CMU 15-781

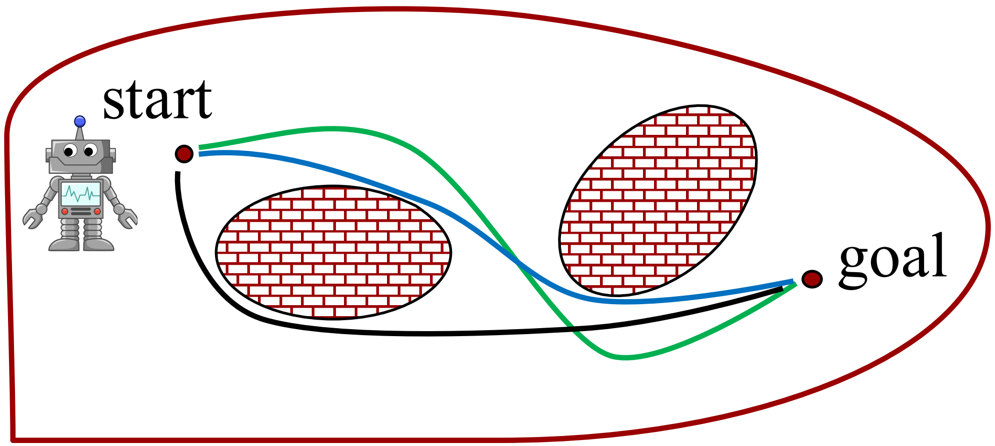
Lecture 5:
Planning I

Teacher:
Gianni A. Di Caro

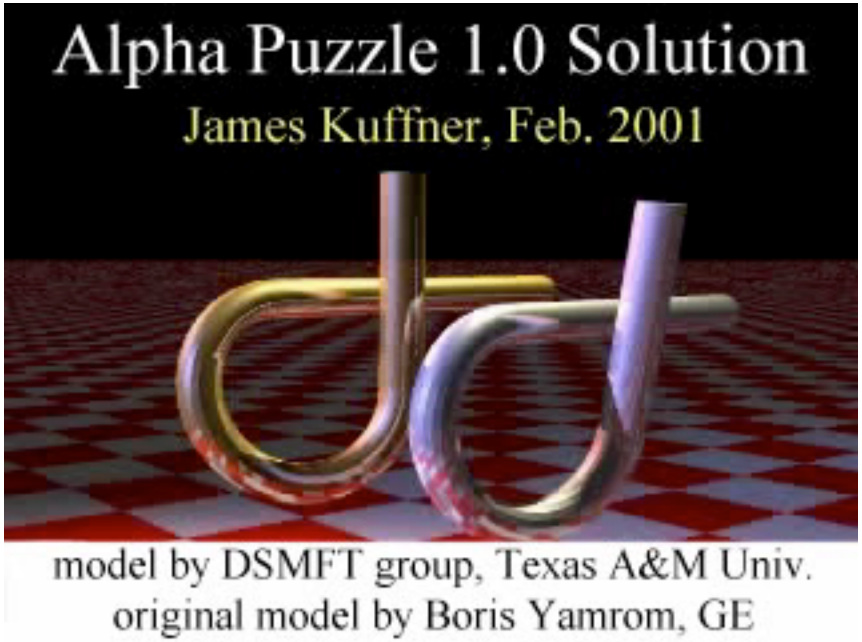
Motion Planning → Search Problem

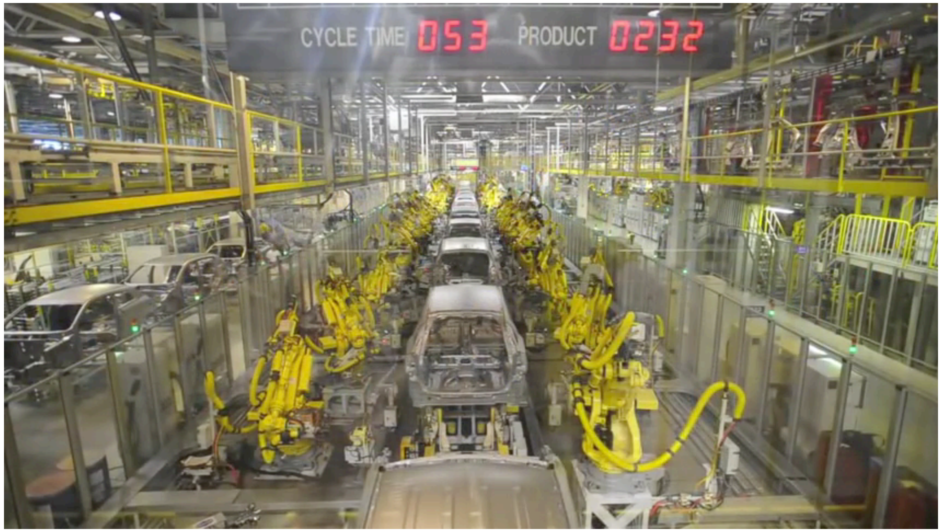
- **Path planning:** computing a continuous sequence ("a path") of configurations (states) between an initial configuration (start) and a final configuration (goal)
  - Respecting *constraints* (e.g., avoiding obstacles, physical limitations in rotations and translations)
  - Optimizing *metrics* (length, energy, time, smoothness, ...)

- **Motion planning:** pp + time parameter
- **Trajectory planning:** pp + velocity profile



Motion Planning Examples





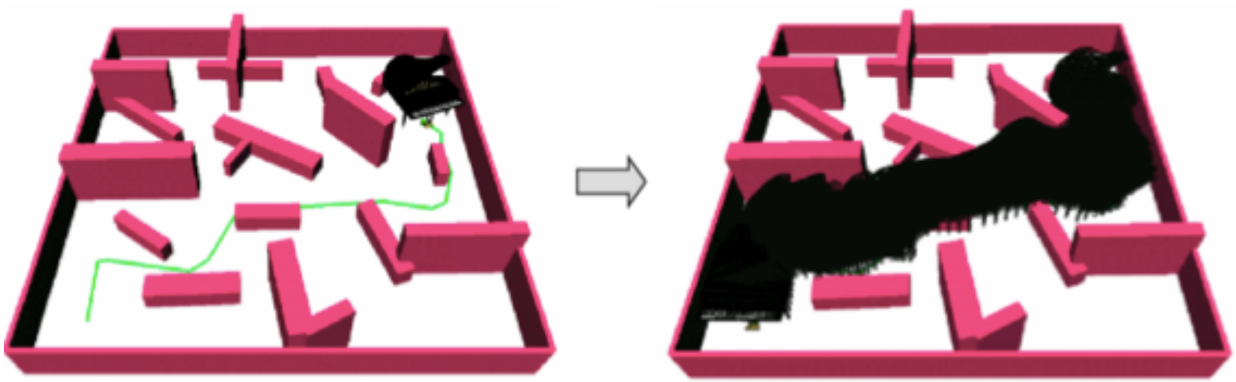
Kia car factory

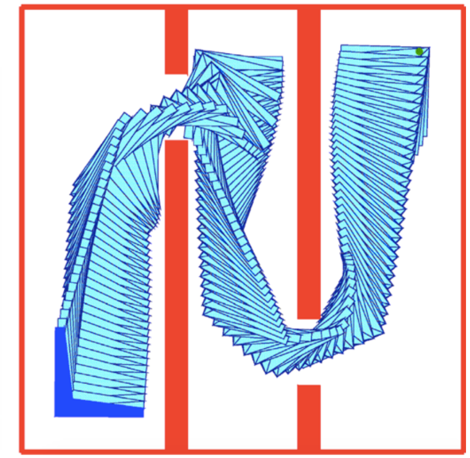
Motion Planning Examples

Baldur's gate

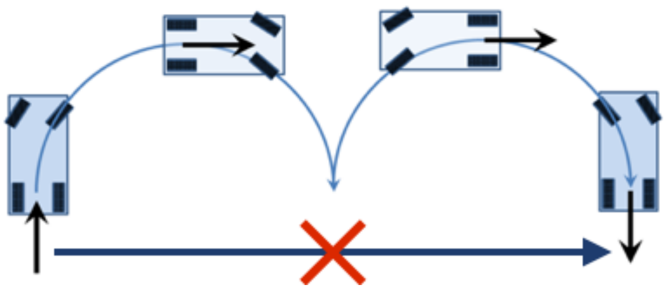
Motion Planning Issues

Free world space ≠ Accessible to the "agent"

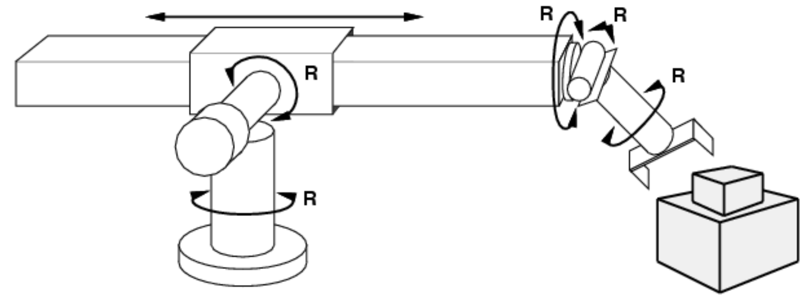




Non-holonomic constraints



States = Configurations + World



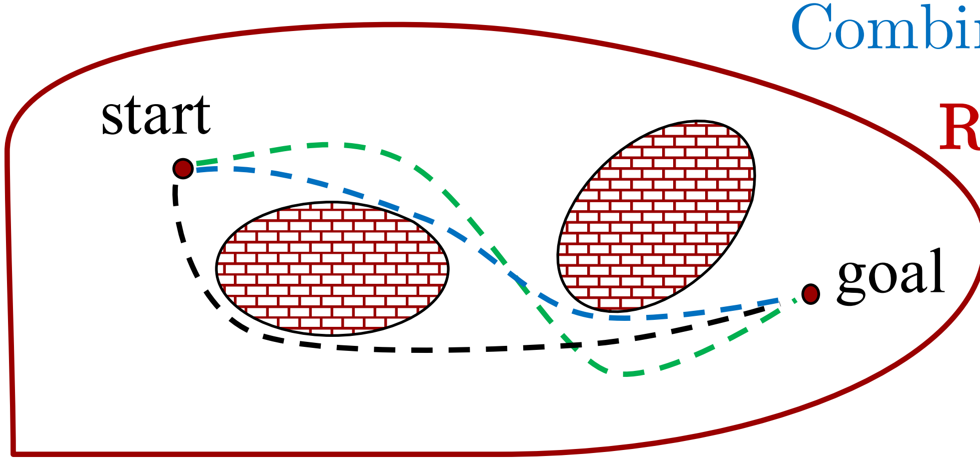
...3D physical spaces, n-dimensional states, continuous spaces ...

