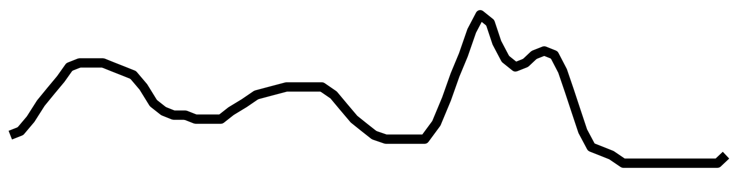
Alternative Search Methods: Hillclimbing, Cycle-checking, Best-First Heuristic Search, Constraint Propagation, etc.

Hill Climbing

- Simplest form of search
- No Storage Costs for cycle checking
- Here is the algorithm:
-   Look around and move in "best" direction
- REPEAT (UNTIL SATISFIED OR FOREVER)
-     add random noise to current solution
-     If new solution is "better" choose it
- else keep current solution

Geographical Metaphor

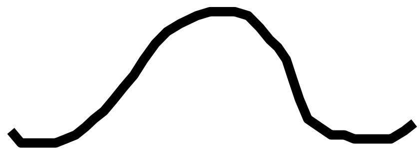
- A fitness function imposes a "landscape" over a problem space.
- What does it look like?
- How can we find the Highest Peak
- (with only a candle?)



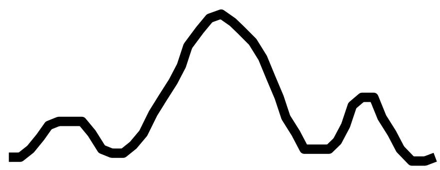
Hill Climbing is dependent on landscape imposed by fitness

- critically depends of shape of landscape
- useful for many problems
  - parameter spaces (lots of real numbers) with unknown topologies
- strictly local computation amenable to parallel processing

The Shape of Landscape is Important!



good for HC



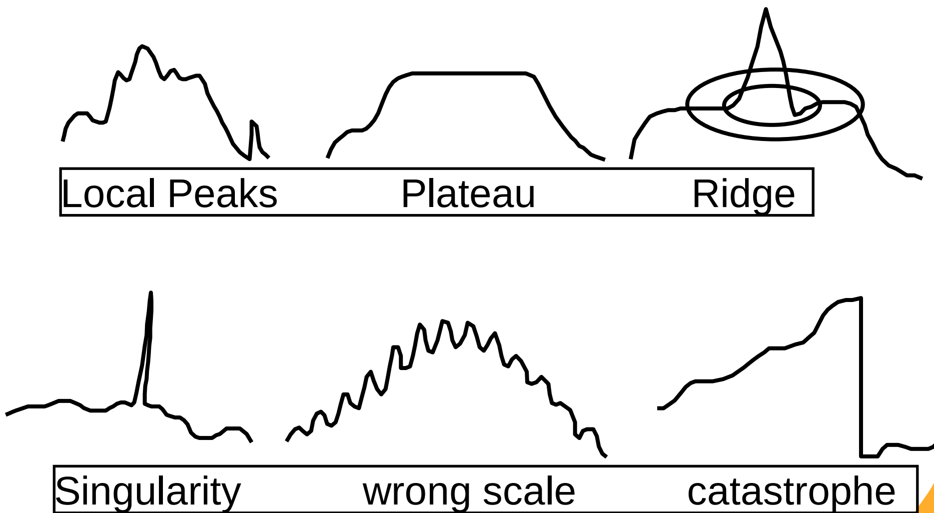
Bad for HC

How to get off local peaks?

The Shape of the Geography Affects use of Hill Climbing

- Problem Dependent
- Heuristic Dependent (e.g. sets landscape)
  - can be many local minima/maxima
- Many ad-hoc ways to solve local minima:
  - big jumps
  - restarts
  - recalibration of heuristic parameters
- If the space is abundant with solutions, and a best solution doesnt matter, HC is still useful

Bad Landscapes



Hill Climbing rejected early, Makes a comeback!

