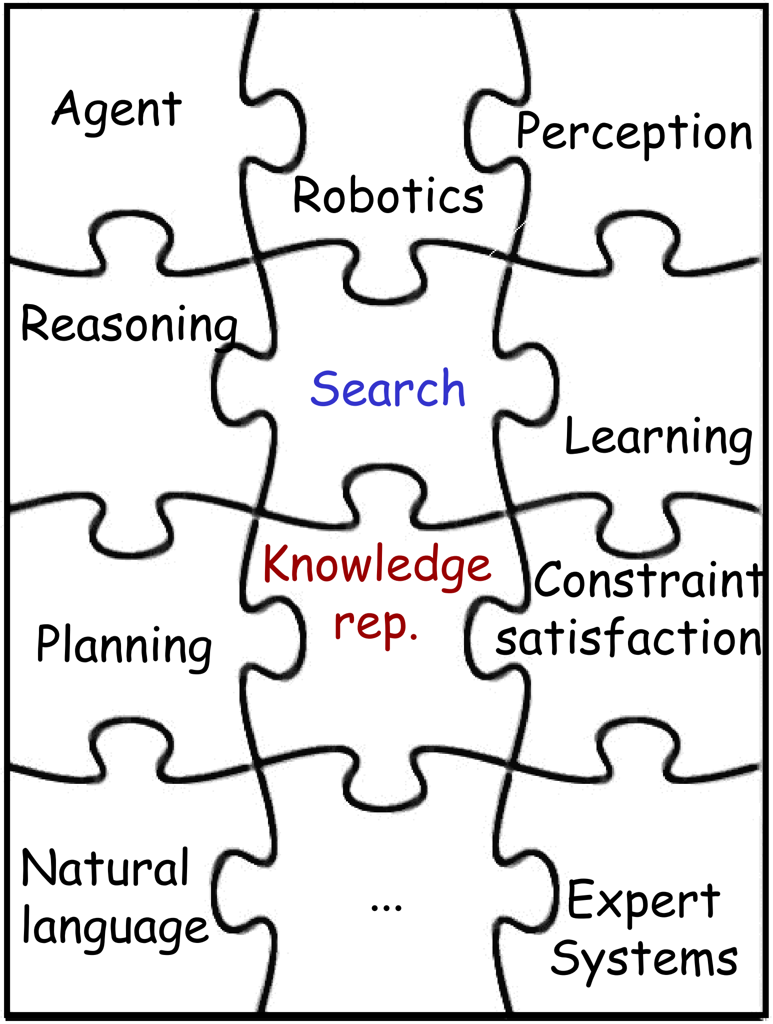
Search Problems

(Where reasoning consists of exploring alternatives)

R&N: Chap. 3, Sect. 3.1-2 + 3.6



- Declarative knowledge creates alternatives:
  - Which pieces of knowledge to use?
  - How to use them?
- Search is a about exploring alternatives. It is a major approach to exploit knowledge

Problem-solving agents

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
   persistent: seq, an action sequence, initially empty
               state, some description of the current world state
               goal, a goal, initially null
               problem, a problem formulation
   state ← UPDATE-STATE(state, percept)
   if seq is empty then
      goal ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, goal)
      seq ← SEARCH(problem)
      if seq = failure then return a null action
   action ← FIRST(seq)
   seq ← REST(seq)
   return action
```

Example: 8-Puzzle

8	2	
3	4	7
5	1	6

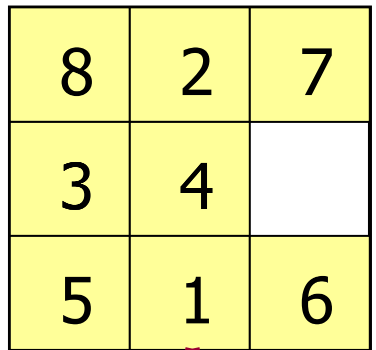
Initial state

1	2	3
4	5	6
7	8	

Goal state

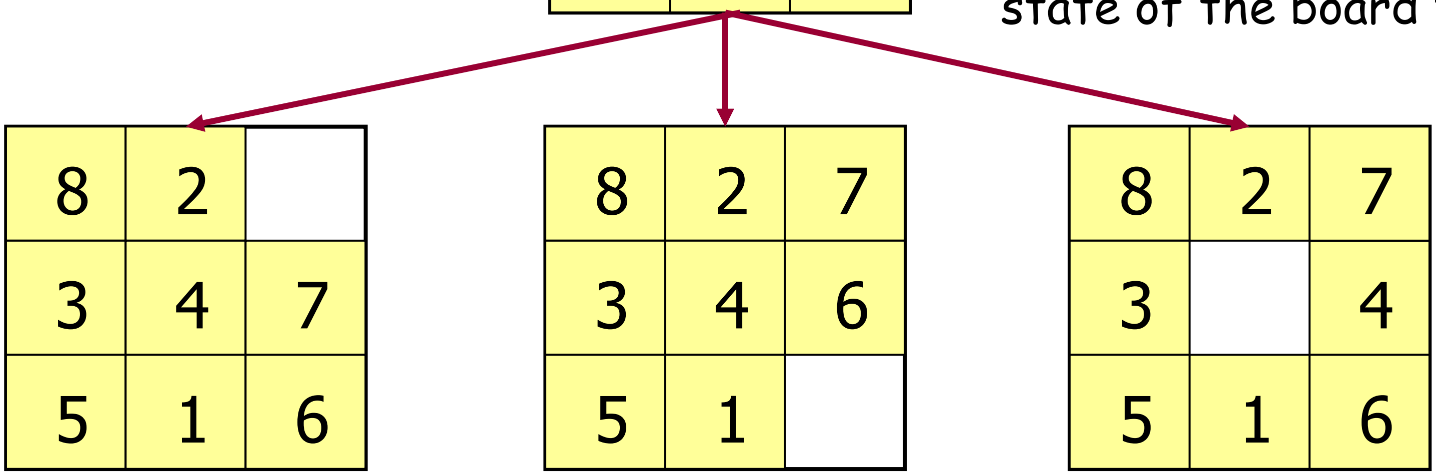
State: Any arrangement of 8 numbered tiles and an empty tile on a 3x3 board

8-Puzzle: Successor Function



SUCC(state) → subset of states

The successor function (RESULT function) is knowledge about the 8-puzzle game, but it does not tell us which outcome to use, nor to which state of the board to apply it.



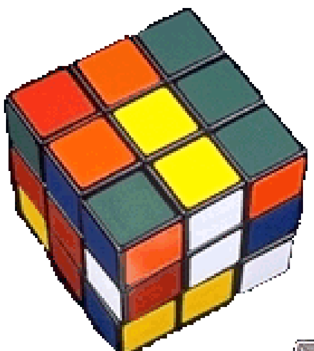
Search is about the exploration of alternatives

Across history, puzzles and games requiring the exploration of alternatives have been considered a challenge for human intelligence:

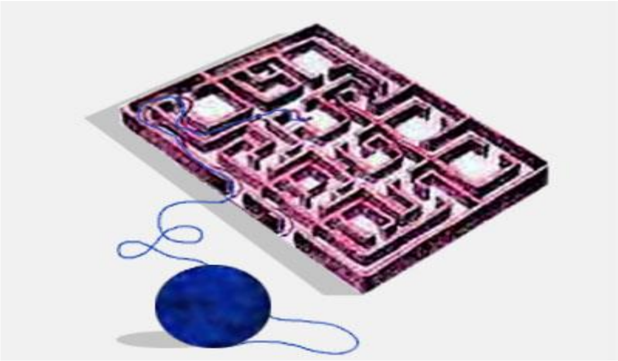
- Chess originated in Persia and India about 4000 years ago
- Checkers appear in 3600-year-old Egyptian paintings
- Go originated in China over 3000 years ago

So, it's not surprising that AI uses games to design and test algorithms


MASH SOMATIC MAPS
PIXIE WAYSIDE ESAU
ODELL ARTICLE LIT
WILLIAMSHAKESPEARE
XS SENIOR
AMFM IRC RIGHT ANY
SALMANRUSHDIE ARSE
ALUMNI BUYIN RC
ONEI MO PH
ERNESTHEMINGWAY
ABE ES MOA
SMBD ULTRA YONDER
MISS ALLENGINSBERG
UN ODEAR EVE CRAB
TOILER SO
BARBARAKINGSOLVER
ELENA MALARIA MOIRE
REACT ONANIST PLASM
ROMEO KINESIS HALT




PUZZLE PICKS
PUZZLE PICKS




Puzzle-Peg
Can YOU Solve It?