SIMPLIFIED (BUT USEFUL!) SETTINGS

- Let's consider an omnidirectional **point** "agent"
- Let's discretize the free world representing it into a **graph**
- Let's search for a (discrete) path in the graph
- Let the world be *static*
- Let the cost be the *length* of the path
- Let's forget about *time* and *velocity*

Combinatorial planning

Road map graph

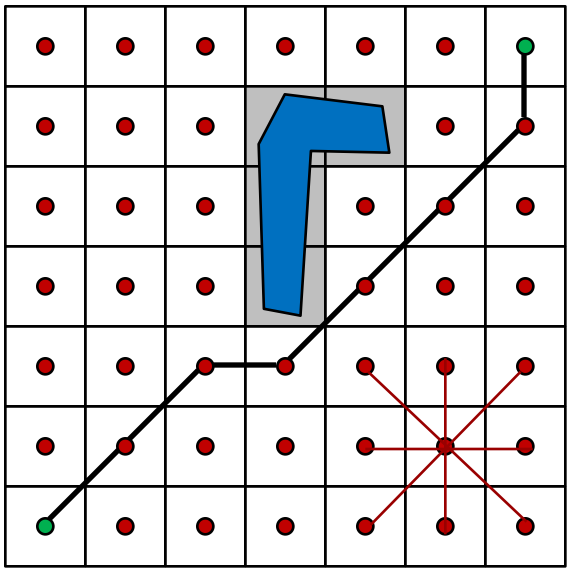


Cell Decomposition

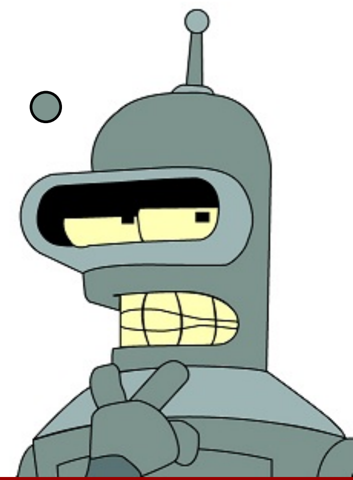
- The world is covered by a *discrete set of cells*
- First guess: a grid
- Mark cells that *intersect* obstacles as **blocked**, **free** otherwise
- The motion through a cell happens through its *center*
- Each cell has n=8 neighbors
- Find path through centers of free cells

Which are the nodes and the edges of the road map graph?

The shortest path is the optimal path *on the graph!*

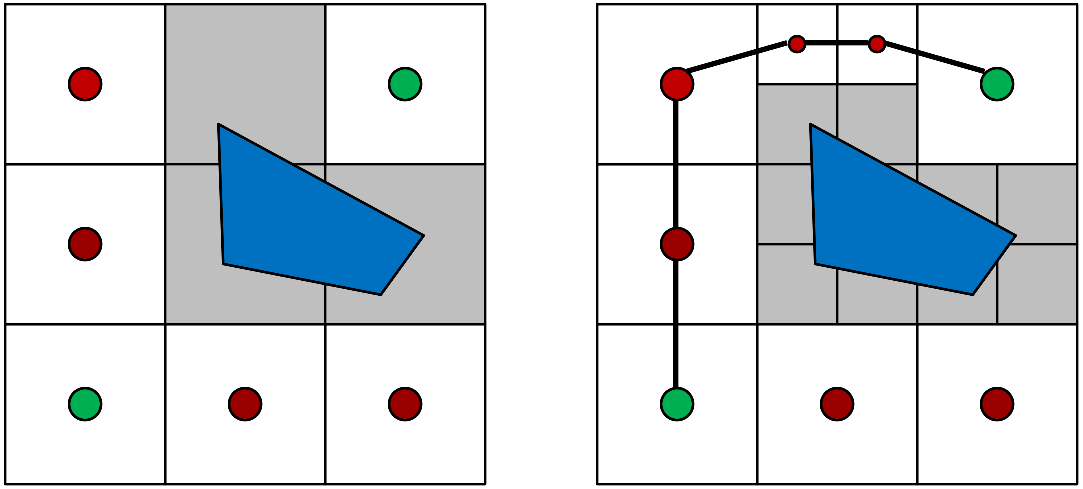


Is this approach complete?
Optimal?



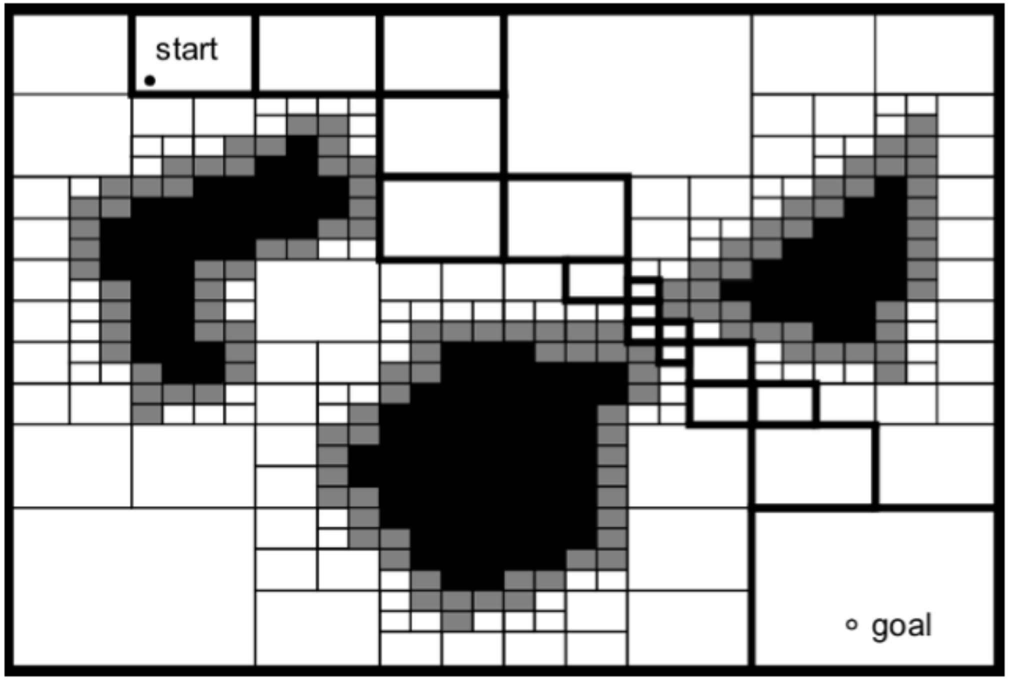
Iterative Cell Decomposition

- Distinguish between
  - Cells that are *fully contained* in obstacles
  - Cells that intersect obstacles
- If no path found, subdivide the mixed cells



Quadtree Cell Decomposition

- Doing the decomposition in a smart way, save on states
- Any n-tree decomposition can be used (quad- and oct-trees are widely used)

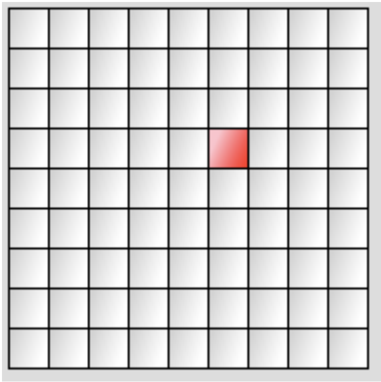


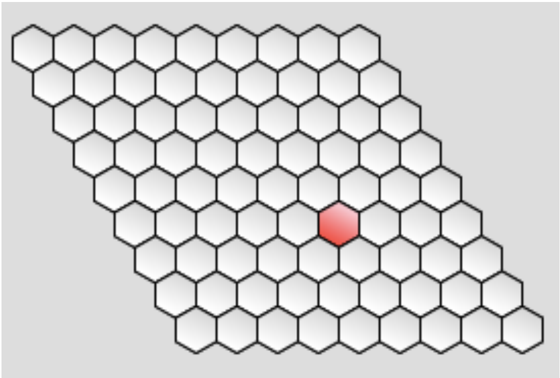
Is it complete now?

