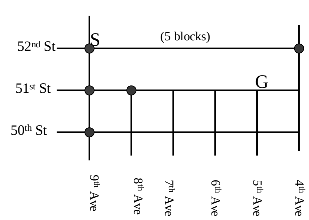
vertex	g(n)	h(n)	f(n)
52nd & 9th	0	5	5

Problem 2 Revisited



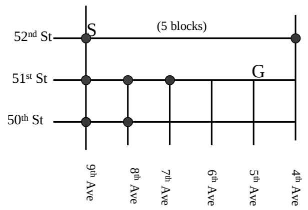
vertex	g(n)	h(n)	f(n)
52nd & 4th	5	2	7
51st & 9th	1	4	5

Problem 2 Revisited



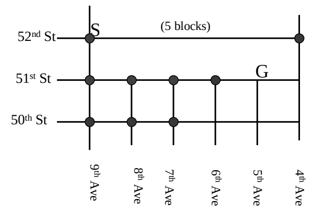
vertex	g(n)	h(n)	f(n)
52nd & 4th	5	2	7
51st & 8th	2	3	5
50th & 9th	2	5	7

Problem 2 Revisited



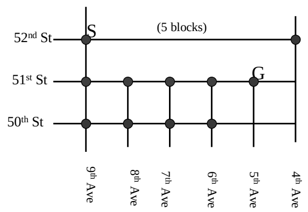
vertex	g(n)	h(n)	f(n)
52nd & 4th	5	2	7
51st & 7th	3	2	5
50th & 9th	2	5	7
50th & 8th	3	4	7

Problem 2 Revisited



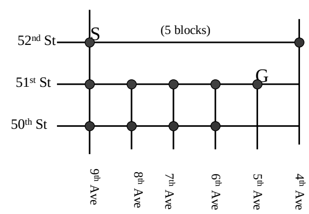
vertex	g(n)	h(n)	f(n)
52nd & 4th	5	2	7
51st & 6th	4	1	5
50th & 9th	2	5	7
50th & 8th	3	4	7
50th & 7th	4	3	7

Problem 2 Revisited



vertex	g(n)	h(n)	f(n)
52nd & 4th	5	2	7
51st & 5th	5	0	5
50th & 9th	2	5	7
50th & 8th	3	4	7
50th & 7th	4	3	7

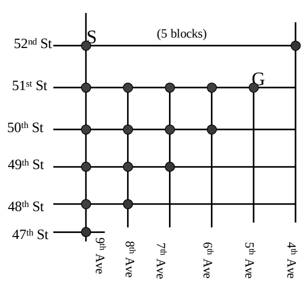
Problem 2 Revisited



vertex	g(n)	h(n)	f(n)
52nd & 4th	5	2	7
50th & 9th	2	5	7
50th & 8th	3	4	7
50th & 7th	4	3	7

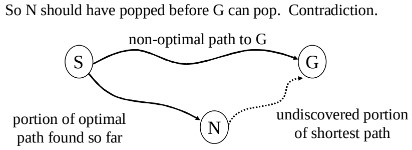
DONE!

What Would Dijkstra Have Done?



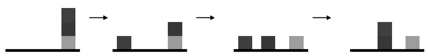
Proof of A* Optimality

- A* terminates when G is popped from the heap.
- Suppose G is popped but the path found isn't optimal:
  - priority(G) > optimal path length c
- Let P be an optimal path from S to G, and let N be the last vertex on that path that has been *visited but not yet popped*.
  - There must be such an N, otherwise the optimal path would have been found.
  - priority(N) = g(N) + h(N) ≤ c
- So N should have popped before G can pop. Contradiction.



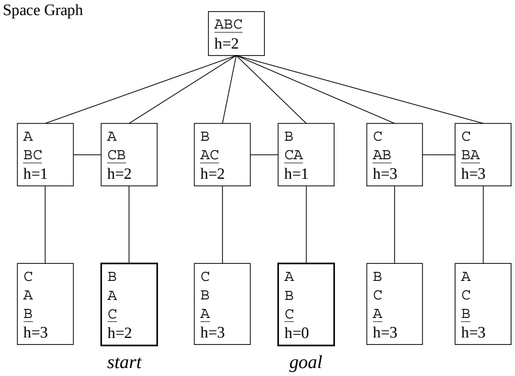
What About Those Blocks?