LUBBERS & BELL MFG. CO.
CLINTON, IOWA, U.S.A.
Puzzle-Peg


SOMETHING FISHY

(n^2-1)-puzzle

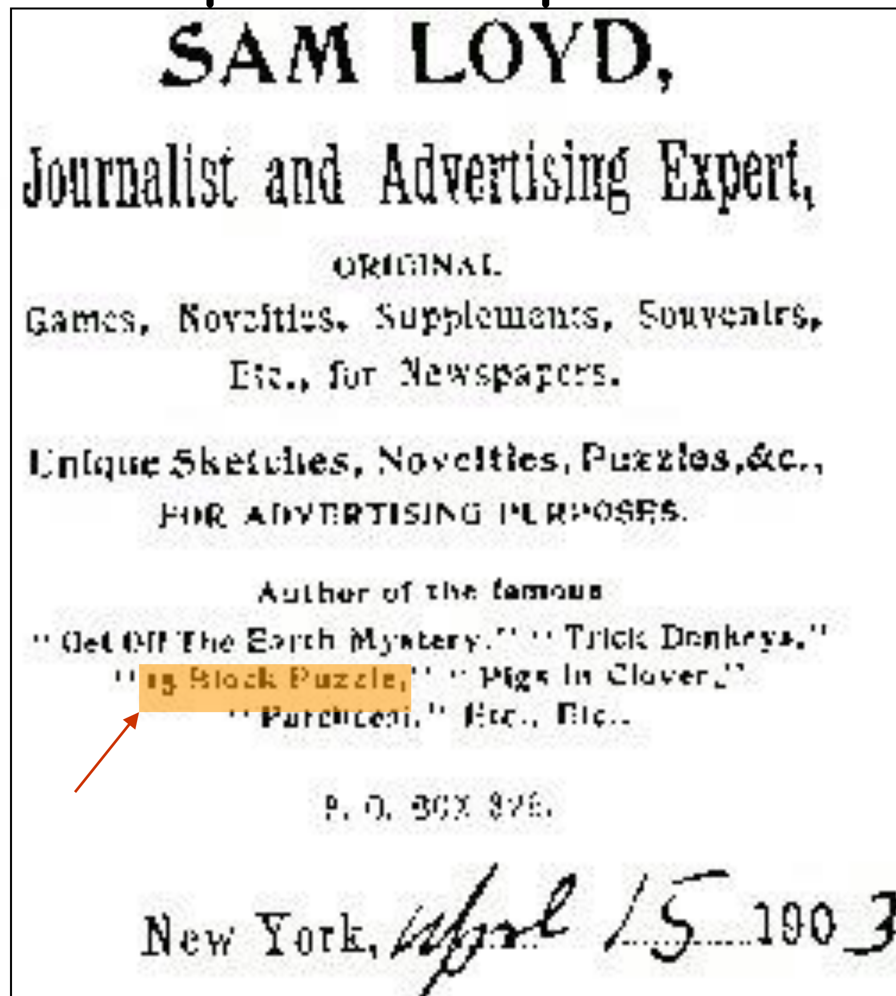
8	2	
3	4	7
5	1	6

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

. . . .

15-Puzzle

Introduced (?) in 1878 by Sam Loyd, who dubbed himself "America's greatest puzzle-expert"



SAM LOYD,

Journalist and Advertising Expert,

ORIGINAL

Games, Novelties, Supplements, Souvenirs, Etc., for Newspapers.

Unique Sketches, Novelties, Puzzles, &c., FOR ADVERTISING PURPOSES.

Author of the famous

"Get Off The Earth Mystery," "Trick Donkeys," "15 Block Puzzle," "Pigs in Clover," "Parcheesi," Etc., Etc.

P. O. BOX [illegible].

New York, April 15 1903

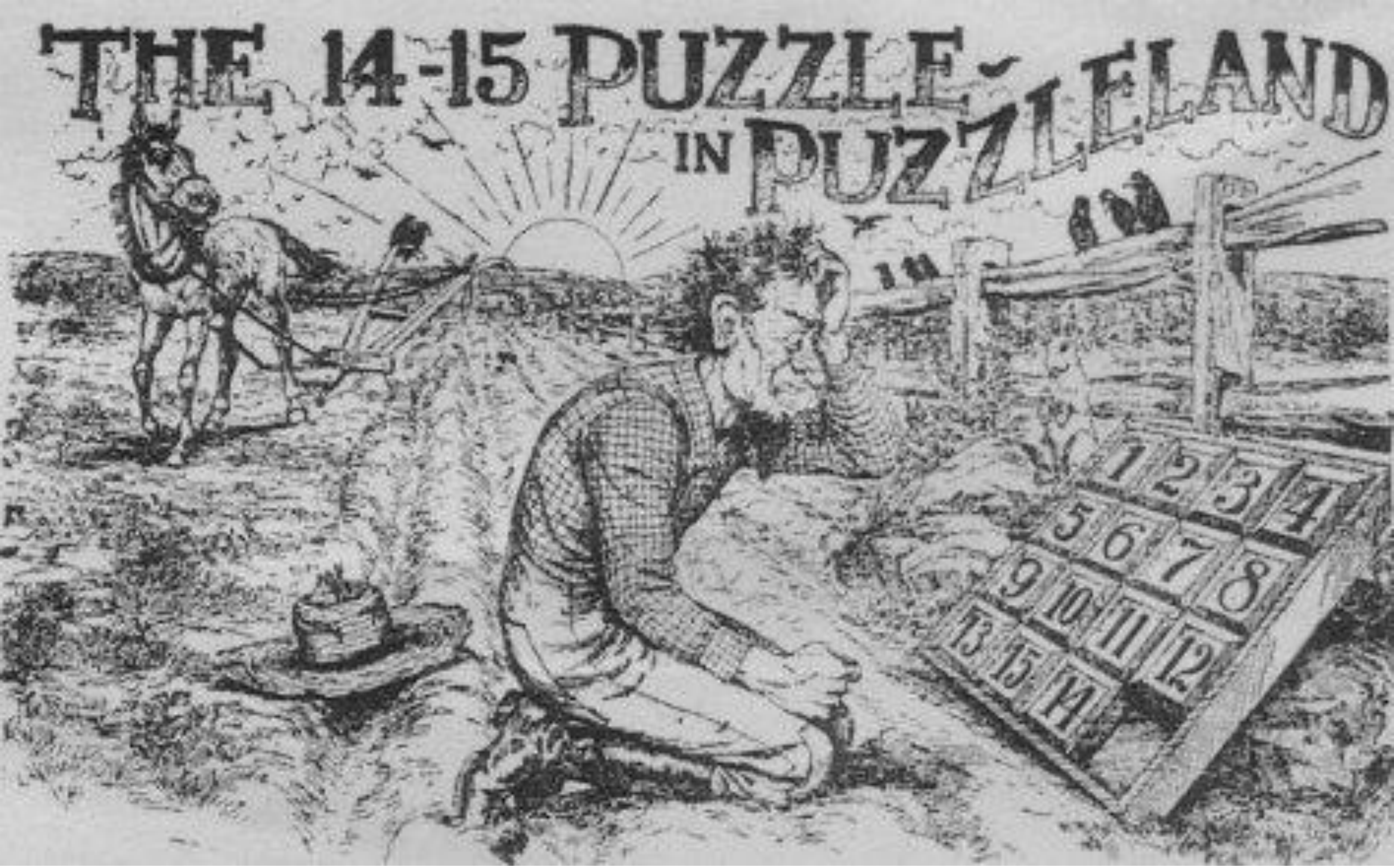
15-Puzzle

Sam Loyd offered $1,000 of his own money to the first person who would solve the following problem:

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

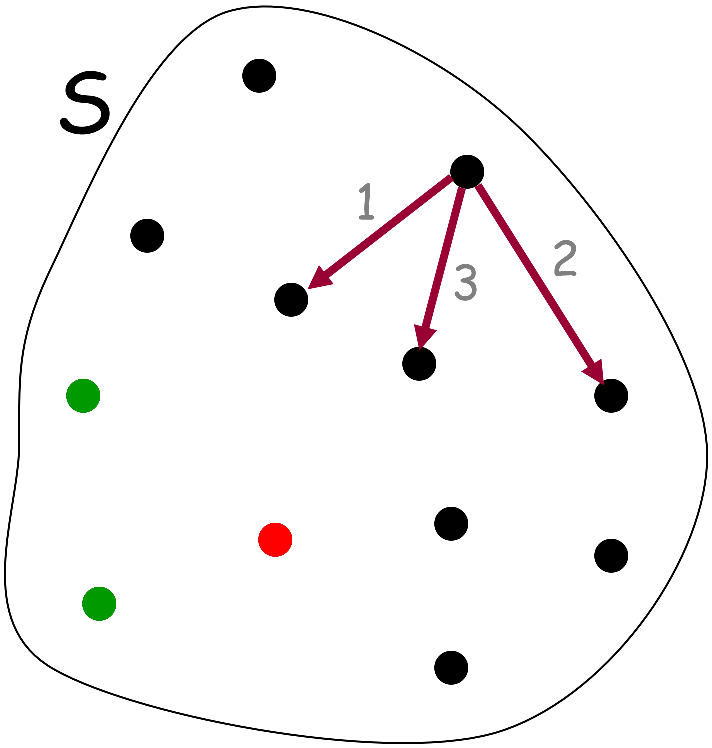
?