- An algorithm is resolution complete when:
  a. If a path exists, it finds it in finite time
  b. If a path does not exist, it returns in finite time
- Poll 1: Cell decomposition satisfies:
  1. a but not b
  2. b but not a
  3. Both a and b
  4. Neither a nor b

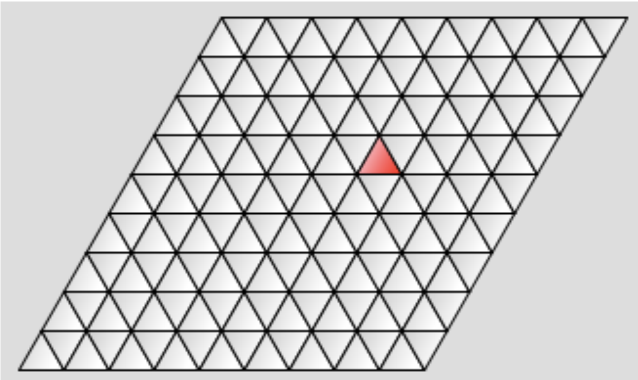
Cell shapes and Path execution

- The cell sequence provides a feasible path, however navigation inside a cell and between cells can be done in many different ways (*path execution*)
- Cells can have different *shapes*





Less distortion of distances

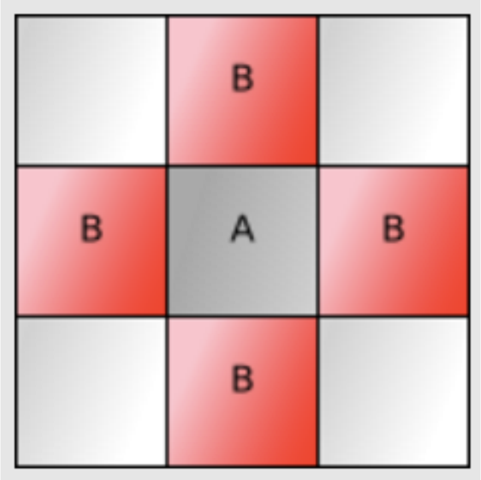


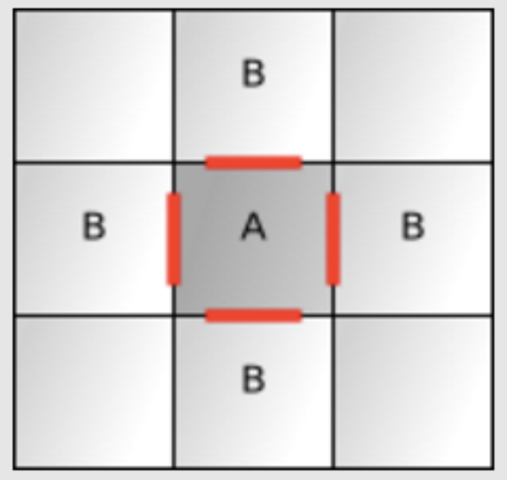
Small area / Large perimeter

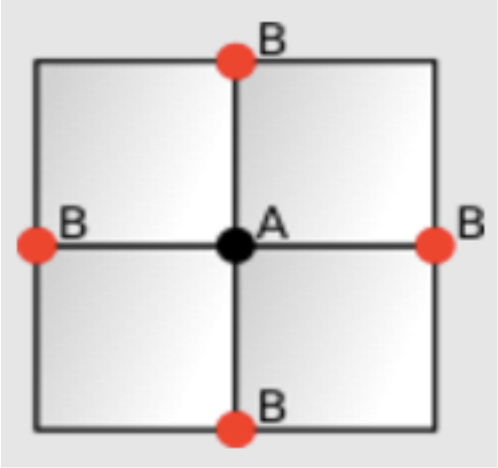
Figures from Amit Patel

Cell shapes and Path execution

- Cells' centers can be replaced by edges or vertices





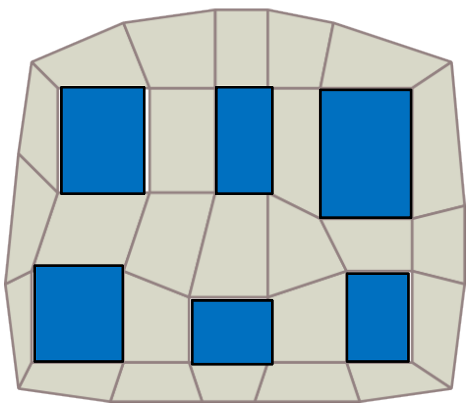


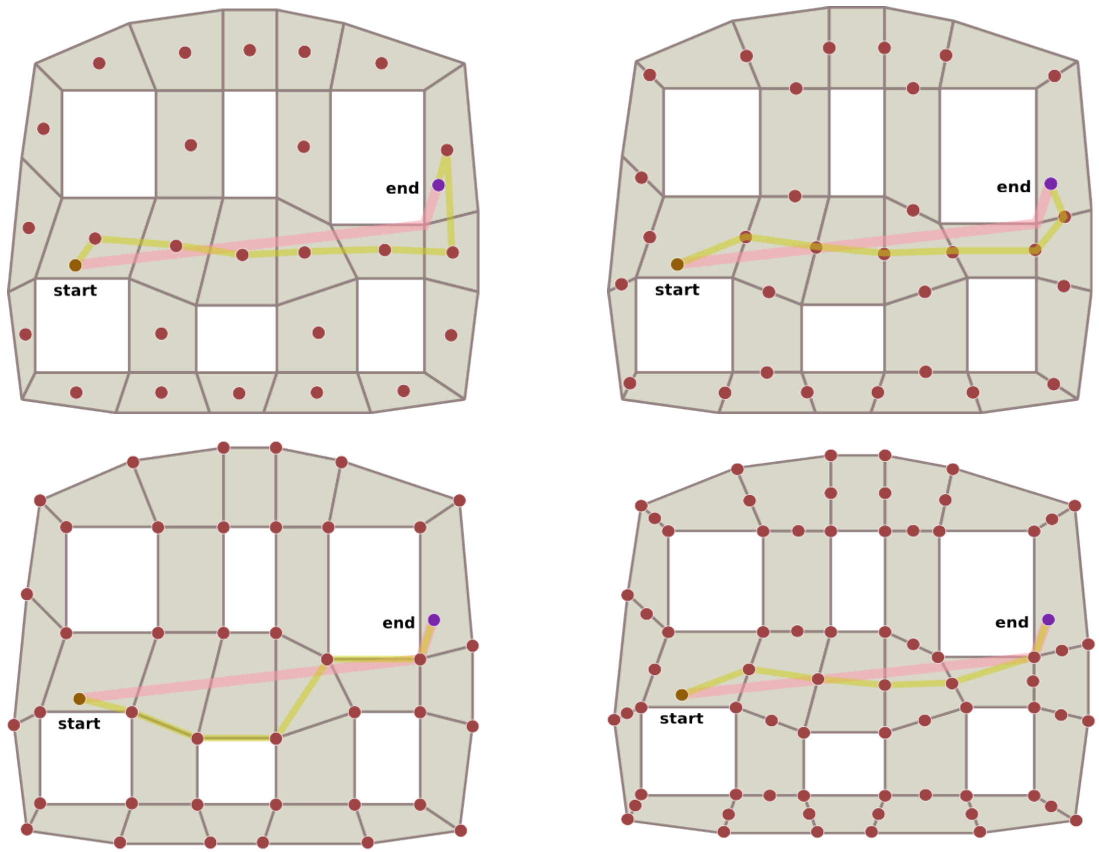
More flexibility for local motion

Figures from Amit Patel

Cell shapes and Path execution

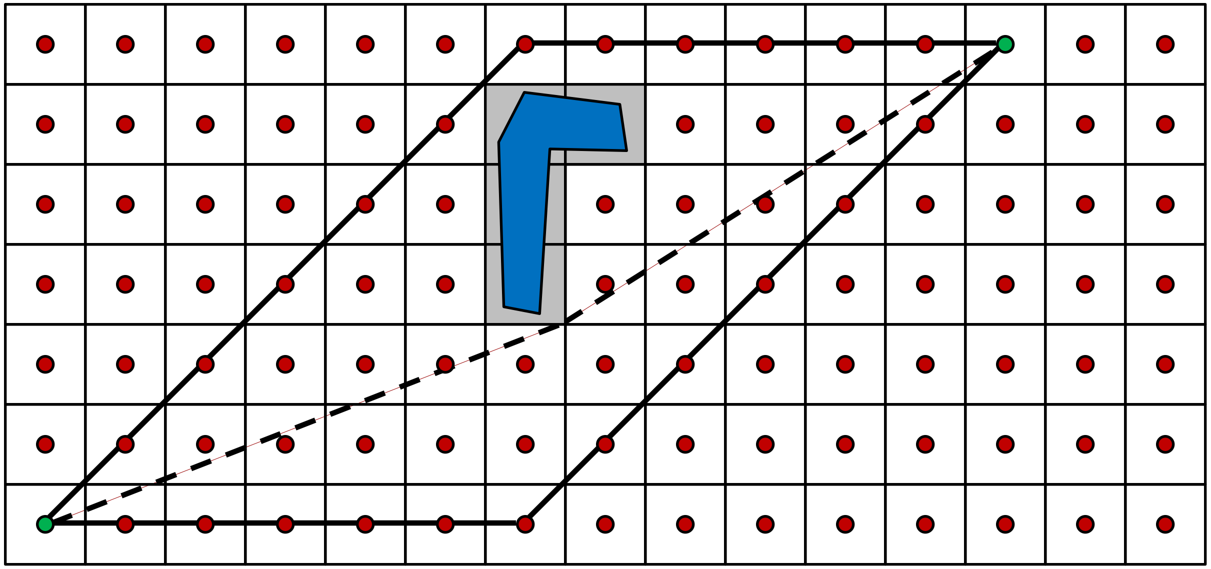
- *Meshes* can be used instead of uniform cells