- Has not been considered particularly useful part of AI toolkit.
- Yet, it recently returned in New Forms we will study later in semester:
  - Genetic Algorithms
    - parallel hillclimbing with solution crossover
  - Neural Net learning
    - gradient descent over floating point weights

Heuristic Search: organized search can be improved with a heuristic function

- Instead of random walking or trying to hillclimb, basic DFS or BFS search keeps track of where you've been.
- Can we add the hill-climbing idea to organized search?
- We call this Heuristic Search
- you can add domain knowledge in a way to guide which node to expand.

Heuristic Search

- Is there an easy function which (approximately) evaluates states with respect to their "goodness", or nearness to goal?
- Why Approximate?
  - Because an exact measure means either problem is trivial, or function isn't easy!
- If so, can we use this information to speed up search?
  - Theoretically, No; Practically, Yes!

Imagine, if you will:

- Imagine a function for the Rubik's cube which accurately measured the distance of a configuration from "home".
- This gives a trivial "greedy" algorithm for solving the rubik's cube:
  - Look at all 12 neighbors
  - Pick one closer to the goal.
- QED perfect heuristics exist only for trivial problems.

Sliding Tile Puzzle

- Similar to the Rubik's cube, the Eight or 15 puzzle requires moves "out of the way" to move the state closer to the goal. Highly cyclical!

1	2	3
8		4
7	6	5

DFS with cycle checking

- (defun dfsc (nodes goalp nextf &optional been-there)
- "DEPTH FIRST SEARCH: list of init nodes, goal func, nextstate func"
- (cond ((null nodes) nil)
- ;;dont expand a node youve seen
- ((member (first nodes) been-there :test #'equal)
- (dfsc (rest nodes) goalp nextf been-there))
- ;; Return the first node if it is a goal node
- ((funcall goalp (first nodes)) (first nodes))
- ;; Put the children in the front of the list
- (t (dfsc (append (funcall nextf (first nodes))(rest nodes));use a stack
- goalp nextf (cons (first nodes) been-there)))))

BFS with cycle checking

```
(defun bfsc (nodes goalp nextf &optional states-so-far)
"BREADTH FIRST SEARCH: list of init nodes, goal func,
nextstate func, sofar"
  (cond ((null nodes) nil)
                    ;;check for cycles
             ((member (first nodes) states-so-far :test #'equal)
                  (bfsc (cdr nodes) goalp nextf states-so-far))
                ;; Return the first node if it is a goal node
                 ((funcall goalp (first nodes)) (first nodes))
               ;; Put the children in the back of the queue
             (t (bfsc (append (rest nodes) (funcall nextf (first nodes)))
                    goalp nextf (cons (first nodes) states-so-far)))))
```

Heuristic Functions "Domain Rules of Thumb"

- Quick to Compute, Approximate measure of goodness, used to guide search methods.

- 8-Puzzle: How many tiles in place?
- Hanoi: How many disks on right peg?

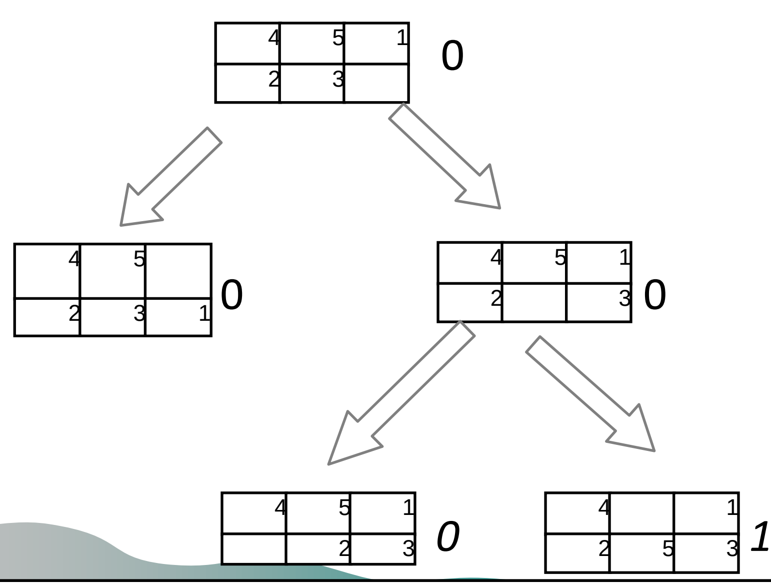
Basic Best-First Search

- Use a Heuristic Evaluation Function to order states-to-expand
- Keep states in a prioritized List (Heap or sorted list)

Heuristic computes "goodness" for any state of the problem

- can use decreasing or increasing to goal
- For the 8-puzzle:
  - Guess: counting tiles in position

Plateaus are no good!