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



But no one ever won the prize !!

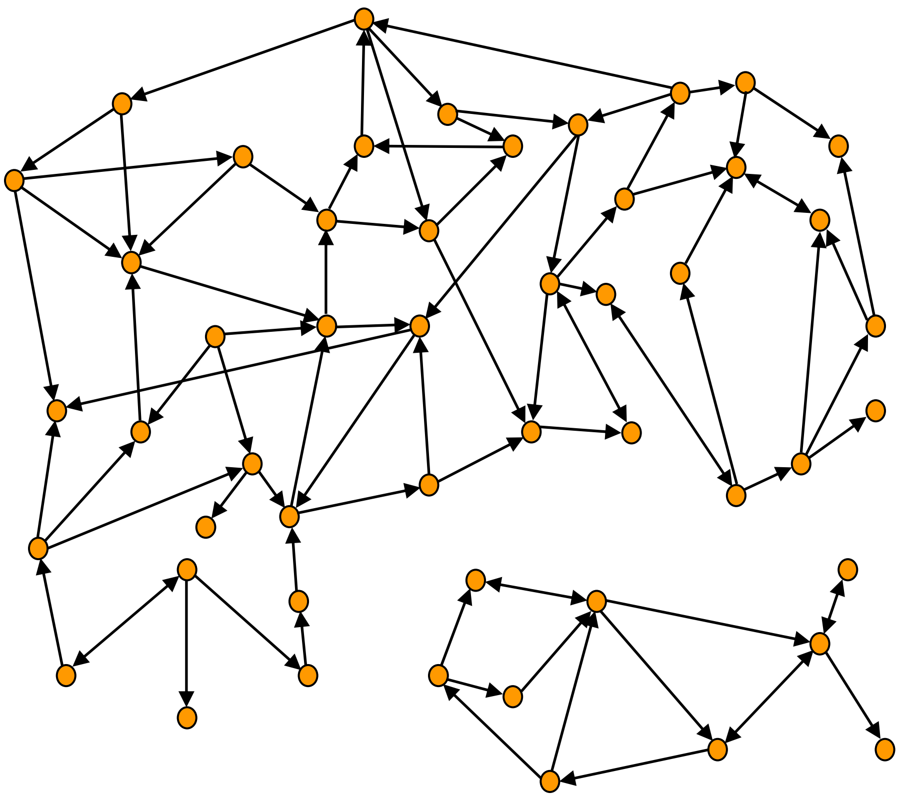
Stating a Problem as a Search Problem



- State space S
- Successor function:
  $x \in S \rightarrow$ SUCCESSORS$(x) \in 2^S$
- Initial state s_0
- Goal test:
  $x \in S \rightarrow$ GOAL?(x) = T or F
- Arc cost

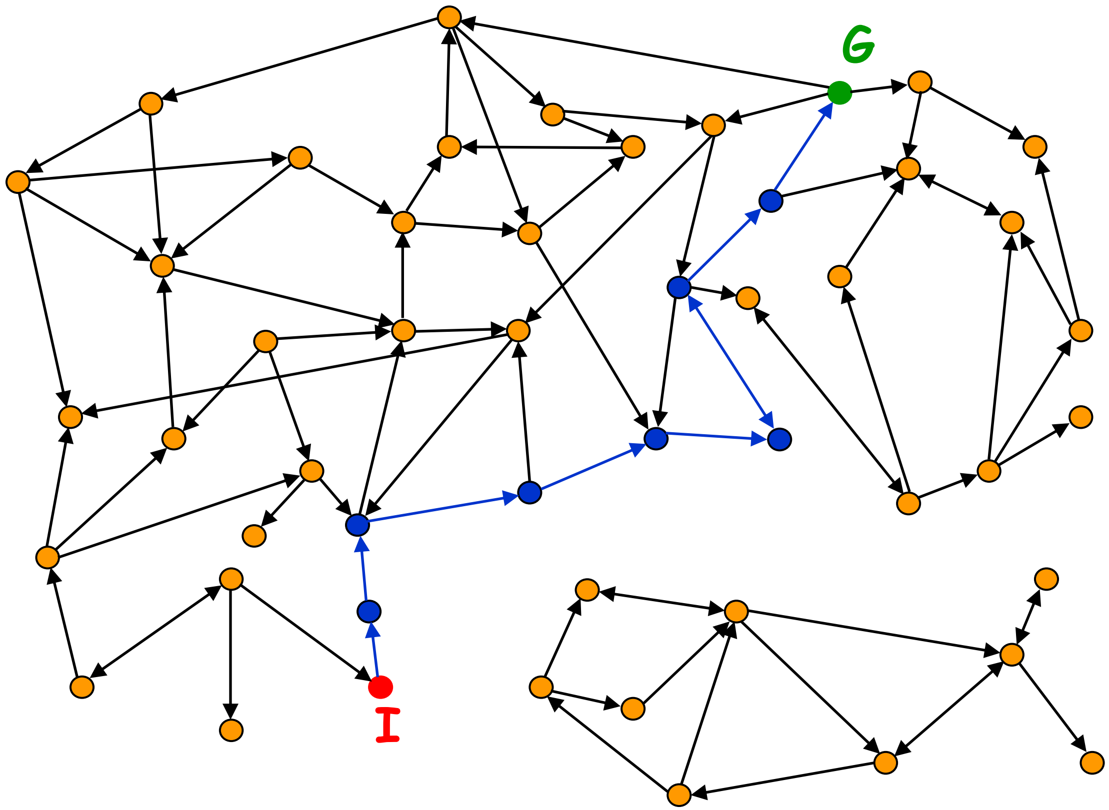
State Graph

- Each state is represented by a distinct node
- An arc (or edge) connects a node s to a node s' if s' ∈ SUCCESSORS(s)
- The state graph may contain more than one connected component



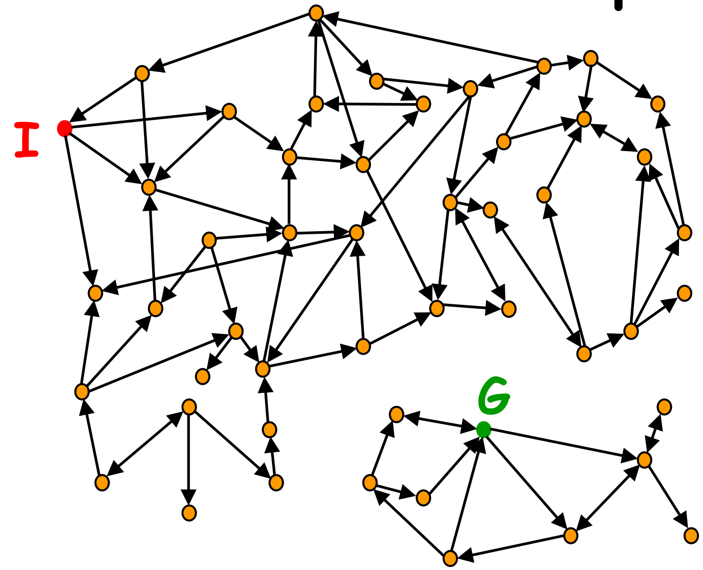
Solution to the Search Problem

- A solution is a path connecting the initial node to a goal node (any one)



Solution to the Search Problem

- A solution is a path connecting the initial node to a goal node (any one)
- The cost of a path is the sum of the arc costs along this path
- An optimal solution is a solution path of minimum cost
- There might be no solution !