Figures from Amit Patel

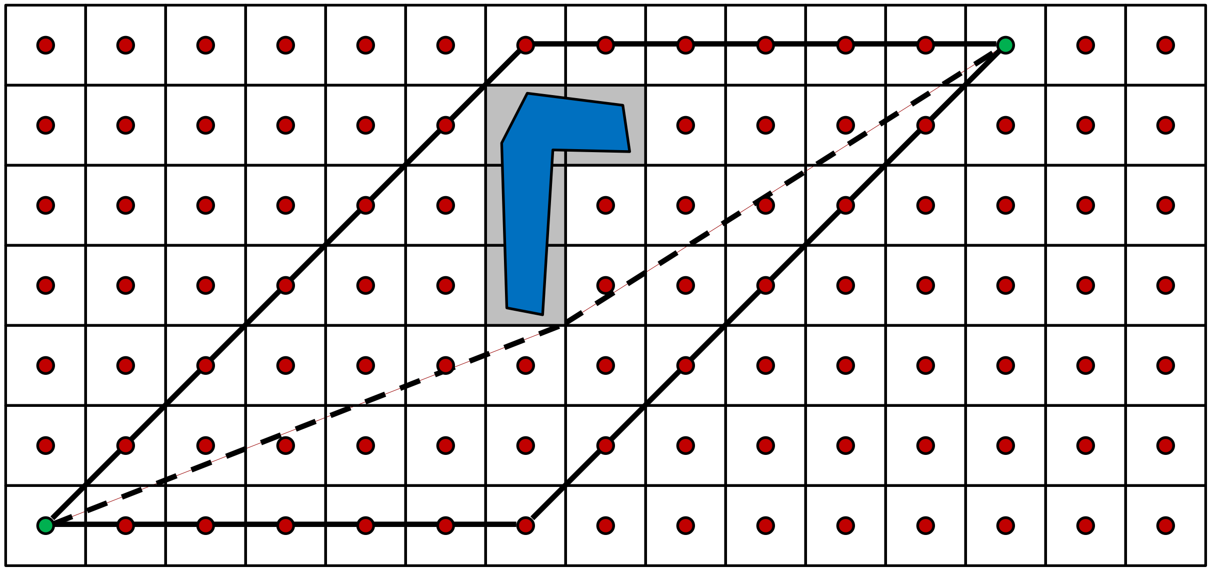
A* WITH TILES AND CENTERS



Shortest paths through cell centers

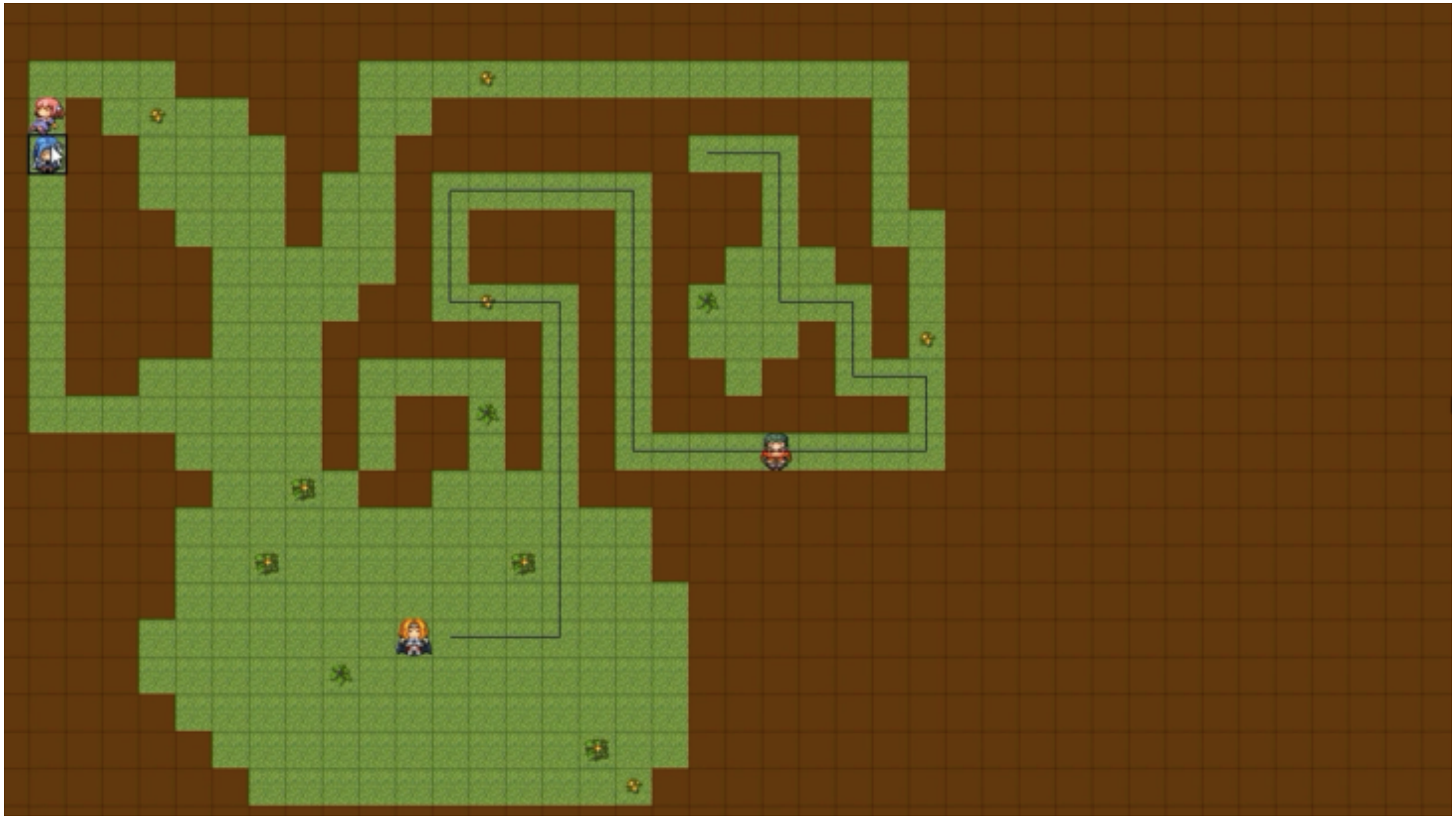
Shortest path

"Problem" of A* / Representation



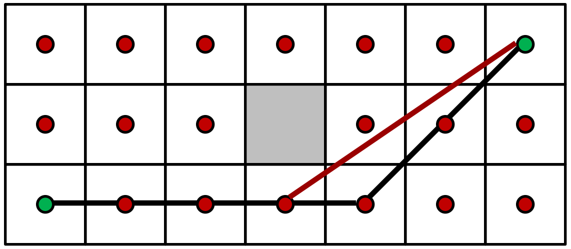
- A shortest path on the road map graph is *not* equivalent to a shortest path in the continuous environment
- A* propagates information on the graph and constrains paths to be formed by edges of the graph, *that only connect neighbor states*

A* WITH TILES AND CENTERS

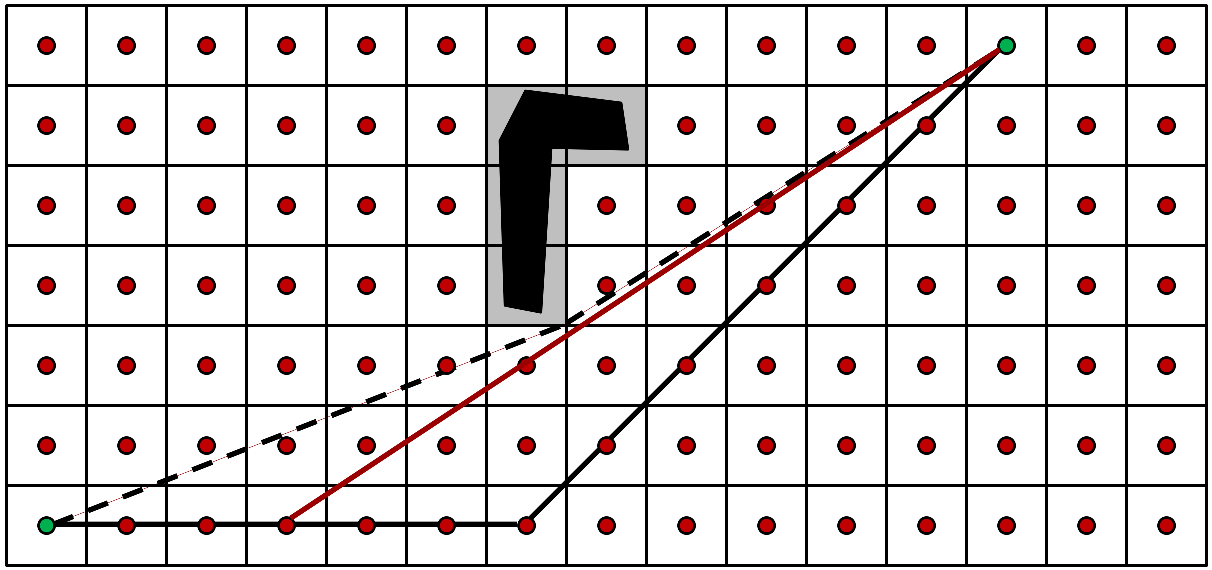


Solution 1: A* smoothing

- Allows connection to further states than neighbors on the graph
- Key observation:
  - If $x_1, \ldots x_j, \ \ldots \ xk \ \ldots \ x_n$ is valid path
  - And x_k is visible from x_j
  - Then $x_1, \ldots, x_j, x_k, \ldots, x_n$ is a valid path



Smoothing works!

