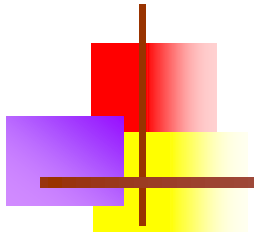
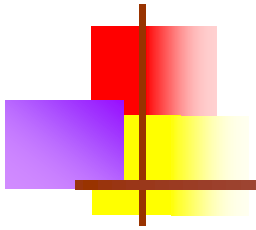
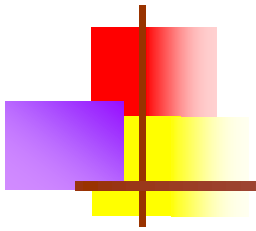


State-Space Searches



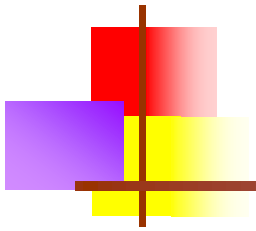
State spaces

- A state space consists of
  - A (possibly infinite) set of states
    - The start state represents the initial problem
    - Each state represents some configuration reachable from the start state
    - Some states may be goal states (solutions)
  - A set of operators
    - Applying an operator to a state transforms it to another state in the state space
    - Not all operators are applicable to all states
- State spaces are used extensively in Artificial Intelligence (AI)



Example 1: Maze

- A maze can be represented as a state space
  - Each state represents "where you are" in the maze
  - The start state represents your starting position
  - The goal state represents the exit from the maze
- Operators (for a rectangular maze) are: move north, move south, move east, and move west
  - Each operator takes you to a new state (maze location)
  - Operators may not always apply, because of walls in the maze



State Space: Water Jug Problem

"You are given two jugs, a 4-litre one and a 3-litre one. Neither has any measuring markers on it. There is a pump that can be used to fill the jugs with water. How can you get exactly 2 litres of water into 4-litre jug."

State Space: Water Jug Problem

- State: (x, y)

  x = 0, 1, 2, 3, or 4 y = 0, 1, 2, 3

- Start state: (0, 0).
- Goal state: (2, n) for any n.
- Attempting to end up in a goal state.

State Space: Water Jug Problem

1. $(x, y) \rightarrow (4, y)$
   if $x < 4$
2. $(x, y) \rightarrow (x, 3)$
   if $y < 3$
3. $(x, y) \rightarrow (x - d, y)$
   if $x > 0$
4. $(x, y) \rightarrow (x, y - d)$
   if $y > 0$

State Space: Water Jug Problem

5. $(x, y) \rightarrow (0, y)$
   if $x > 0$
6. $(x, y) \rightarrow (x, 0)$
   if $y > 0$
7. $(x, y) \rightarrow (4, y - (4 - x))$
   if $x + y \geq 4$, $y > 0$
8. $(x, y) \rightarrow (x - (3 - y), 3)$
   if $x + y \geq 3$, $x > 0$

State Space: Water Jug Problem

9. $(x, y) \rightarrow (x + y, 0)$
   if $x + y \leq 4$, $y > 0$

10. $(x, y) \rightarrow (0, x + y)$
    if $x + y \leq 3$, $x > 0$

11. $(0, 2) \rightarrow (2, 0)$

12. $(2, y) \rightarrow (0, y)$

State Space Search: Water Jug Problem

1. current state = (0, 0)
2. Loop until reaching the goal state (2, 0)
   - Apply a rule whose left side matches the current state
   - Set the new current state to be the resulting state

(0, 0)
(0, 3)
(3, 0)
(3, 3)
(4, 2)
(0, 2)
(2, 0)

Example 2: The 15-puzzle

Start state:

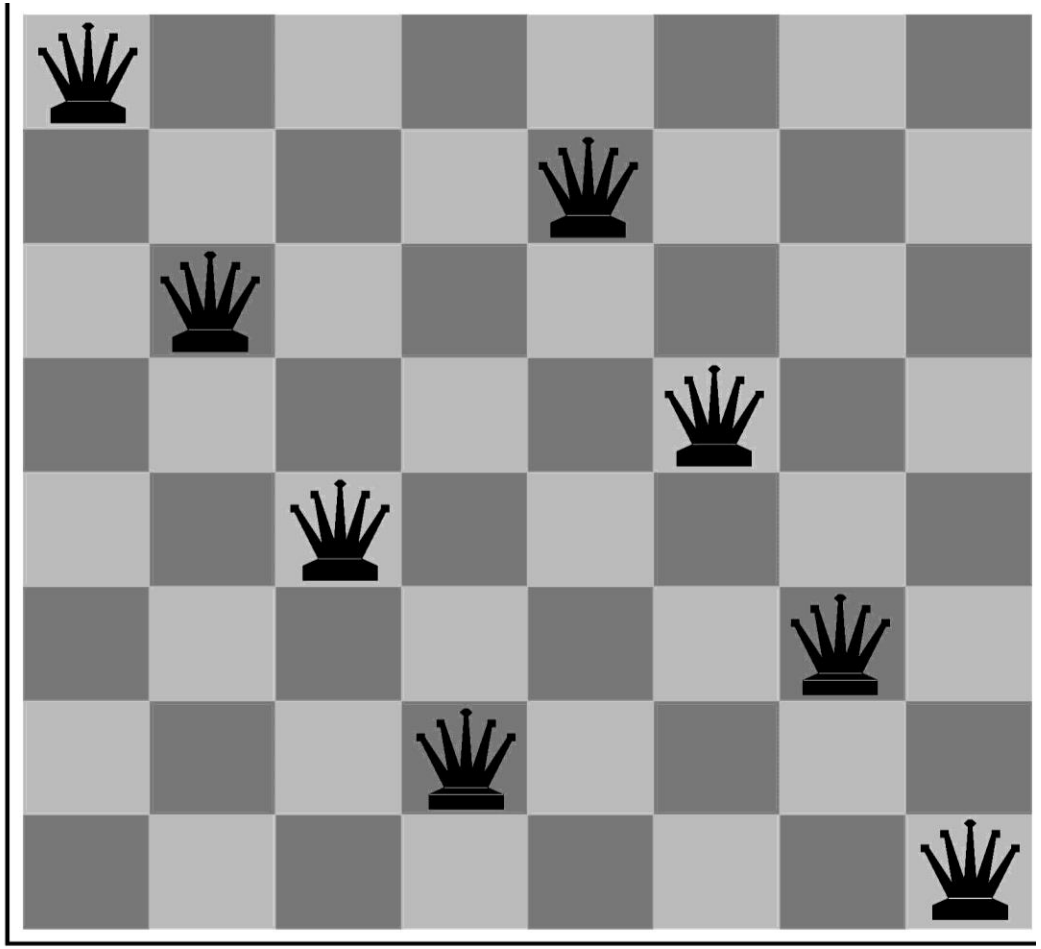
3	10	13	7
9	14	6	1
4		15	2
11	8	5	12

Goal state:

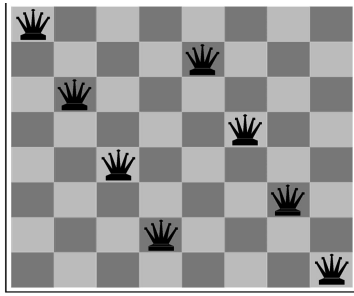
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

- The start state is some (almost) random configuration of the tiles
- The goal state is as shown
- Operators are
  - Move empty space up
  - Move empty space down
  - Move empty space right
  - Move empty space left
- Operators apply if not against edge