How big is the state space of the (n^2-1)-puzzle?

- 8-puzzle → ?? states

How big is the state space of the (n^2-1)-puzzle?

- 8-puzzle → 9! = 362,880 states
- 15-puzzle → 16! ~ 2.09×10^{13} states
- 24-puzzle → 25! ~ 10^{25} states

But <u>only half</u> of these states are reachable from any given state
(but you may not know that in advance)

Permutation Inversions

- Wlg, let the goal be:

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

- A tile j appears after a tile i if either j appears on the same row as i to the right of i, or on another row below the row of i.
- For every i = 1, 2, ..., 15, let n_i be the number of tiles j < i that appear after tile i (permutation inversions)
- $N = n_2 + n_3 + \ldots + n_{15}$ + row number of empty tile

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

$n_2 = 0$ $n_3 = 0$ $n_4 = 0$
$n_5 = 0$ $n_6 = 0$ $n_7 = 1$
$n_8 = 1$ $n_9 = 1$ $n_{10} = 4$
$n_{11} = 0$ $n_{12} = 0$ $n_{13} = 0$
$n_{14} = 0$ $n_{15} = 0$

→ N = 7 + 4

- Proposition: (N mod 2) is invariant under any legal move of the empty tile
- Proof:
  - Any horizontal move of the empty tile leaves N unchanged
  - A vertical move of the empty tile changes N by an even increment ($\pm 1 \pm 1 \pm 1 \pm 1$)

s =

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

s' =

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

$N(s') = N(s) + 3 + 1$

- Proposition: (N mod 2) is invariant under any legal move of the empty tile
- → For a goal state g to be reachable from a state s, a necessary condition is that N(g) and N(s) have the same parity
- It can be shown that this is also a sufficient condition
- → The state graph consists of two connected components of equal size

15-Puzzle

Sam Loyd offered $1,000 of his own money to the first person who would solve the following problem:

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

?

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

N = 4

N = 5

So, the second state is not reachable from the first, and Sam Loyd took no risk with his money ...

What is the Actual State Space?

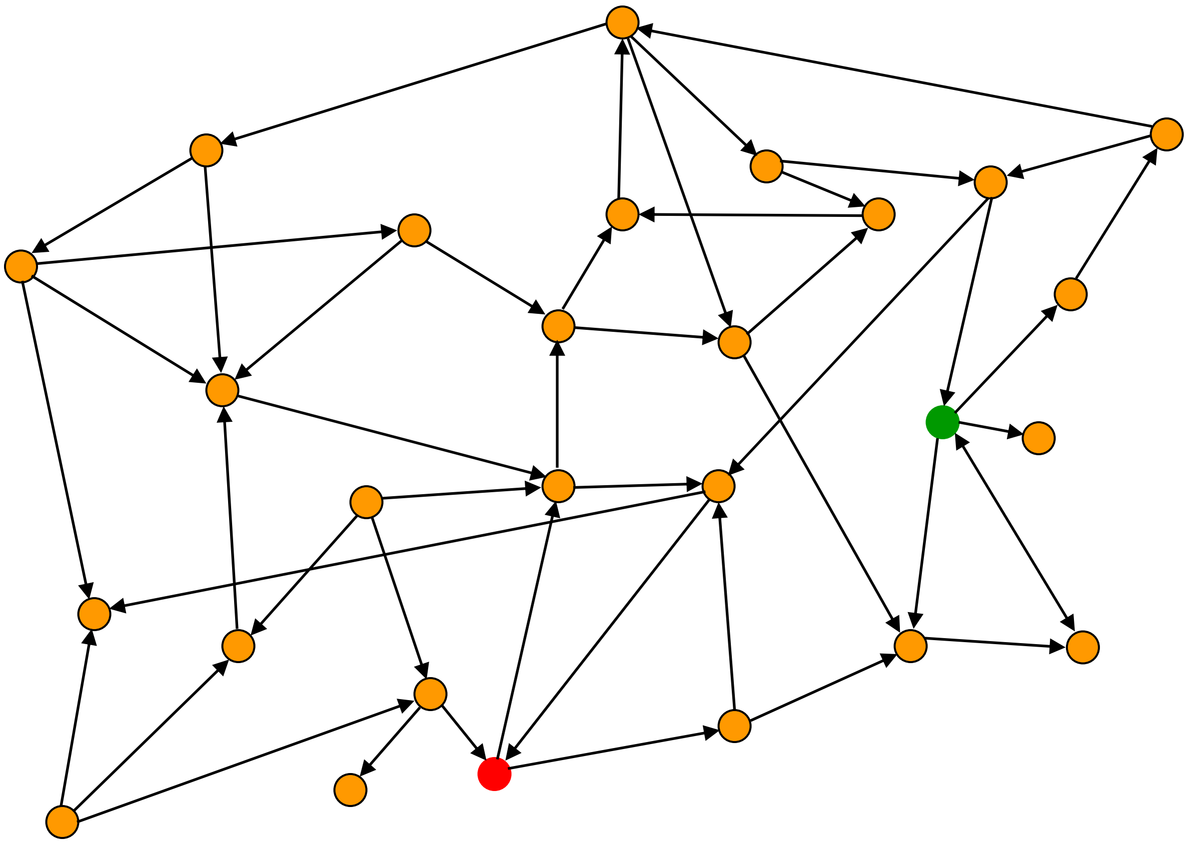
a) The set of all states?
[e.g., a set of 16! states for the 15-puzzle]

b) The set of all states reachable from a given initial state?