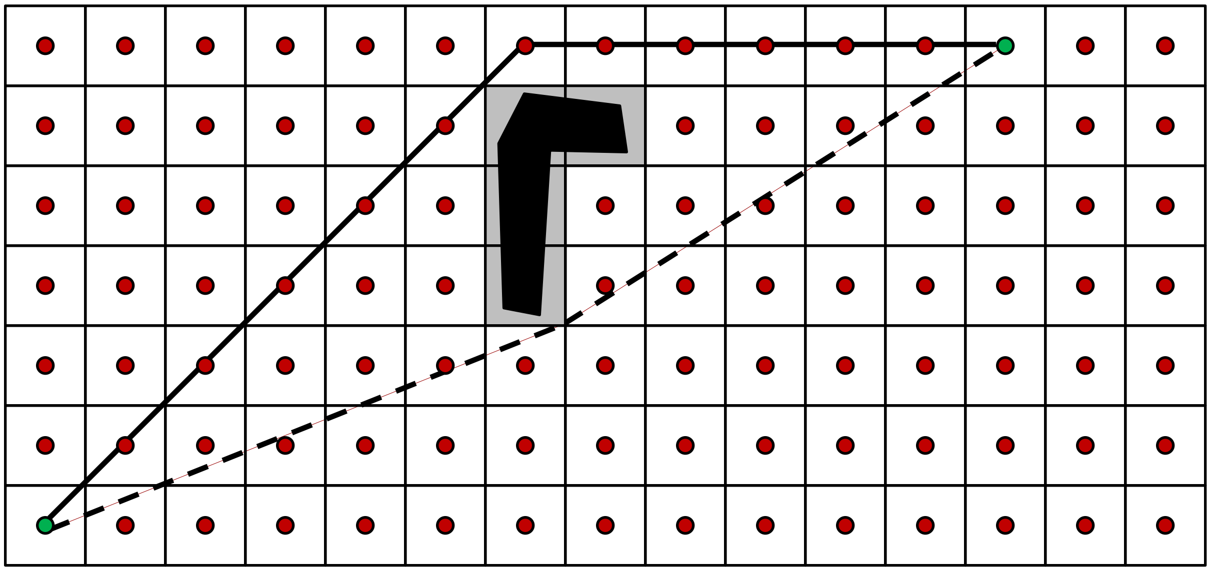


A shortest path through cell centers

Shortest path

What is left are only the navigation points that go around the corners of obstacles

Smoothing doesn't work!



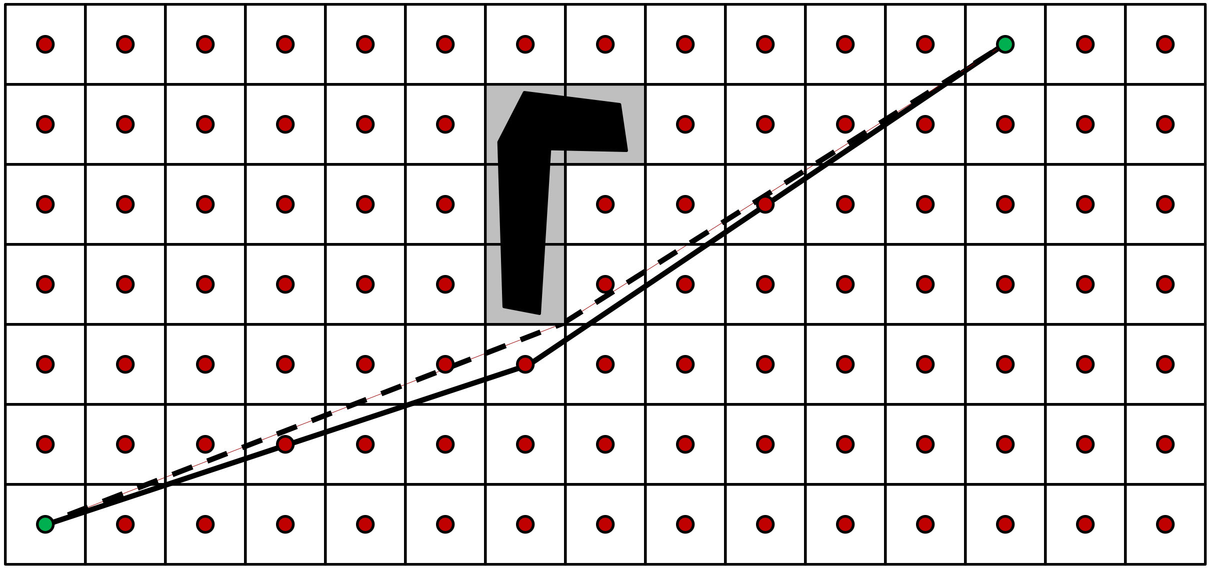
A shortest path through cell centers

Shortest path

Solution 2: Theta*

- Allow parents that are *non-neighbors* in the graph to be used during search
- Standard A*
  - Cost-to-come: $g(y) = g(x) + c(x, y)$
  - Insert y in frontier with cost estimate
  $$f(y) = g(x) + c(x, y) + h(y)$$
- Theta*
  - If parent(x) is visible from y, insert y with cost estimate $f(y) = g(\text{parent}(x)) + c(\text{parent}(x), y) + h(y)$
  - parent$(\boldsymbol{x})$ becomes the parent of $\boldsymbol{y}$, allowing the two-step stretching to iterate, if possible

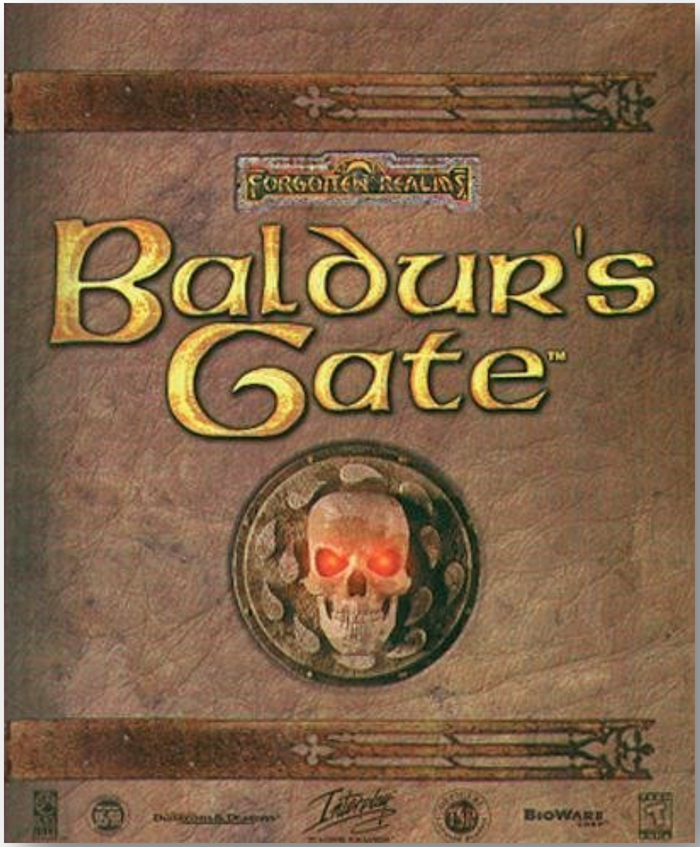
Theta* works!

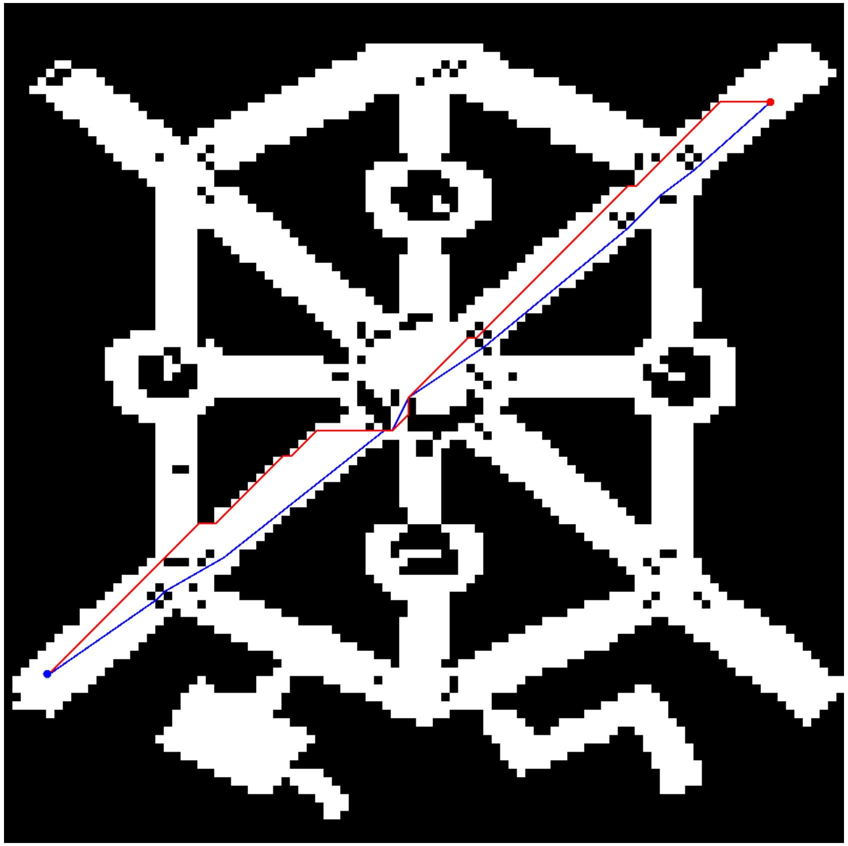


Theta* path, likely ☺

Shortest path

Theta* works!