Better Heuristic: Manhattan Distance

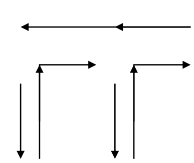
- Needs to be "informative"
- Need to be as "smooth" as possible across the space!
- Guess: Sum of "City Block" distance from goal for every tile.

- Remember: There is no perfect heuristic except for trivial problems!

Summed City Blocks
Assumes independence of interacting Subgoals

- How many moves to get each tile to goal if nothing were in the way?

4	5	1
2	3	



Goodness: 8

goal: 0

code for 5 puzzle (using sixtuples) (i'd rather do graycodes:)

```
(defparameter *moves*
 '((d r)(d r l)(d l)
   (u r)(u r l)(u l)))
(defparameter *goal* '(1 2 3 4 5 0))
(defparameter *delta*
  '((l . -1)(r . 1)(u . -3)(d . 3)))

(defun index (item list)
 (cond ((null list) 0)
       ((eq (car list) item) 0)
       (t (+ 1 (index item (cdr list))))))
```

sucessor function - puzmoves

```
(defun move (state m)
  (let* ((i (index 0 state))
        (j (+ i (cdr (assoc m *delta*)))))
    (loop for k from 0 to (- (length state) 1) collect
          (cond ((eq k i)(nth j state))
                ((eq k j)(nth i state))
                (t (nth k state))))))

(defun allmoves (state)
 (loop for m in (nth (index 0 state) *moves*) collect
(move state m)))
```

Manhattan Distance

```
(defun manhattan (position)
    ;;;uses *goal* rather than goalp
   (loop for x from 0 below (length position)
         sum (manhat (index x position)(index x *goal*))))

(defun manhat (p1 p2)
 (+ (abs (- (rem p1 3)(rem p2 3)))
   (abs (- (quotient p1 3)(quotient p2 3)))))

(defun quotient (a b) (floor (/ a b)))
```

Just add a sort to BFS

```
(defun bestfsc (nodes goalp nextf &optional states-so-far)
"best FIRST SEARCH: list of init nodes, goal func, nextstate func, sofar"
    ;;using wasteful sort instead of a heap, sorry.
      (setf nodes (sort nodes
                           #'(lambda (a b)(< (manhattan a)(manhattan b)))))
  (cond ((null nodes) nil)
                 ;;check for cycles.
                 ((member (first nodes) states-so-far :test #'equal)
                  (bestfsc (cdr nodes) goalp nextf states-so-far))
               ;; Return the first node if it is a goal node
                 ((funcall goalp (first nodes)) (first nodes))
               ;; Put the children in the back of the queue
                ;; use a heap insert instead of append please
                 (t (bestfsc (append (rest nodes) (funcall nextf (first nodes)))
                             goalp nextf (cons (first nodes) states-so-far)))))
```

Now we are ready for 8 puzzle

Random puzzle may be unsolveable so:

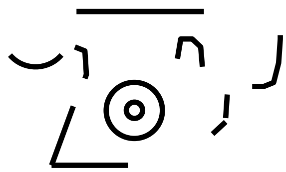
```
(defun pick (items) (nth (random (length items)) items))

(defun randmoves (state n)
  (if (zerop n) state
    (randmoves (pick (puzmoves state)) (- n 1))))

* (RANDMOVES *GOAL* 10)
(3 5 2 8 0 6 1 4 7)
```

Constraint Satisfaction

- Ubiquitous in Perception
- Form of "Expert" rapid problem solving
- Exploits Parallelism
- Often Difficult to Implement but beautiful when it works!