[e.g., a set of 16!/2 states for the 15-puzzle]

In general, the answer is a)
[because one does not know in advance which states are reachable]

But a fast test determining whether a state is reachable from another is very useful, as search techniques are often **inefficient** when a problem has no solution

Searching the State Space



- It is often not feasible (or too expensive) to build a complete representation of the state graph

8-, 15-, 24-Puzzles

8-puzzle → 362,880 states

0.036 sec

15-puzzle → 2.09 x 10^{13} states

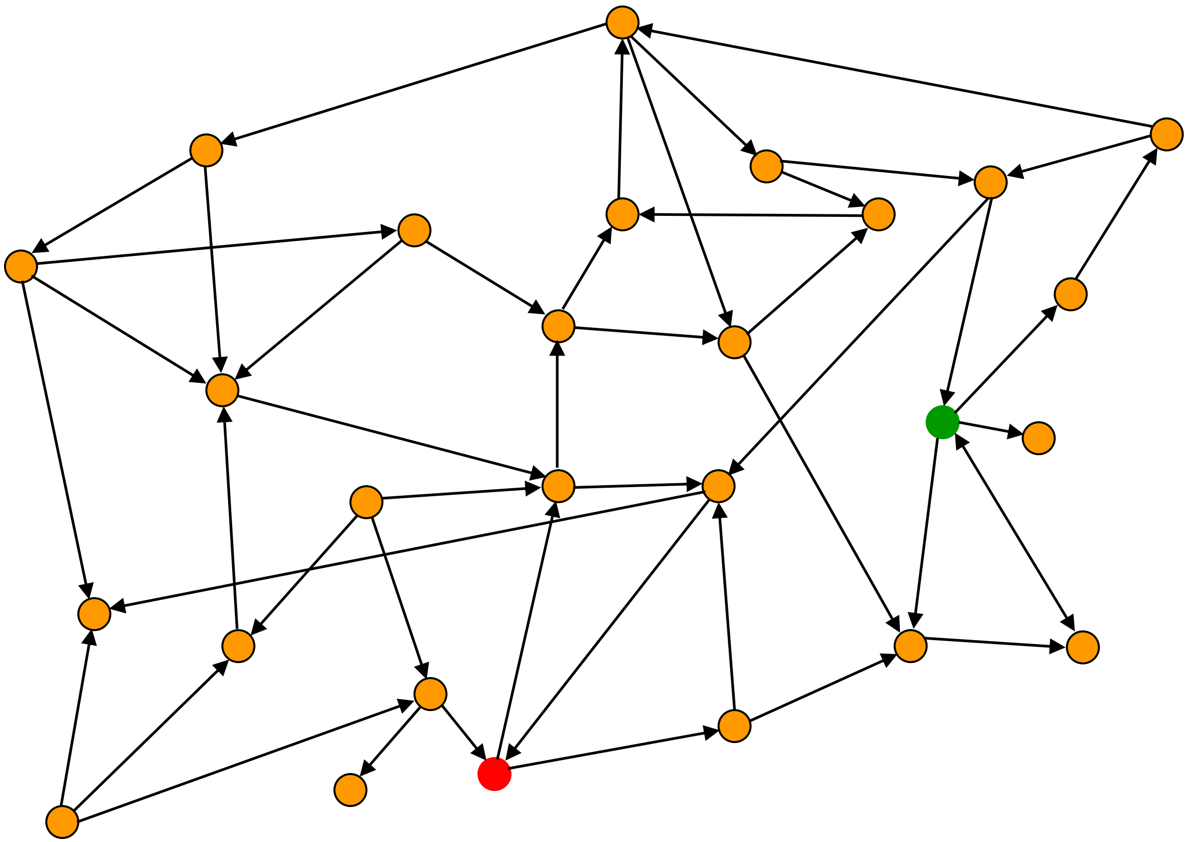
~ 55 hours

24-puzzle → 10^{25} states

> 10^9 years

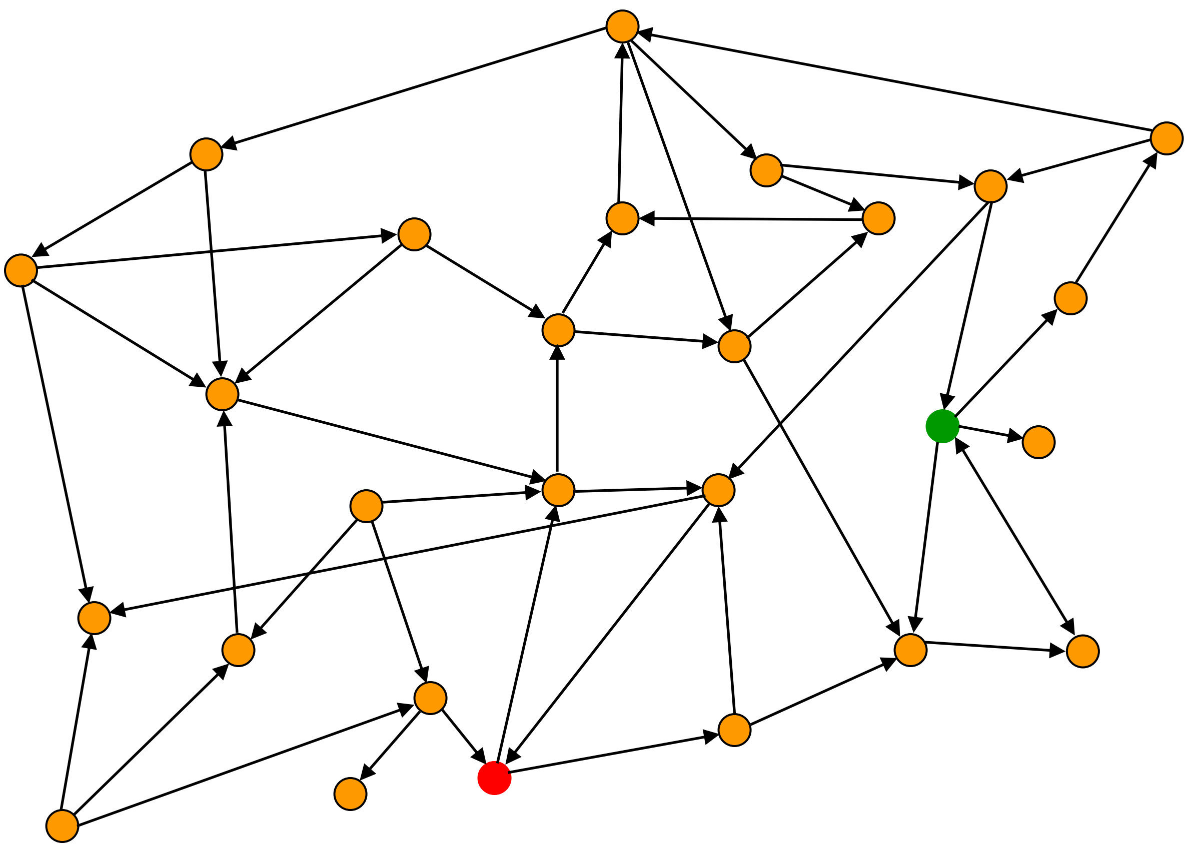
100 millions states/sec

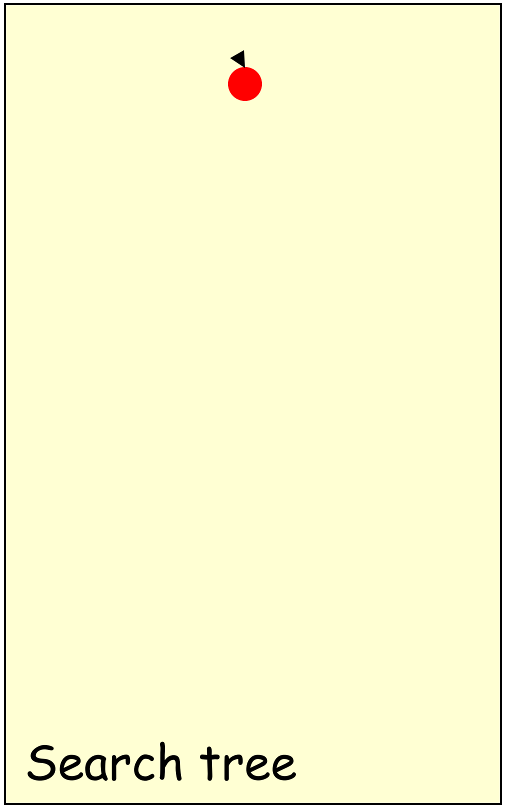
Searching the State Space



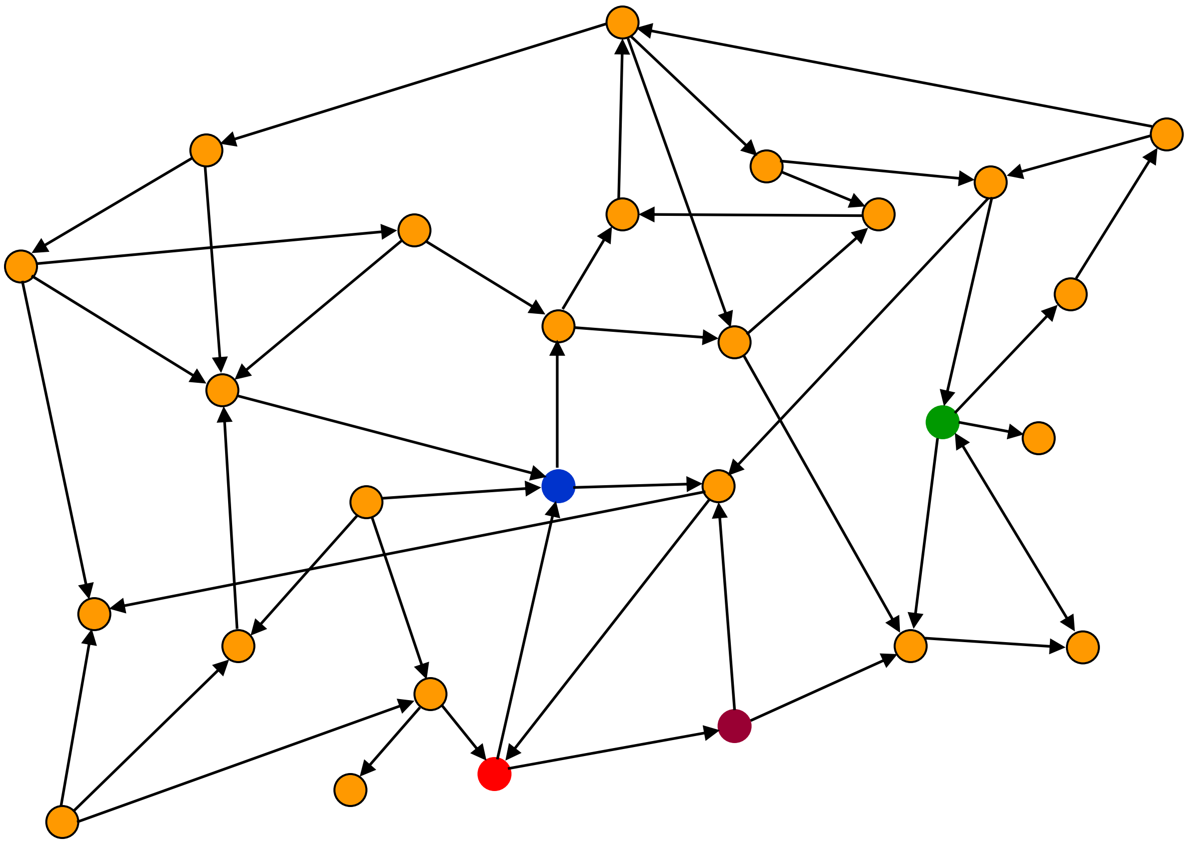
- Often it is not feasible (or too expensive) to build a complete representation of the state graph
- A problem solver must construct a solution by exploring a small portion of the graph