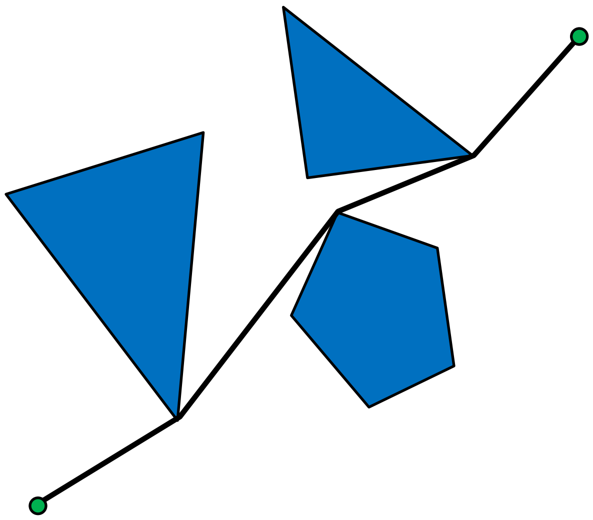




[Nash, AIGameDev 2010]

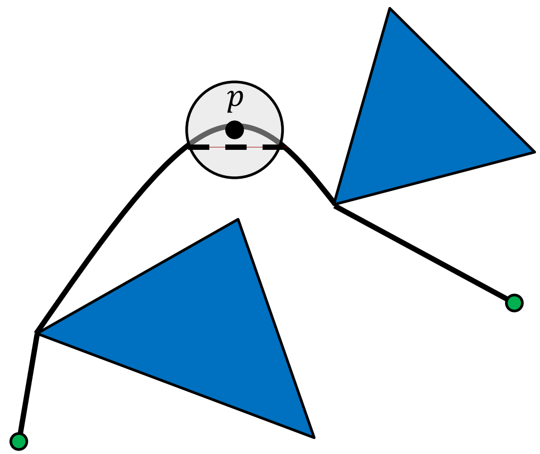
The optimal path

- Polygonal path: sequence of connected straight lines
- Inner vertex of polygonal path: vertex that is not beginning or end
- Theorem: Assuming *polygonal obstacles*, a shortest path is a *polygonal path* whose inner vertices are vertices of obstacles



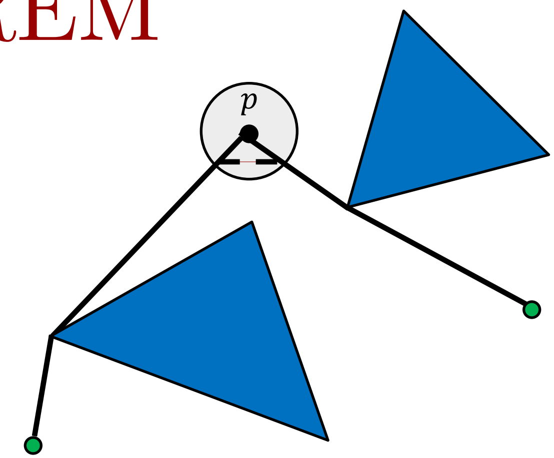
Proof of theorem

- Suppose for contradiction that shortest path is not polygonal
- Obstacles are polygonal $\Rightarrow \exists$ point p in interior of free space such that "spath through p is curved"
- $\exists$ disc of free space around p
- Path through disc can be shortened by connecting points of entry and exit → It's polygonal!

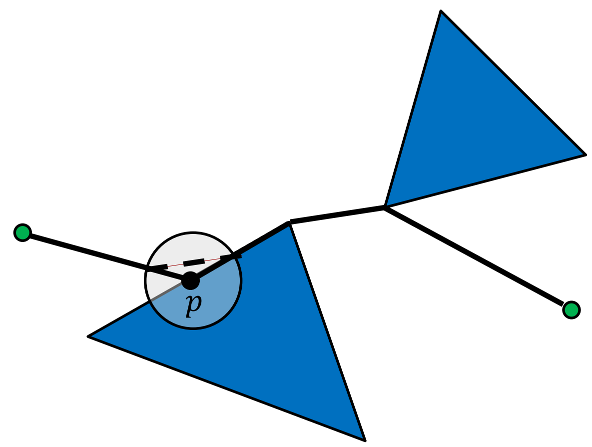


Proof of theorem

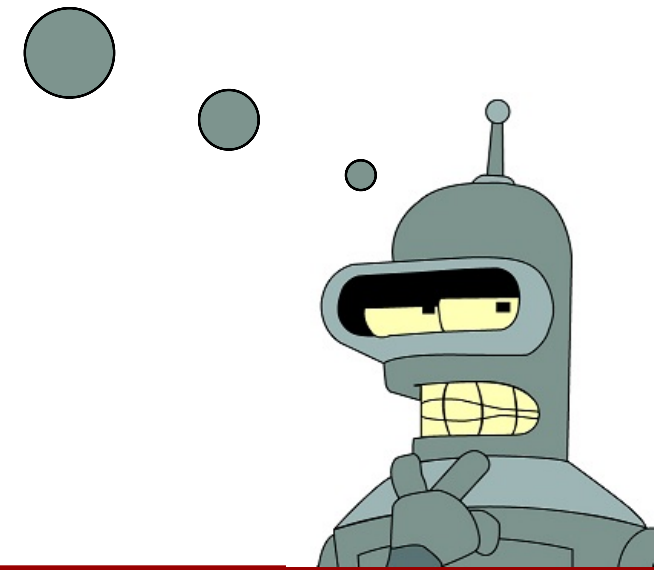
- Path is polygonal!
- Vertex cannot lie in interior of free space, otherwise we can do the same trick



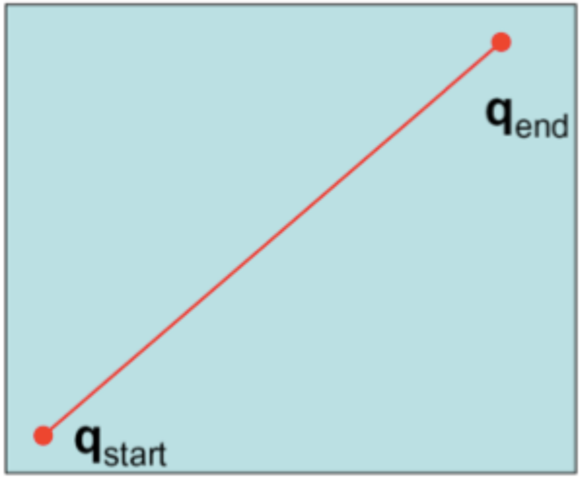
- Vertex cannot lie on a the interior of an edge, otherwise we can do the same trick ■

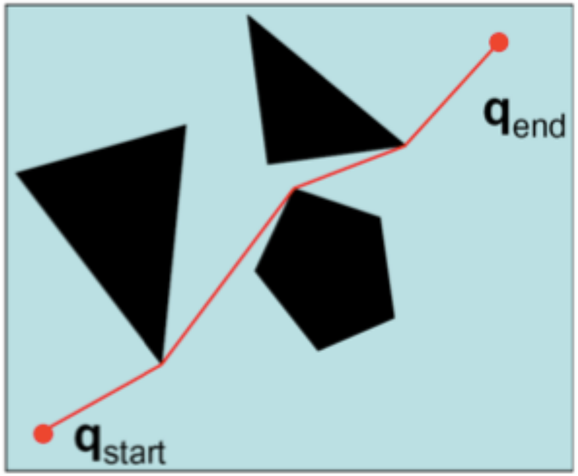


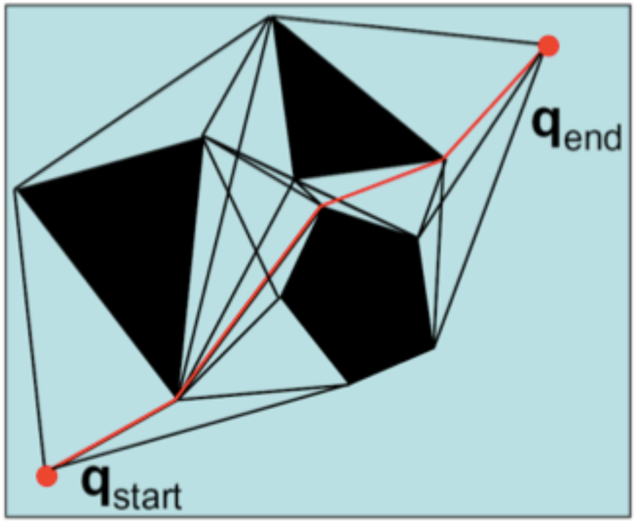
How would we define a graph on which A* would be optimal?