- "Distance to goal" is not always physical distance
- Blocks world:
  - distance = number of stacks to perform
  - heuristic lower bound = number of blocks out of place

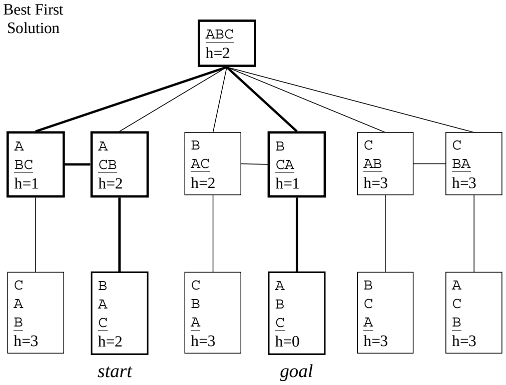


out of place = 1, true distance to goal = 3

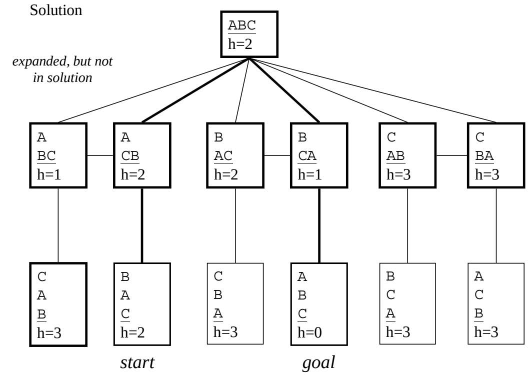
3-Blocks State Space Graph



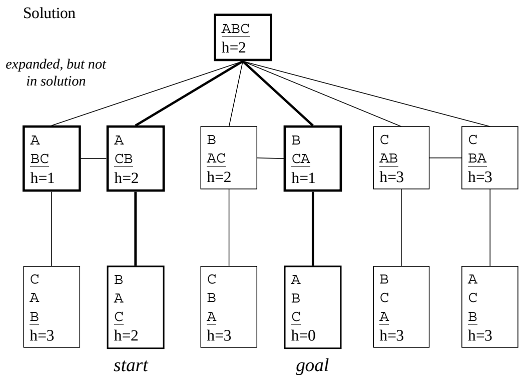
3-Blocks Best First Solution



3-Blocks BFS Solution



3-Blocks A* Solution



Maze Runner Demo

Other Real-World Applications

- Routing finding – computer networks, airline route planning
- VLSI layout – cell layout and channel routing
- Production planning – “just in time” optimization
- Protein sequence alignment
- Many other “NP-Hard” problems
  - A class of problems for which no exact polynomial time algorithms exist – so heuristic search is the best we can hope for

Importance of Heuristics

7	2	3
4	1	6
8	5	

- h1 = number of tiles in the wrong place
- h2 = sum of distances of tiles from correct location

D	IDS	A*(h1)	A*(h2)
2	10	6	6
4	112	13	12
6	680	20	18
8	6384	39	25
10	47127	93	39
12	364404	227	73
14	3473941	539	113
18		3056	363
24		39135	1641

A* STRIPS Planning

- Is there some general way to *automatically* create a heuristic for a given set of STRIPS operators?

1. Count number of false goal propositions in current state
   Admissible?
2. Delete all preconditions from actions, solve easier relaxed problem (why easier?), use length
   Admissible?
3. Delete negative effects from actions, solve easier relaxed problem, use length
   Admissible?

Planning as A* Search

- HSP (Geffner & Bonet 1999), introduced admissible “ignore negative effects” heuristic
- FF (Hoffman & Nebel 2000), used a modified non-admissible heuristic
  - Often dramatically faster, but usually non-optimal solutions found
  - Best overall performance AIPS 2000 planning competition

Search Algorithms

Backtrack Search

1. **DFS**
2. **BFS / Dijkstra’s Algorithm**
3. **Iterative Deepening**
4. **Best-first search**
5. **A***

Constraint Propagation

1. **Forward Checking**
2. **k-Consistency**
3. **DPLL & Resolution**

Local Search

1. **Hillclimbing**
2. **Simulated annealing**
3. **Walksat**

Guessing versus Inference

All the search algorithms we’ve seen so far are variations of guessing and backtracking

But we can reduce the amount of guesswork by doing more reasoning about the consequences of past choices

- **Example: planning a trip**

Idea:

- **Problem solving as constraint satisfaction**
- **As choices (guesses) are made, propagate constraints**

Map Coloring

CSP

- V is a set of variables v1, v2, …, vn
- D is a set of finite domains D1, D2, …, Dn
- C is a set of constraints C1, C2, …, Cm

  Each constraint specifies a *restriction* over *joint values* of a subset of the variables