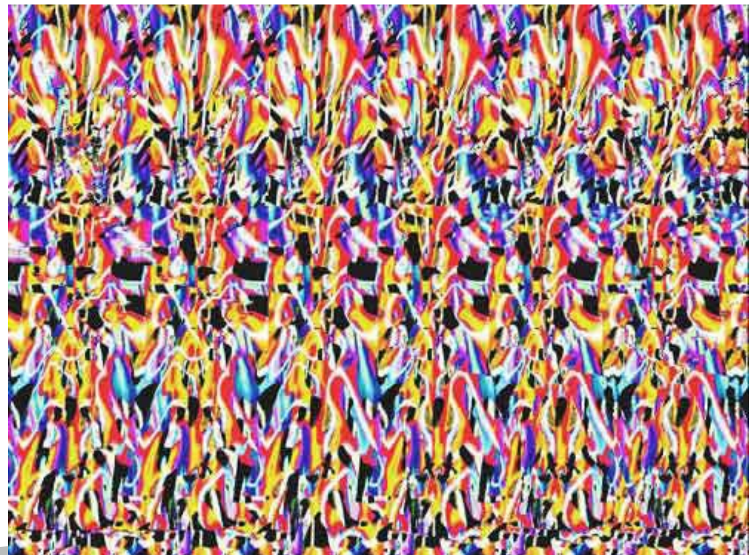
Constraint Satisfaction

- What is this a picture of?



What is this a picture of?

Magic Eye
(colorful random dot stereogram)



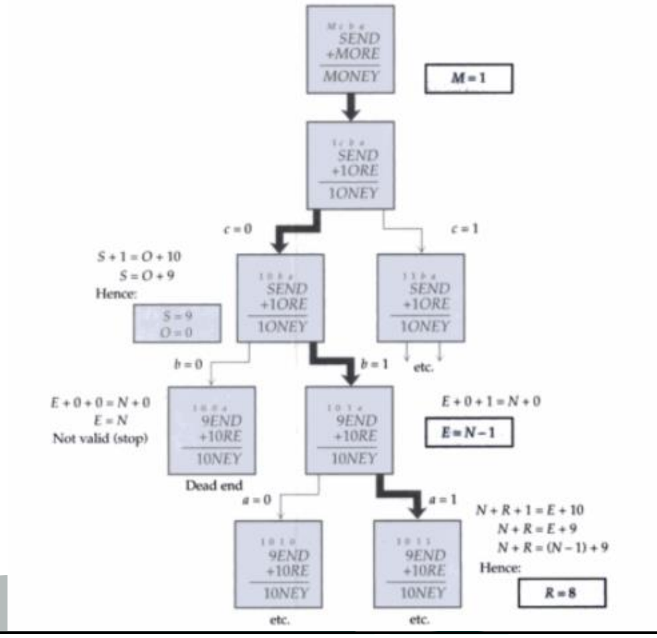
Example of CS Problems from protocol analysis
Cryptarithmetic

- Protocol Analysis lets humans solve problems and has them provide a running dialog of what they are doing.
- Revealed sophisticated patterns for some problems, e.g. Cryptarithmetic:

```
 SEND
+MORE
-------------
MONEY
```

- M must be 1
- O is 0 or 1
-   (but M is 1, so O is 0)
- QED: S must be 9.

constraint propagation



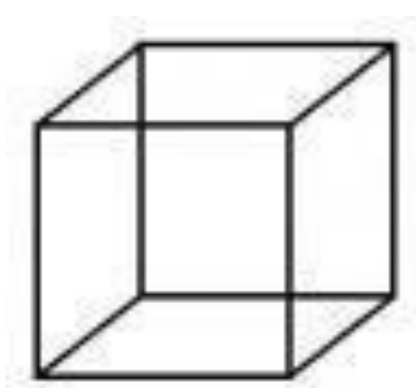
Sudoku Puzzle Family

8	7	6	9					
	1				6			
	4		3		5	8		
4						2	1	
	9		5					
	5			4		3		6
	2	9						8
		4	6	9		1	7	3
					1			4

Great AI Success in Constraint Propagaion

- Waltz Labelling
- Input is ambiguous 2D representation of 3D figure
- Output is representation of the 3d objects
- Simplifying Assumptions
- "Blocks world"
- Cleaned up, reduced input representation
- view should be stable w.r.t. slight rotations