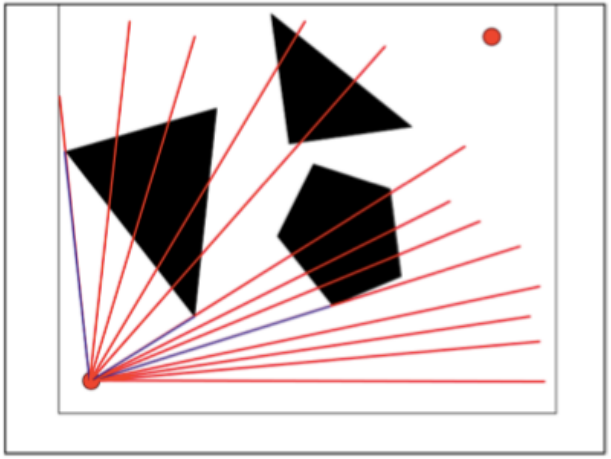
Visibility Graphs

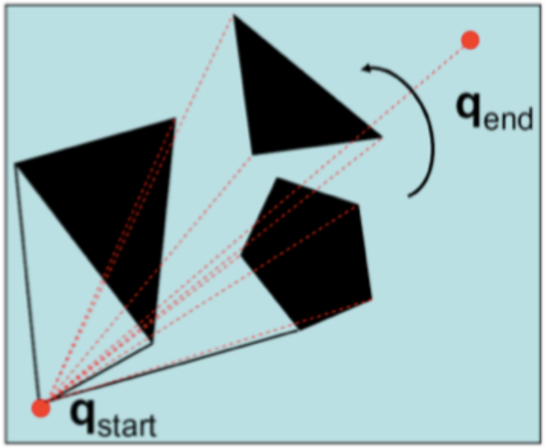






Sweeping,
O(n^2) complexity



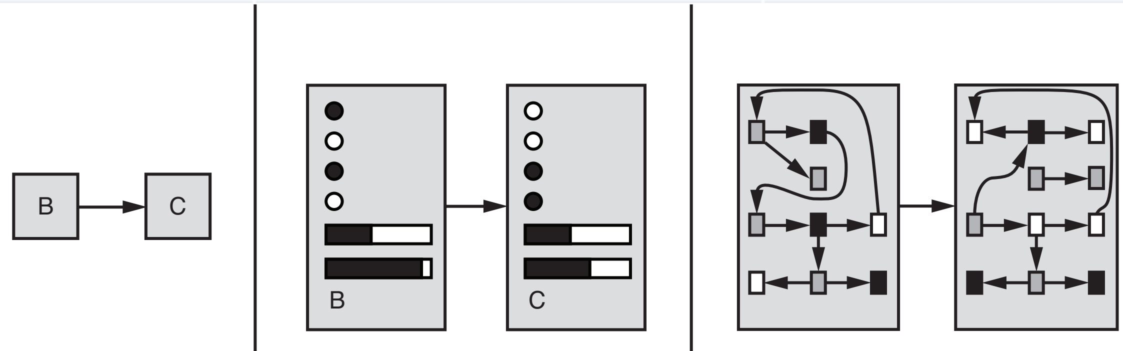


Planning, More Generally

- AI (also) studies rational action
- Devising a plan of action to achieve one's goal is a critical part of AI
- In fact, planning is glorified search
- We will consider a *structured representation of states*

State Representations

Type	State representation	Focus
Atomic	States are indivisible; No internal structure	Search on atomic states;
Propositional (aka Factored)	States are made of state variables that take values (Propositional or Multi-valued or Continuous)	Search+inference in logical (prop logic) and probabilistic (bayes nets) representations
Relational	States describe the objects in the world and their inter-relations	Search+Inference in predicate logic (or relational prob. Models)



(a) Atomic (b) Factored (b) Structured

State Representations

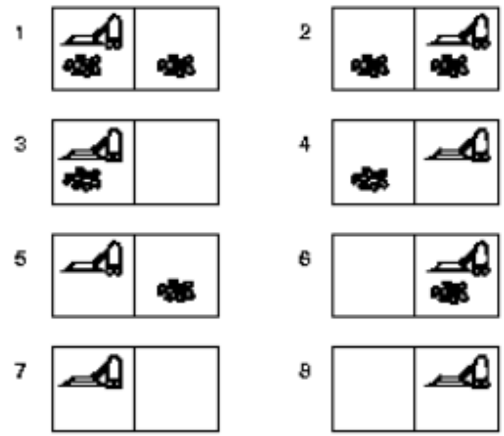
Atomic:

S1, S2…. S8

state is seen as an indivisible snapshot

All Actions are SXS matrices..

If you add a second roomba the state space *doubles*



Propositional/Factored:

States made up of 3 state variables

Dirt-in-left-room T/F

Dirt-in-right-room T/F

Roomba-in-room L/R

Each state is an assignment of Values to state variables

2^3 Different states

Actions can just mention the variables they affect

Note that the representation is compact (logarithmic in the size of the state space)

If you add a second roomba, the Representation increases by just one More state variable.

If you want to consider "noisiness" of rooms, we need *two* variables, one for Each room

Relational:

World made of objects: Roomba; L-room, R-room, dirt

Relations: In (<robot>, <room>); dirty(<room>)

If you add a second roomba, or more rooms, only the objects increase.

If you want to consider noisiness, you just need to add one other relation

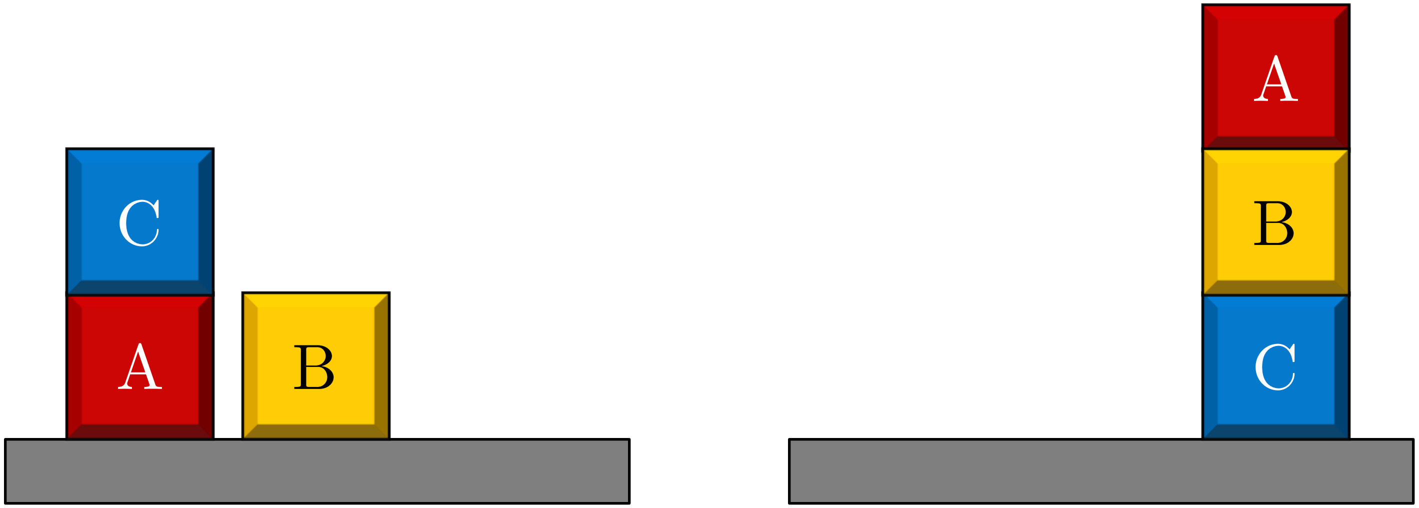
Propositional STRIPS planning

- STRIPS = Stanford Research Institute Problem Solver (1971)
- PDDL = Planning Domain Definition Language
- State is a conjunction of conditions, e.g., $\mathrm{at}(\mathrm{Truck}_1,\mathrm{Shadyside}) \wedge \mathrm{at}(\mathrm{Truck}_2,\mathrm{Oakland})$
- States are transformed via operators that have the form Preconditions $\Rightarrow$ Postconditions (effects)

PROPOSITIONAL STRIPS PLANNING

- Pre is a conjunction of positive and negative conditions that must be satisfied to apply the operation
- Post is a conjunction of positive and negative conditions that become true when the operation is applied
- We are given the initial state
- We are also given the goals, a conjunction of positive and negative conditions
- We think of a state as a *set of positive conditions*, hence an operation has an "add list" (the positive postconditions) and a "delete list" (the negative postconditions)

Blocks world



Start

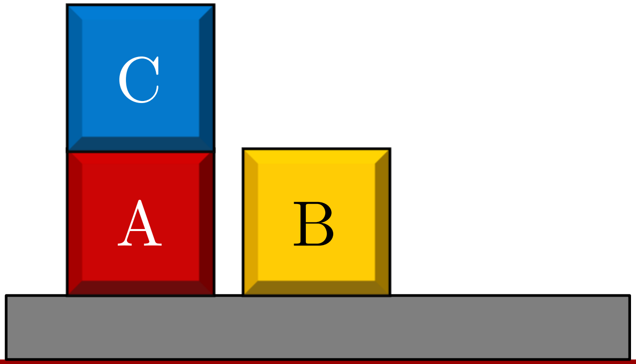
Goal

Blocks world

- Conditions: on(A,B), on(A,C), on(B,A), on(B,C), on(C,A), on(C,B), clear(A), clear(B), clear(C), on(A,Table), on(B,Table), on(C,Table)
- Operators for moving blocks
  - Move C from A to the table:
    clear(C) $\wedge$ on(C,A)
    $\Rightarrow$ on(C,Table) $\wedge$ clear(A) $\wedge$ $\neg$on(C,A)
  - Move A from the table to B
    clear(A) $\wedge$ on(A,Table) $\wedge$ clear(B)
    $\Rightarrow$ on(A,B) $\wedge$ $\neg$clear(B) and $\neg$on(A,Table)

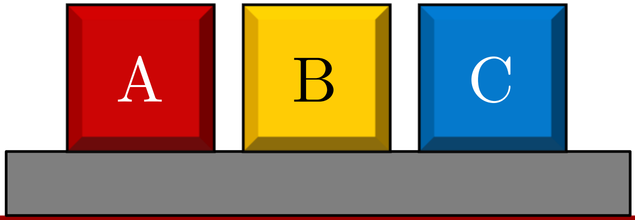
The plan

- State: on(C,A), on(A,Table), on(B,Table), clear(B), clear(C)
- Action:
  clear(C) $\wedge$ on(C,A)
  $\Rightarrow$ on(C,Table) $\wedge$ clear(A) $\wedge$ $\neg$on(C,A)



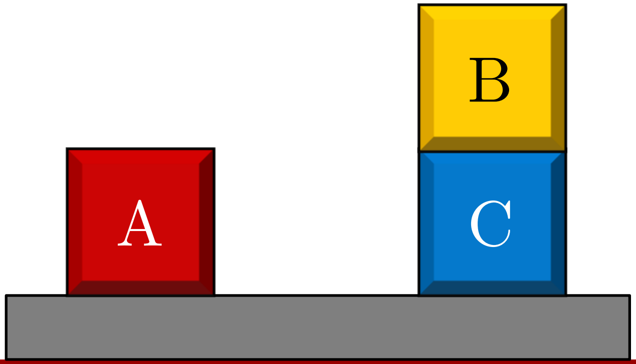
The plan

- State: on(A,Table), on(B,Table), clear(B), clear(C), on(C,Table), clear(A)
- Action:
  clear(C) $\wedge$ on(B,Table) $\wedge$ clear(B)
  $\Rightarrow$ on(B,C) $\wedge$ $\neg$clear(C) and $\neg$on(B,Table)