E.g.:

v1 is Spain, v2 is France, v3 is Germany, …

Di = { Red, Blue, Green} for all i

For each adjacent vi, vj there is a constraint Ck

(vi,vj) ∈ { (R,G), (R,B), (G,R), (G,B), (B,R), (B,G) }

Variations

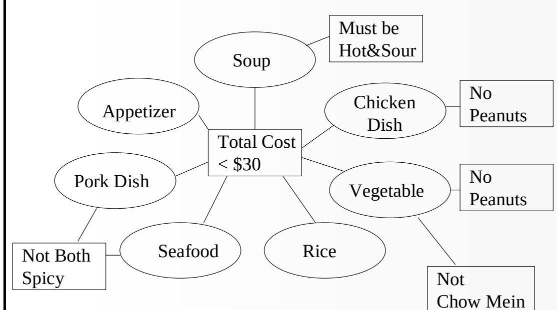
- Find a solution that satisfies all constraints
- Find all solutions
- Find a "tightest form" for each constraint

  (v1,v2) ∈ { (R,G), (R,B), (G,R), (G,B), (B,R), (B,G) }

  ➔

  (v1,v2) ∈ { (R,G), (R,B), (B,G) }
- Find a solution that minimizes some additional *objective function*

Chinese Dinner Constraint Network



Exploiting CSP Structure

Interleave inference and guessing

- At each *internal* node:
  - Select unassigned variable
  - Select a value in domain
  - Backtracking: try another value
    - Branching factor?
- At each node:
  - *Propagate Constraints*

Running Example: 4 Queens

Variables:

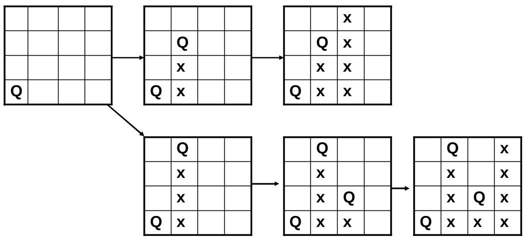
Q1 ∈ {1,2,3,4}
Q2 ∈ {1,2,3,4}
Q3 ∈ {1,2,3,4}
Q3 ∈ {1,2,3,4}

	Q		
			Q
Q			
		Q	

Constraints:

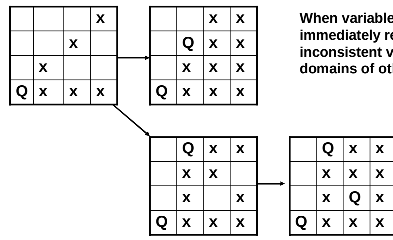
Q1	Q2
1	3
1	4
2	4
3	1
4	1
4	2

Constraint Checking



Takes 5 guesses to determine first guess was wrong

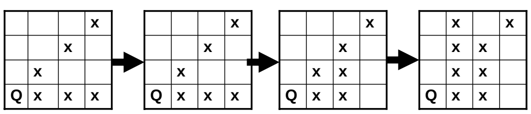
Forward Checking



When variable is set, immediately remove inconsistent values from domains of other variables

Takes 3 guesses to determine first guess was wrong

Arc Consistency



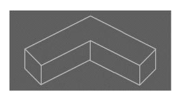
Iterate forward checking

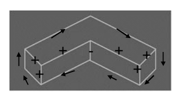
Propagations:

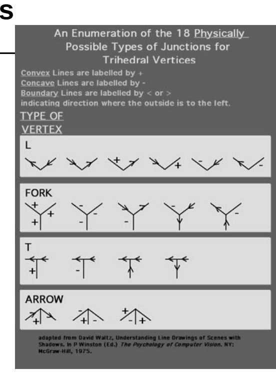
1. Q3=3 inconsistent with Q4 ∈ {2,3,4}
2. Q2=1 and Q2=2 inconsistent with Q3 ∈ {1}

Inference alone determines first guess was wrong!

Huffman-Clowes Labeling

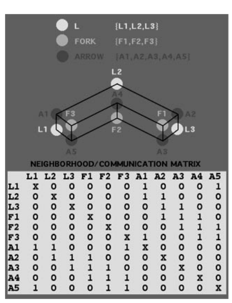






Waltz's Filtering: Arc-Consistency

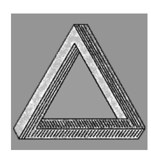
- **Lines: variables**
- **Conjunctions: constraints**
- **Initially Di = {+,-, ←, →)**
- **Repeat until no changes:**
  - **Choose edge (variable)**
  - **Delete labels on edge not consistent with both endpoints**