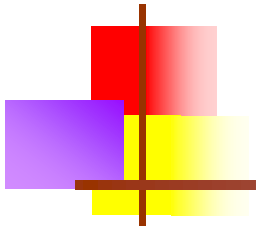
Example 2: 8 Queen Problem



State-Space problem formulation



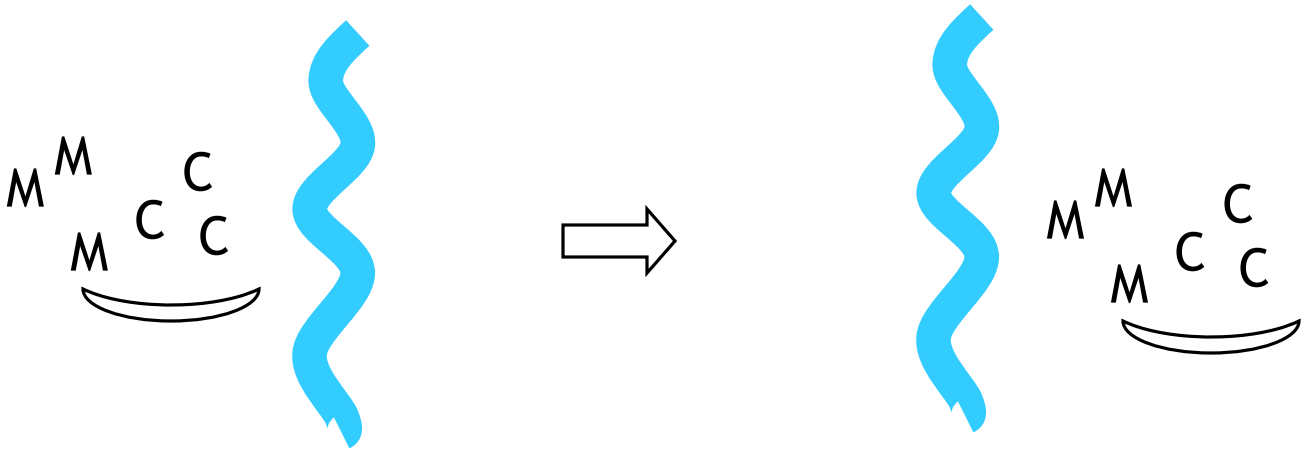
- states? -any arrangement of n<=8 queens
  -*or* arrangements of n<=8 queens in leftmost n columns, 1 per column, such that no queen attacks any other.
- initial state? no queens on the board
- actions? -add queen to any empty square
  -*or* add queen to leftmost empty square such that it is not attacked by other queens.
- goal test? 8 queens on the board, none attacked.
- path cost? 1 per move



- Formuation 1
  - Operators:Add a queen in any square
- Formuation 2
  - Place the queen in the leftmost empty column
- Formuation 3

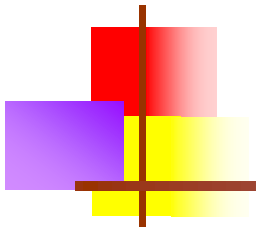
Example 3: Missionaries and cannibals

- An old puzzle is the "Missionaries and cannibals" problem (in various guises)
- The missionaries and cannibals wish to cross a river
- They have a canoe that can hold two people
- It is unsafe to have cannibals outnumber missionaries