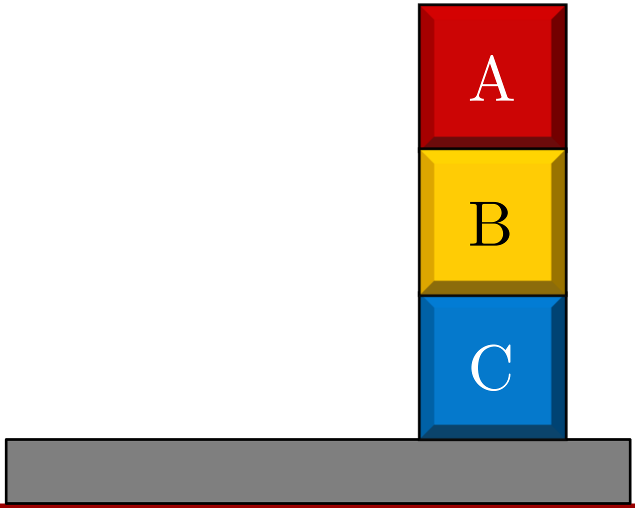
The plan

- State: on(A,Table), clear(B), on(C,Table), clear(A), on(B,C)
- Action:
  clear(B) $\wedge$ on(A,Table) $\wedge$ clear(A)
  $\Rightarrow$ on(A,B) $\wedge$ $\neg$clear(B) and $\neg$on(A,Table)



The plan

- State: on(C,Table), clear(A), on(B,C), on(A,B)
- Goals: on(A,B), on(B,C)



Complexity of planning

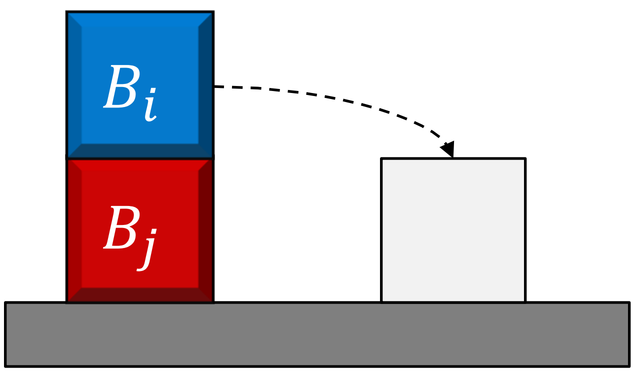
- Plansat is the problem of determining whether a given planning problem is *satisfiable*
- In general Plansat is **PSPACE**-complete
- We will look at some special cases (that are solved in Polynomial time)

COMPLEXITY OF PLANNING

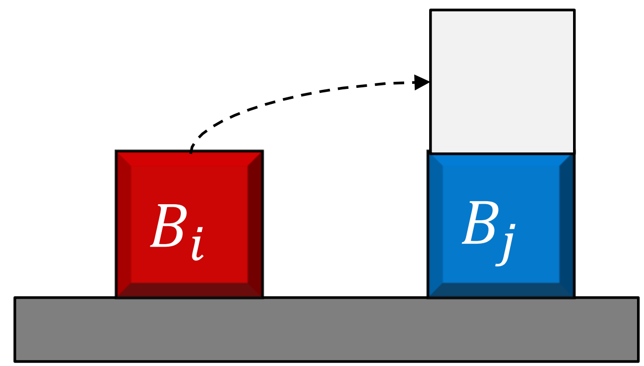
- Theorem 1: Assume that actions have only positive preconditions and a single postcondition. Then PLANSAT is in **P**
- Theorem 2: Blocks world problems can be encoded as above
- Silly corollary: Blocks world problems can be solved in polynomial time (Duh)

PROOF OF THEOREM 2*

- We will convert blocks world operators to operators that have only positive preconditions and a single postcondition
- Let the blocks be $B_1, \ldots, B_n$
- Conditions: $\mathbf{off}(i,j)$ means B_i is not on top of B_j



$$\bigwedge_k \text{off}(k,i) \wedge \bigwedge_{k\neq i} \text{off}(k,j) \Rightarrow \text{off}(i,j)$$



$$\bigwedge_k \text{off}(k,i) \wedge \bigwedge_k \text{off}(i,k) \wedge \bigwedge_k \text{off}(k,j) \Rightarrow \neg\text{off}(i,j)$$

*Just for fun

Proof of Theorem 1*

- Lemma: It is sufficient to consider plans that first make conditions true, then make conditions false
- Proof:
  - Suppose that o_i and o_{i+1} are adjacent operators s.t. the postcondition p of o_i is negative and the postcondition q of o_{i+1} is positive
  - If $p = q$ then we can delete o_i because its effect is reversed
  - Otherwise, we can switch o_i and o_{i+1} $\blacksquare$

* Just for fun

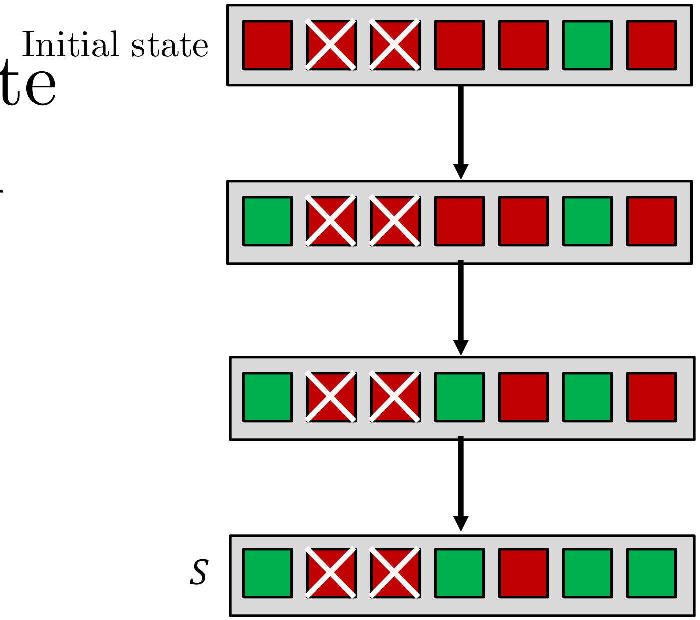
PROOF OF THEOREM 1*

- By the lemma, if there is a solution, there is an intermediate state S such that
  - S can be reached from the initial state using operations with positive postconditions
  - The positive goals are a subset of S
  - Negative goals can be achieved via operations with negative postconditions
- Search for an intermediate state S with these properties

* Just for fun

Proof of Theorem 1*

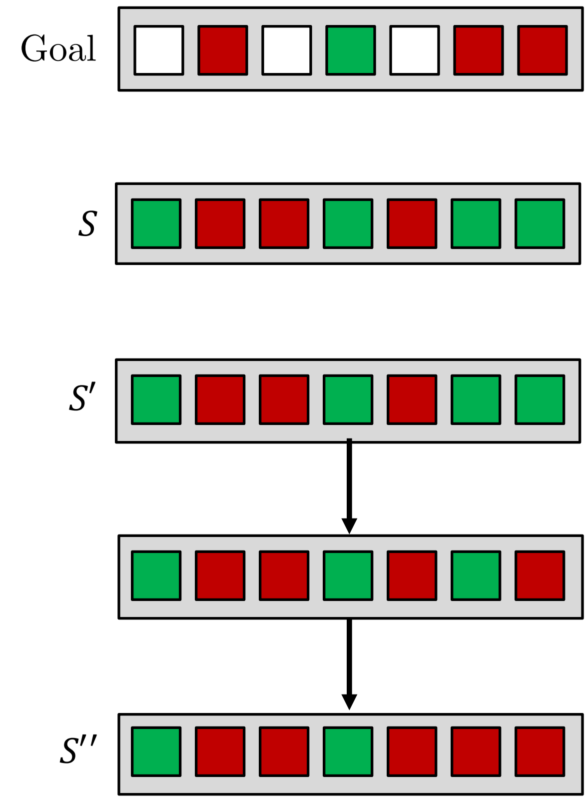
- Implement procedure $\mathsf{TurnOn}(X)$: given set of conditions X, find maximal state S such that $S \cap X = \emptyset$ that can be reached from initial state using operators with positive postconditions
  - Preconditions are positive, so:
  - Simply apply all such operators until it makes no difference



* Just for fun

Proof of Theorem 1*

- Denote S'' the state resulting from removing negative goals from S
- Implement procedure TurnOff(S): find the maximal S' such that S'' is reachable from S' using operators with negative postconditions in S
  - Simply search backwards from S'' and reverse operators with (i) negative postconditions in S (ii) preconditions satisfied



* Just for fun

Proof of Theorem 1*

- In the first iteration, if positive goals are not satisfied by S, there is no way to achieve them
- If $S \setminus S' \neq \emptyset$, it is impossible to remove these conditions; must be added to X
- X grows monotonically $\Rightarrow$ polynomial time ∎

$X = \emptyset$
loop
    $S = \mathrm{TurnOn}(X)$
    if S does not contain positive
        goals then return reject
    $S' = \mathrm{TurnOff}(S)$
    if $S = S'$ then return accept
    $X = X \cup (S \setminus S')$
    if X intersects with initial
        state then return reject

* Just for fun

Summary

- Terminology:
  - Road map graph
  - Cell decomposition
  - Resolution completeness
  - Visibility graph
  - Theta*
  - STRIPS
- Useful ideas:
  - Natural restrictions can drastically decrease the complexity of planning