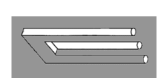


NEIGHBORHOOD/COMMUNICATION MATRIX

	L1	L2	L3	F1	F2	F3	A1	A2	A3	A4	A5
L1	X	0	0	0	0	0	1	0	0	0	1
L2	0	X	0	0	0	0	1	1	0	0	0
L3	0	0	X	0	0	0	0	1	1	0	0
F1	0	0	0	X	0	0	0	1	1	1	0
F2	0	0	0	0	X	0	0	0	1	1	1
F3	0	0	0	0	0	X	1	0	0	1	1
A1	1	1	0	0	0	1	X	0	0	0	0
A2	0	1	1	1	0	0	0	X	0	0	0
A3	0	0	1	1	1	0	0	0	X	0	0
A4	0	0	0	1	1	1	0	0	0	X	0
A5	1	0	0	0	1	1	0	0	0	0	X

No labeling!





Path Consistency

Path consistency (3-consistency):

- **Check every triple of variables**
- **More expensive!**
- **k-consistency:**

$|V|^k$ k-tuples to check

Worst case: each iteration eliminates 1 choice

$|D||V|$ iterations

$|D||V|^{k+1}$ steps! (But usually not this bad)

- ***n*-consistency: backtrack-free search**

Variable and Value Selection

- **Select variable with smallest domain – why?**
- **Which values to try first?**
- **Why different?**
- **Tie breaking?**

Variable and Value Selection

- **Select variable with smallest domain**
  - **Minimize branching factor**
  - **Most likely to propagate: most constrained variable heuristic**
- **Which values to try first?**
  - **Most likely value for solution**
  - **Least propagation! Least constrained variable**
- **Why different?**
  - **Every constraint must be eventually satisfied**
  - **Not every value must be assigned to a variable!**
- **Tie breaking?**
  - **In general randomized tie breaking best – less likely to get stuck on same bad pattern of choices**

N-queens Demo

Board size 15
Delay 6
Deterministic vs. Randomized tie breaking

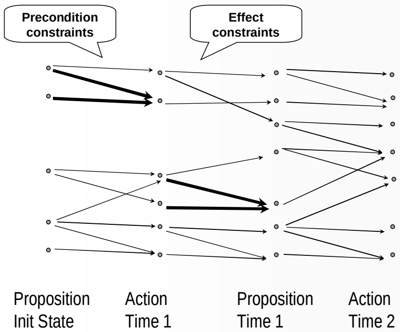
CSPs in the real world

- Scheduling Space Shuttle Repair
- Transportation Planning
- Computer Configuration
  - AT&T CLASSIC Configurator
    - #5ESS Switching System
    - Configuring new orders: 2 months → 2 hours

Planning as CSP

- Phase 1 - Convert **planning problem in a CSP**
  - **Choose a fixed plan length**
  - **Boolean variables**
    - **Action executed at a specific time point**
    - **Proposition holds at a specific time point**
  - **Constraints**
    - **Initial conditions true in first state, goals true in final state**
    - **Actions do not interfere**
    - **Relation between action, preconditions, effects**
- Phase 2 - Solution Extraction
  - Solve the CSP

Planning Graph Representation of CSP



Constructing the planning graph…

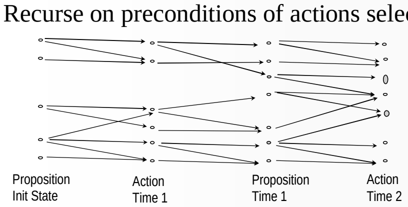
- Initial proposition layer
  – Just the initial conditions
- Action layer i
  – If all of an action's preconditionss are in i-1
  – Then add action to layer I
- Proposition layer i+1
  – For each action at layer i
  – Add all its effects at layer i+1

Mutual Exclusion

- Actions A,B ***exclusive (at a level)*** if
  – A deletes B's precondition, or
  – B deletes A's precondition, or
  – A & B have inconsistent preconditions
- Propositions P,Q ***inconsistent (at a level)*** if
  – All ways to achieve P exclude all ways to achieve Q
- Constraint propagation (arc consistency)
  – Can force variables to become true or false
  – Can create new mutexes

Solution Extraction

- For each goal G at time t
  – For some action A making G true @t
    - If A isn't mutex with a previously chosen action, select it
    - If no actions work, backup to last G (breadth first search)
- Recurse on preconditions of actions selected, t-1



Graphplan

- Create level 0 in planning graph
- Loop
  – If goal ⊆ contents of highest level (nonmutex)
  – Then search graph for solution
    - If find a solution then return and terminate
  – Else Extend graph one more level

A kind of double search: forward direction checks necessary (but insufficient) conditions for a solution, ... Backward search verifies...