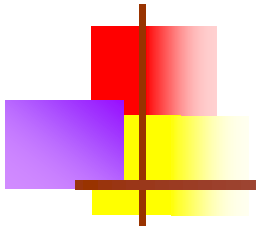
Initial state

Goal state



States

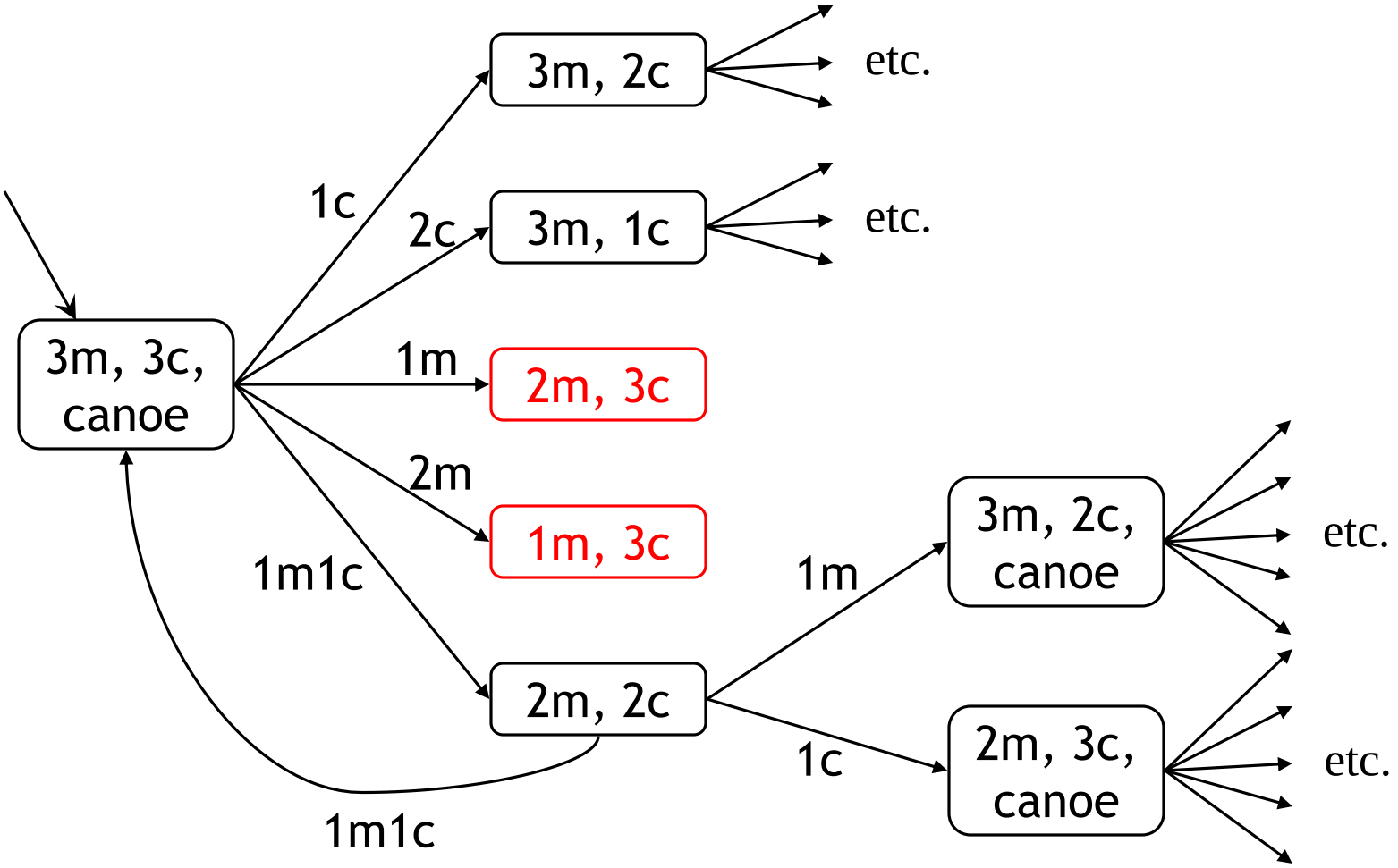
- A *state* can be represented by the number of missionaries and cannibals on each side of the river
  - Initial state: 3m,3c,canoe / 0m,0c
  - Goal state: 0m,0c / 3m,3c,canoe
  - We assume that crossing the river is a simple procedure that always works (so we don't have to represent the canoe being in the middle of the river)
- However, this is redundant; we only need to represent how many missionaries/cannibals are on *one* side of the river
  - Initial state: 3m,3c,canoe
  - Goal state: 0m,0c

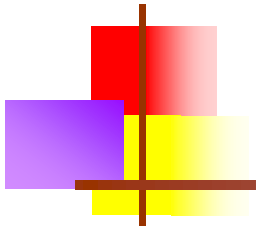


Operations

- An *operation* takes us from one state to another
- Here are five possible operations:
  - Canoe takes 1 missionary across river (1m)
  - Canoe takes 1 cannibal across river (1c)
  - Canoe takes 2 missionaries across river (2m)
  - Canoe takes 2 cannibals across river (2c)
  - Canoe takes 1 missionary and 1 cannibal across river (1m1c)
- We don't have to specify "west to east" or "east to west" because only one of these will be possible at any given time

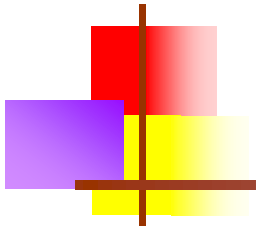
The state space





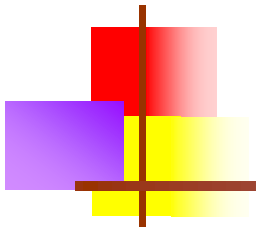
Example 3, revisited

- 3 missionaries, 3 cannibals, 1 canoe
- The canoe can hold at most two people
- Cannibals may never outnumber missionaries (on either side)
- Initial state is (3, 3, 1), representing the number of missionaries, cannibals, boats on the *initial* side
- The goal state is (0, 0, 0)
- Operators are addition or subtraction of the vectors
  (1 0 1), (2 0 1), (0 1 1), (0 2 1), (1 1 1)
- Operators apply if result is between (0 0 0) and (3 3 1)



State-space searching

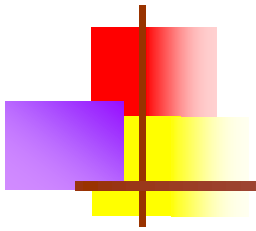
- Most problems in AI can be cast as searches on a state space
- The space can be tree-shaped or graph-shaped
  - If a graph, need some way to keep track of where you have been, so as to avoid loops
- The state space is often very, very large
- We can minimize the size of the search space by careful choice of operators
- Exhaustive searches don't work—we need *heuristics*