Necker Cube



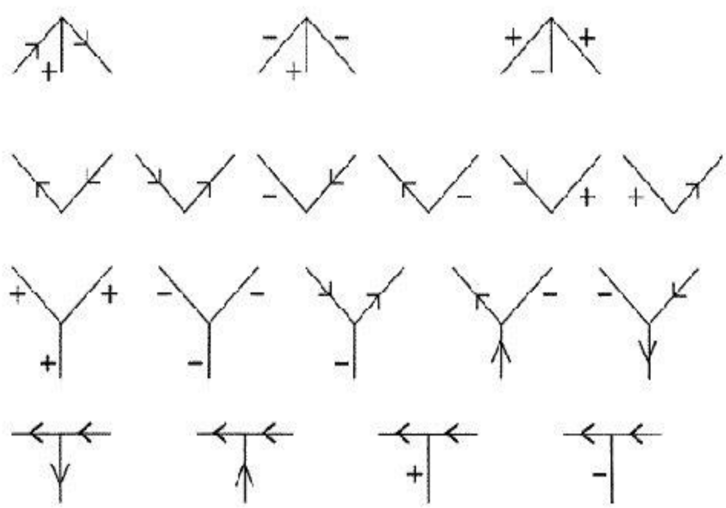
Defining the problem

- Input consists of a graph
  - links are lines in the picture
  - nodes are junctions in the picture
- Output is hopefully a labeling of each link as
  - Convex +
  - Concave -
  - Boundary ->- or -<- (interior on right)
- Labeling provides insight
  - No labeling --> impossible object
  - Multiple labelings --> ambiguity

Using Constraints from the World

- For a drawing with N lines, there are n*4 possible labellings
- However, in the case of blocksworld there are only certain junctions possible. :
  - L Junctions (6/16 varieties)
  - F (fork) junctions (5/64 possible)
  - T Junctions (4/64 possible)
  - W (Arrow) Junctions (3/64 possible)
- Possibilities are based on local examination from all perspectives: 18 out of a possible 204

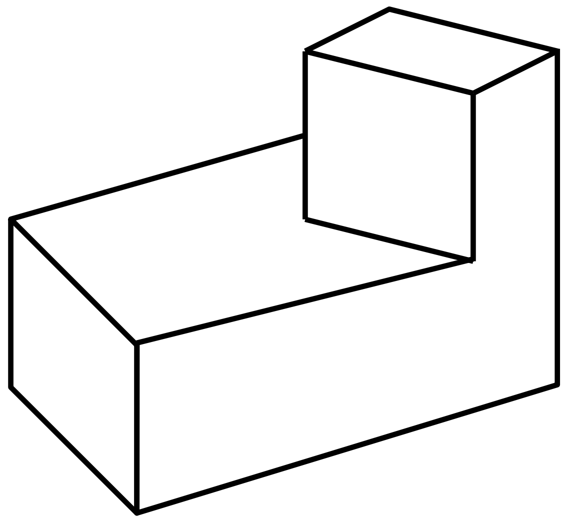
the legal vertices



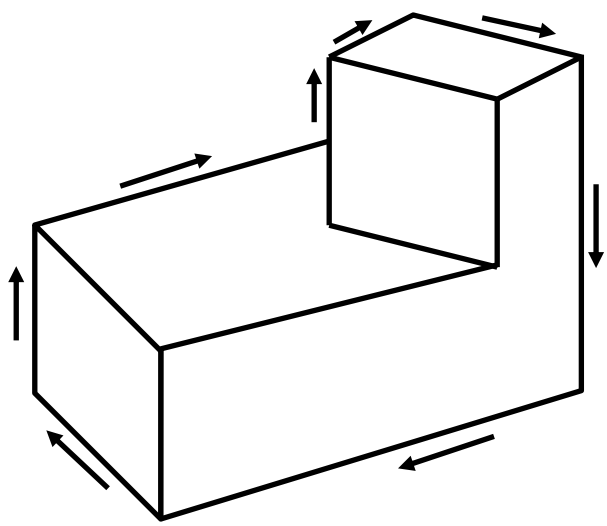
How to use them (in rapid search)