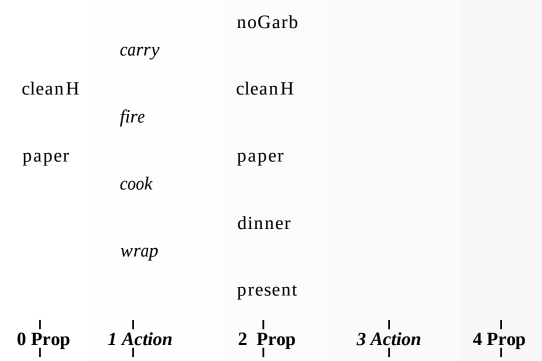
Dinner Date

<u>Initial Conditions</u>: (:and (cleanHands) (quiet))

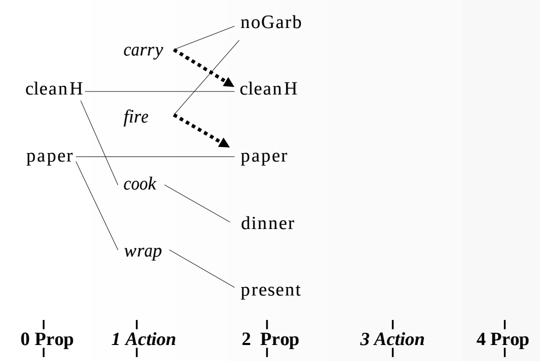
<u>Goal</u>: (:and (noGarbage) (dinner) (present))

<u>Actions</u>:
(:operator **carry** *:precondition*
:effect (:and (noGarbage) (:not (cleanHands)))
(:operator **fire** *:precondition*
:effect (:and (noGarbage) (:not (paper)))
(:operator **cook** *:precondition* (cleanHands)
:effect (dinner))
(:operator **wrap** *:precondition* (paper)
:effect (present))

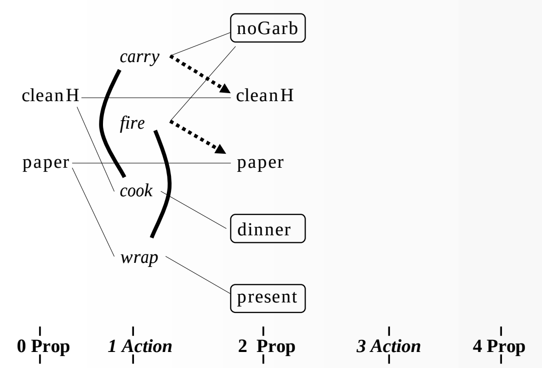
Planning Graph



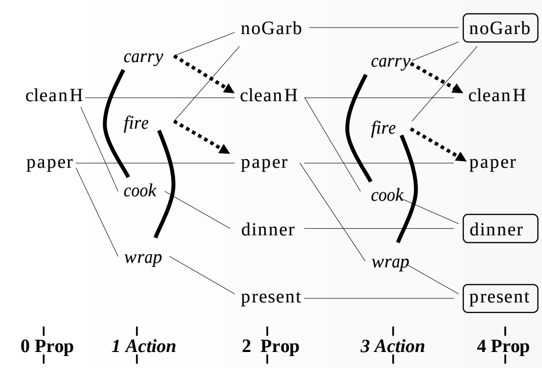
Are there any exclusions?



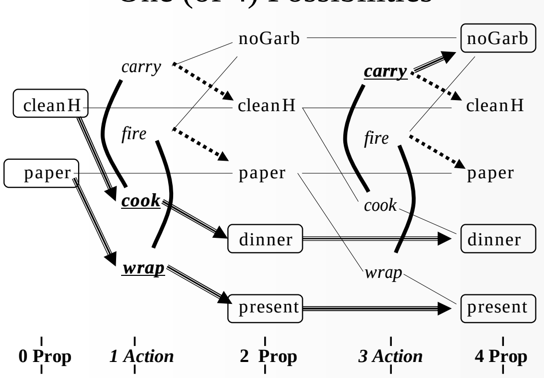
Do we have a solution?



Extend the Planning Graph



One (of 4) Possibilities



Summary Planning

- Reactive systems *vs.* planning
- Planners can handle medium to large-sized problems
- Relaxing assumptions
  - Atomic time
  - Agent is omniscient (no sensing necessary).
  - Agent is sole cause of change
  - Actions have deterministic effects
- Generating contingent plans
  - Large time-scale Spacecraft control

Coming Up

- Logical reasoning
- Planning as satisfiability testing
- Local search

- Start thinking about a project!