The basic search algorithm

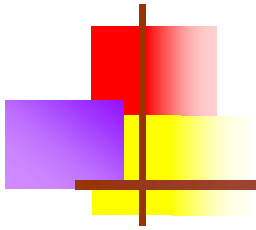
```
Initialize: put the start node into OPEN
while OPEN is not empty
    take a node N from OPEN
    if N is a goal node, report success
    put the children of N onto OPEN
Report failure
```

- If OPEN is a stack, this is a depth-first search
- If OPEN is a queue, this is a breadth-first search
- If OPEN is a *priority queue*, sorted according to *most promising first*, we have a best-first search



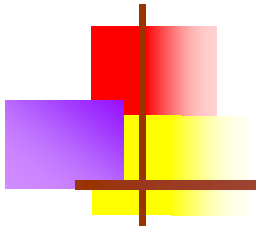
Heuristic searching

- All the previous searches have been blind searches
  - They make no use of any knowledge of the problem
  - If we know *something* about the problem, we can usually do much, much better
- Example: 15-puzzle
  - For each piece, figure out how many moves away it is from its goal position, if no other piece were in the way
  - The total of these gives a measure of distance from goal
  - This is a *heuristic* measure



Heuristics

- A heuristic is a rule of thumb for deciding which choice might be best
- There is no general theory for finding heuristics, because every problem is different
- Choice of heuristics depends on knowledge of the problem space

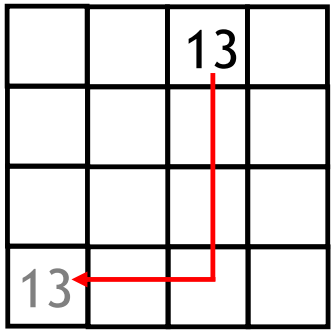


Best-first searching

- Use the same basic search algorithm
- Choose from OPEN the “best” node, that is, the one that *seems to be* closest to a goal
- Generally, even very poor heuristics are significantly better than blind search, but...
- No guarantee that the best path will be found
- No guarantee on the space requirements

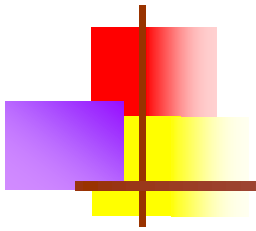
The 15-puzzle again

- Consider one piece in the 15-puzzle
  - If nothing were in the way, how many moves would it take to get this piece to where it belongs?



- This piece will have to be moved *at least* that many times to get it to where it belongs
- If we do this for every piece, and add up the moves, we get a (very) optimistic measure of how many moves it will take to solve the puzzle

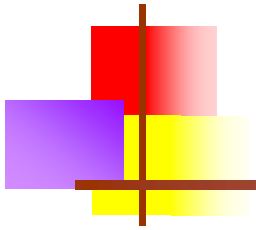
- Suppose, from a given position, we try every possible single move (there can be up to four of them), and pick the move with the smallest sum
- This is a reasonable *heuristic* for solving the 15-puzzle



Iterative deepening

Set LIMIT to zero
do forever
    Do a depth-first search up to LIMIT levels deep
    If a goal is found, return success,
    else add 1 to LIMIT