1. Make initial assumptions of grounding the external lines
2. Start with all possible labels on links and all possible junctions on nodes
3. Use adjacency to prune the possibilities

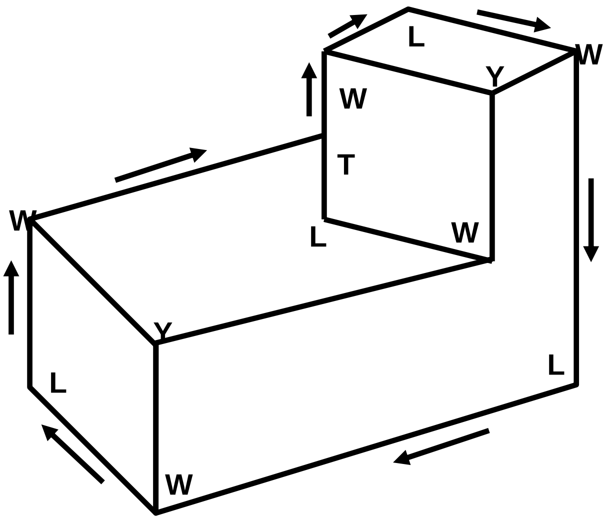
example



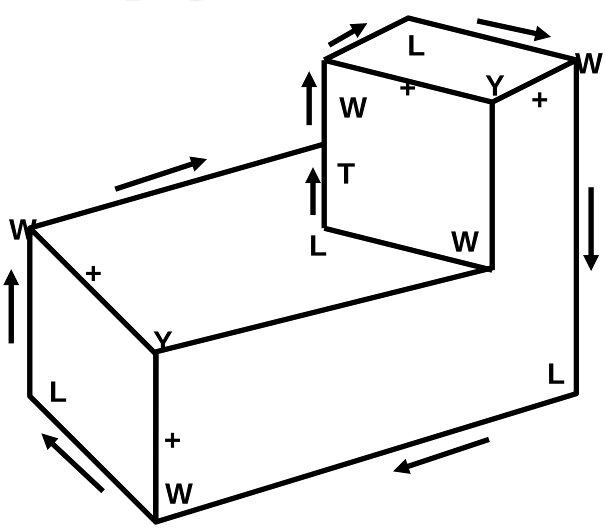
start with boundary



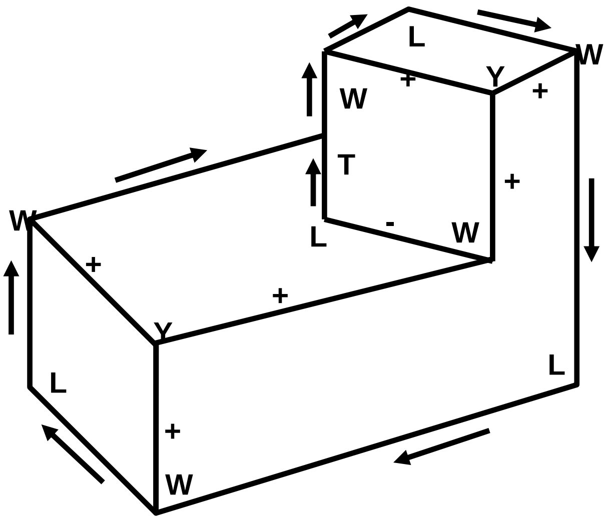
label vertices



Propagate Constraints



done!



History

- Guzman (1968) Defined Problem
- Huffman (1971),Clowes (1971), formalized simplified trihedral version
- Waltz (1975) Dealt with cracks, shadows, demonstrated drastic computation time savings from capturing constraints

Waltz's Version

- Allowed cracks, shadows, for 11 different labels (n*11)
- Dealt with 4 & 5 way junctions
- While # of unconstrained junctions can be 4*11 or 5*11actual physically allowable junctions were in the 100's
- Once cataloguing was accomplished, constraint program ran almost in linear time given the number of lines.

David Waltz taught at Brandeis from 1984 through 1993.