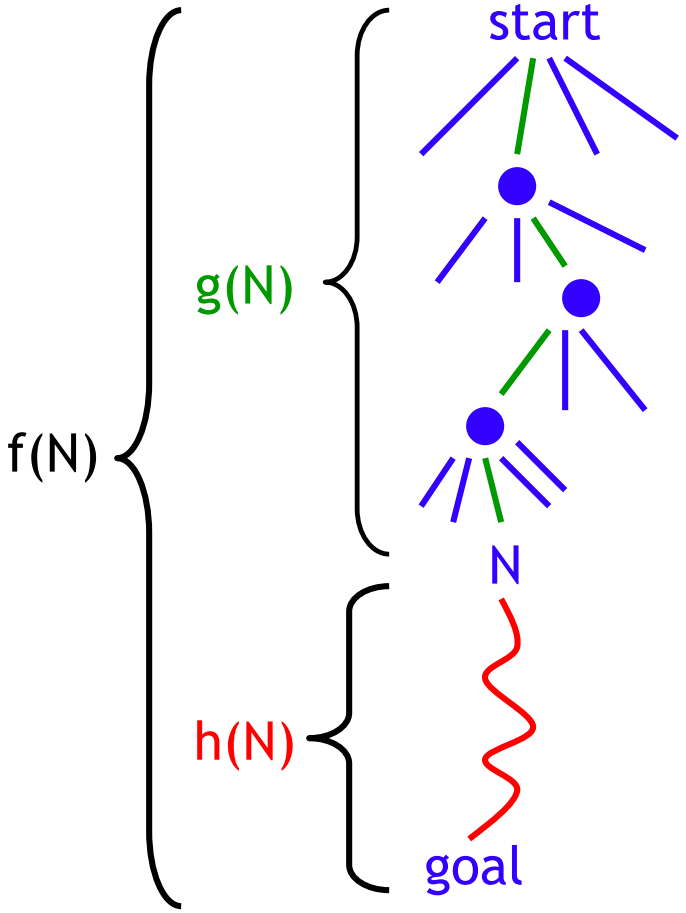
- Each time through the loop we *start all over!*
- If we find a path, it will be a shortest possible path
- Only requires linear space, because it only uses DFS
- Increased time requirements are only *linear*



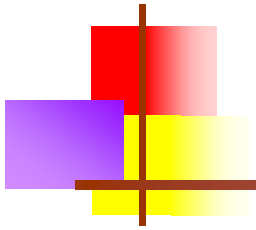
The A* algorithm

- Suppose:
  - You keep track of the distance g(N) that each node N that you visit is from the start state
  - You have some heuristic function, h(N), that estimates the distance between node N and a goal
- Then:
  - f(N) = g(N) + h(N) gives you the (partially estimated) distance from the start node to a goal node
- The A* algorithm is: choose from OPEN the node N with the smallest value of f(N)

The A* formula: f(N) = g(N) + h(N)

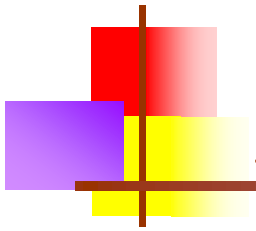


- g(N) is the (known) distance from start to N
- h(N) is the (estimated) distance from N to a goal
- f(N) is just the sum of these
- f(N) is our best guess as to the distance from start to a goal, passing through N



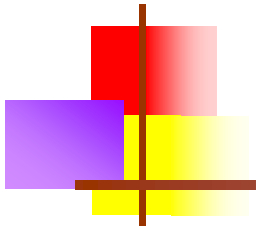
How good is A*?

- Memory usage depends on the heuristic function
  - If h(N) = *constant*, then A* = breadth-first
  - If h(N) is a *perfect* estimator, A* goes straight to goal
  - ...but if h(N) were perfect, why search?
- Quality of solution also depends on h(N)
- It can be proved that, if h(N) is *optimistic* (never overestimates the distance to a goal), then A* will find a best solution (shortest path to a goal)



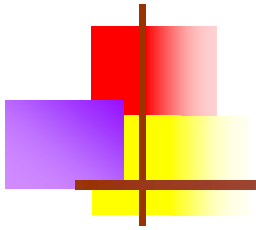
A* applied to the 15-puzzle

- Remember, if h(N) is *optimistic* (never overestimates the distance to a goal), then A* will find a best solution (shortest path to a goal)
- Our heuristic for the 15-puzzle was optimistic
- Therefore, using A* will find a solution with the fewest possible moves



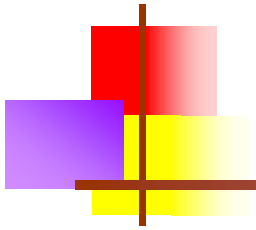
IDA*

- A* may require exponential storage
- Solution: *Iterative-deepening A* (IDA*)*
  - Just like Iterative Deepening, but...
  - ...instead of using g(N) (the actual depth so far) to cut off searching...
  - ...use f(N) (the estimated *total* depth)
- IDA* gives the same results as A*
- Since IDA* is basically a depth-first search, storage requirements are linear in length of path



Conclusion

- Many (or most) problems in AI can be represented as state-space searches
- The best searches combine a basic blind search technique with heuristic knowledge about the problem space
- A* and its variations, especially IDA*, are the best heuristic search techniques known



The End