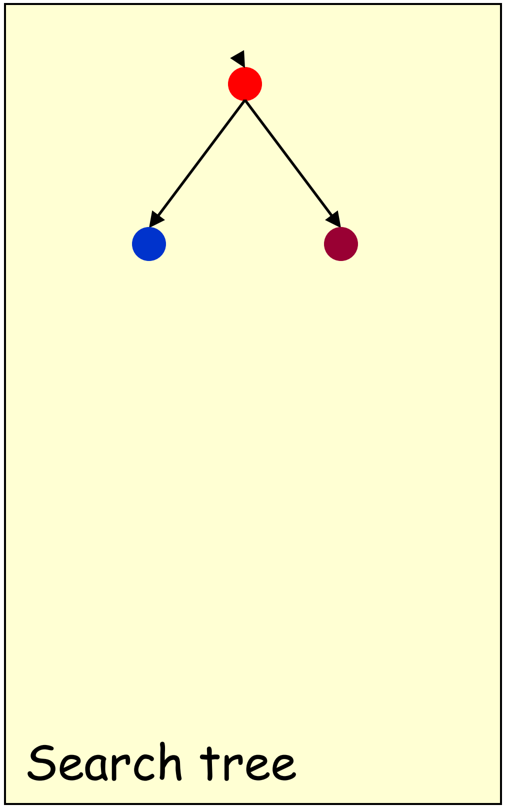
Searching the State Space



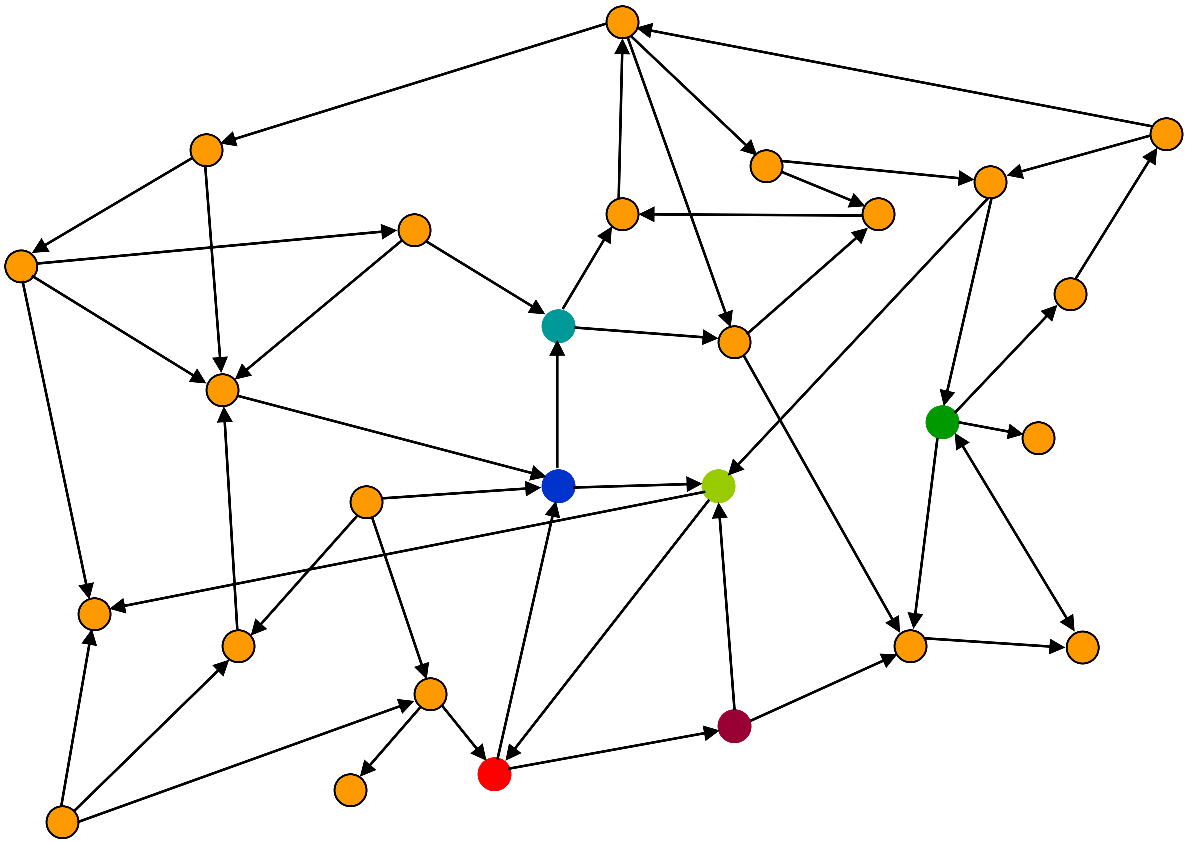


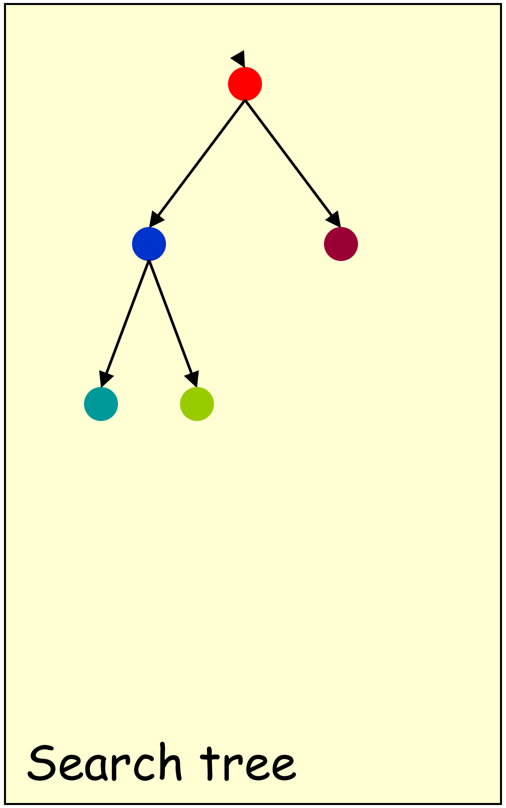
Searching the State Space



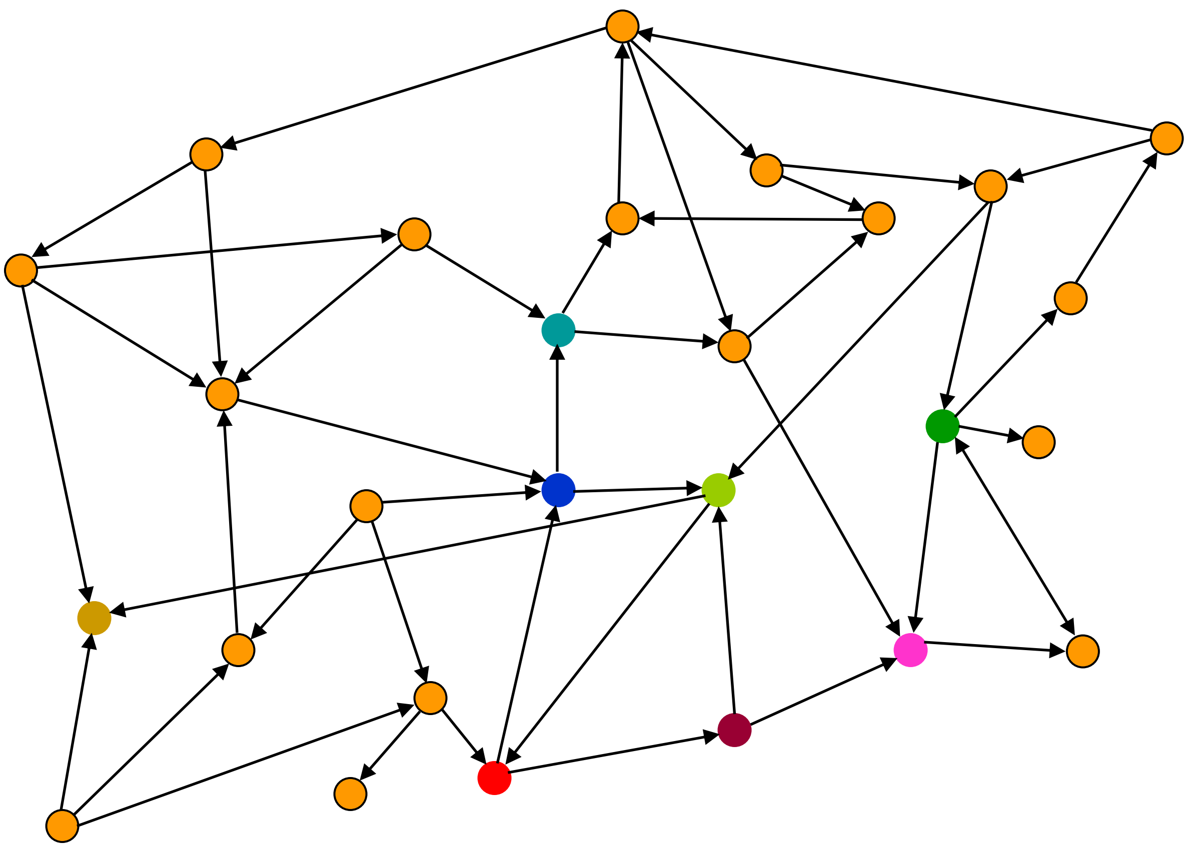


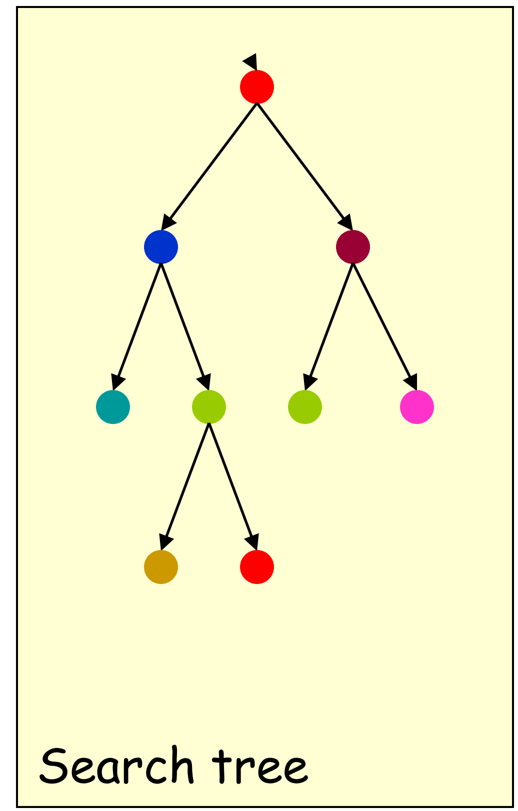
Searching the State Space



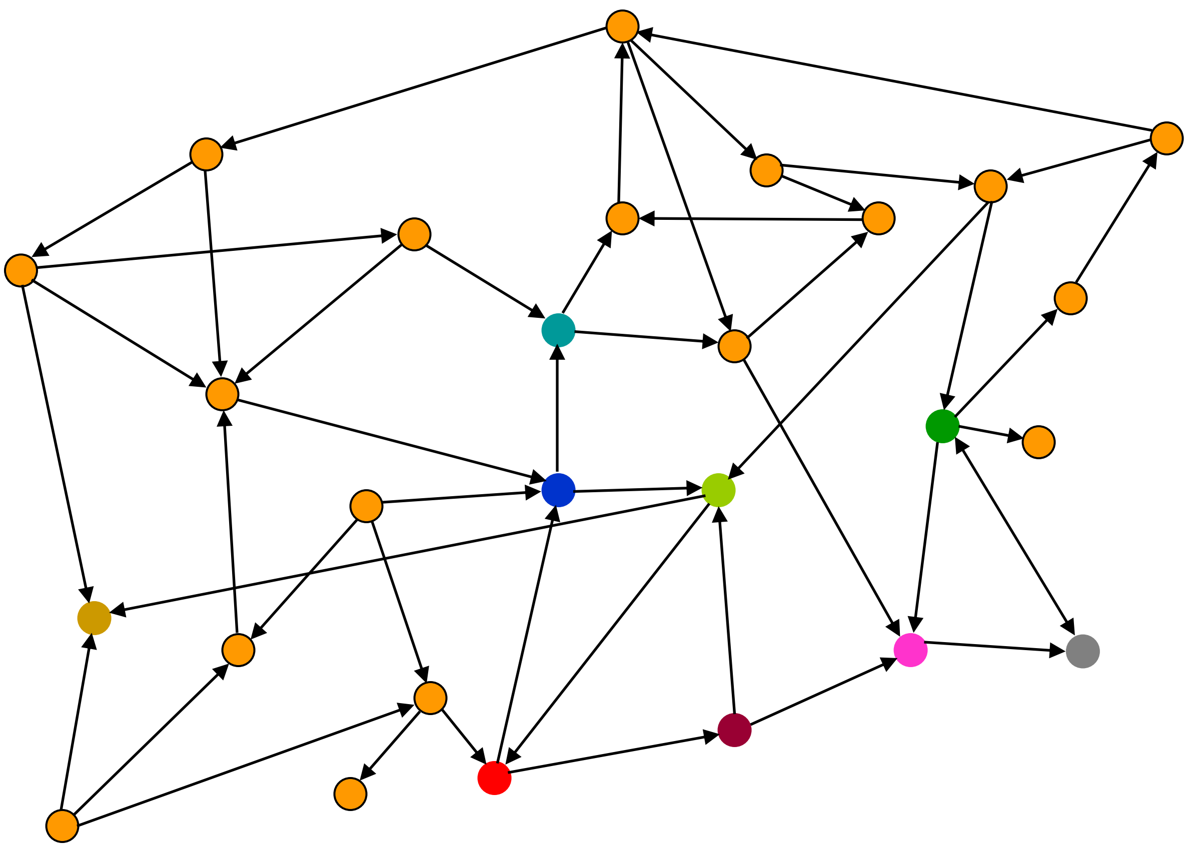


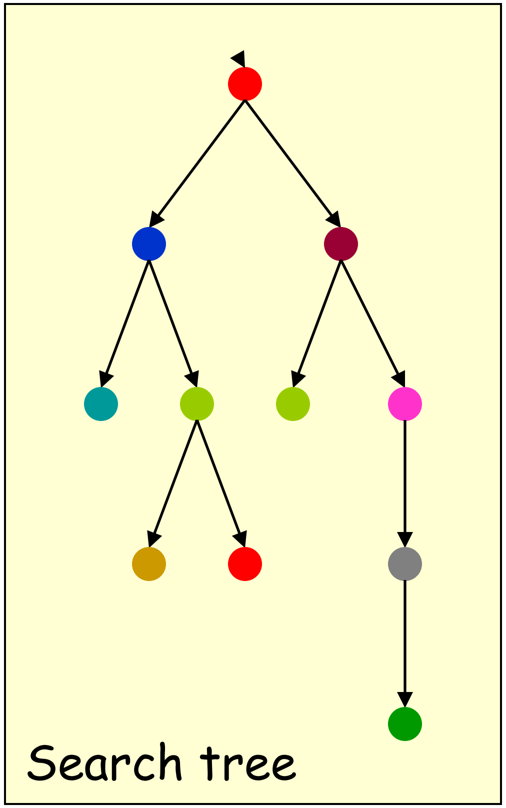
Searching the State Space



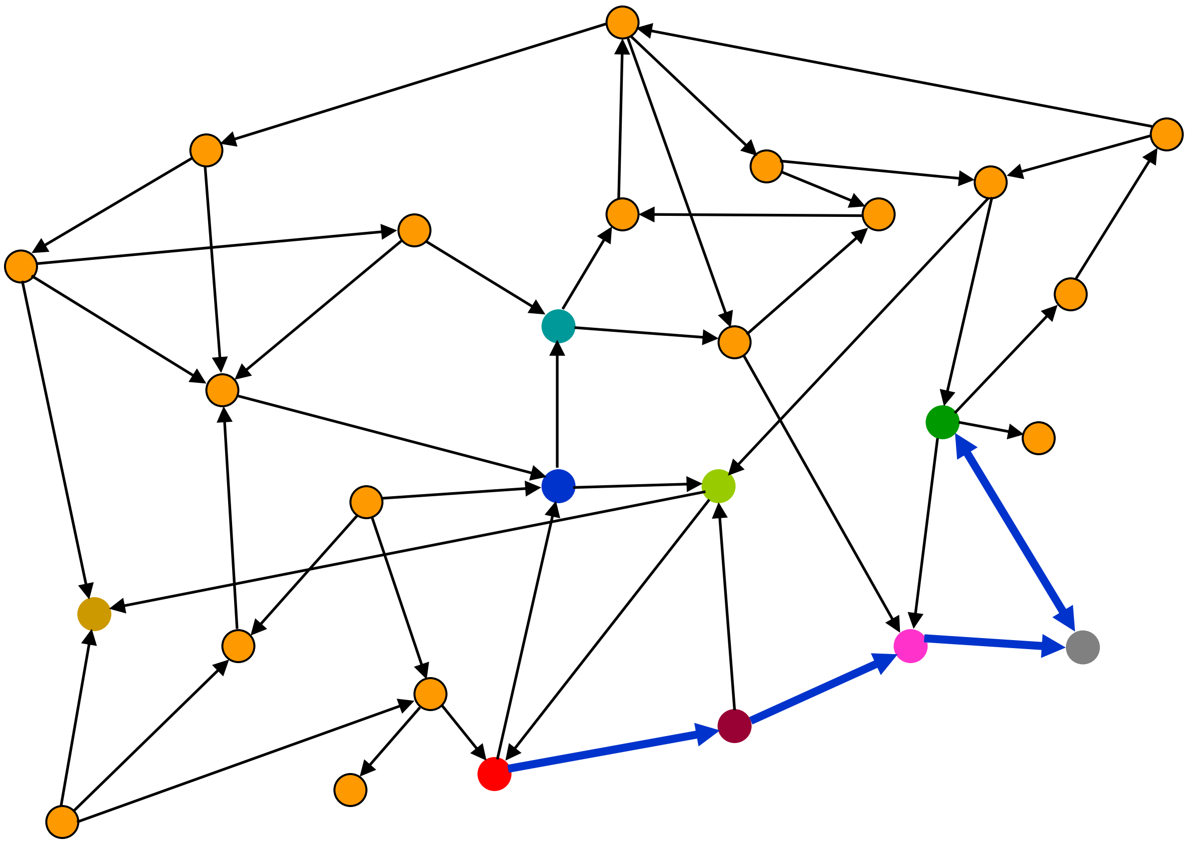


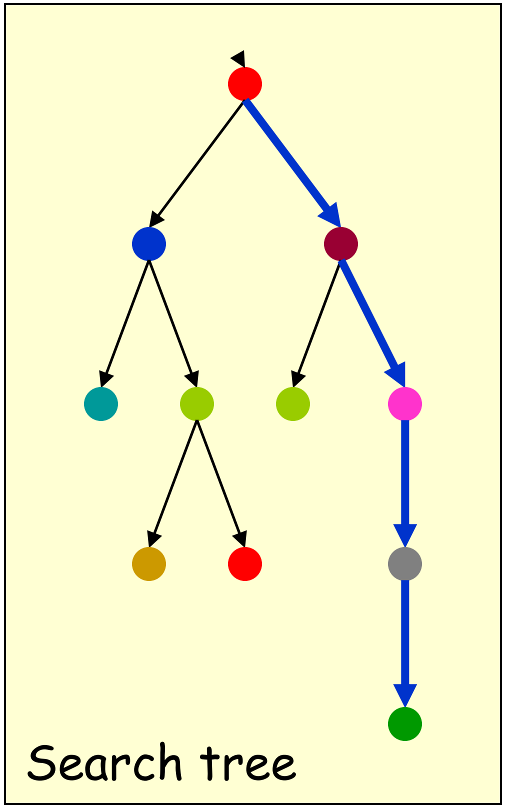
Searching the State Space





Searching the State Space





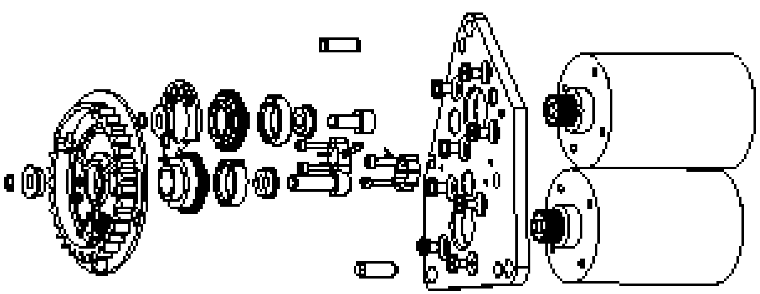
Simple Problem-Solving-Agent Algorithm

1. I ← sense/read initial state
2. GOAL? ← select/read goal test
3. Succ ← select/read successor function
4. solution ← **search**(I, GOAL?, Succ)
5. perform(solution)

State Space

- Each state is an abstract representation of a collection of possible worlds sharing some crucial properties and differing on non-important details only

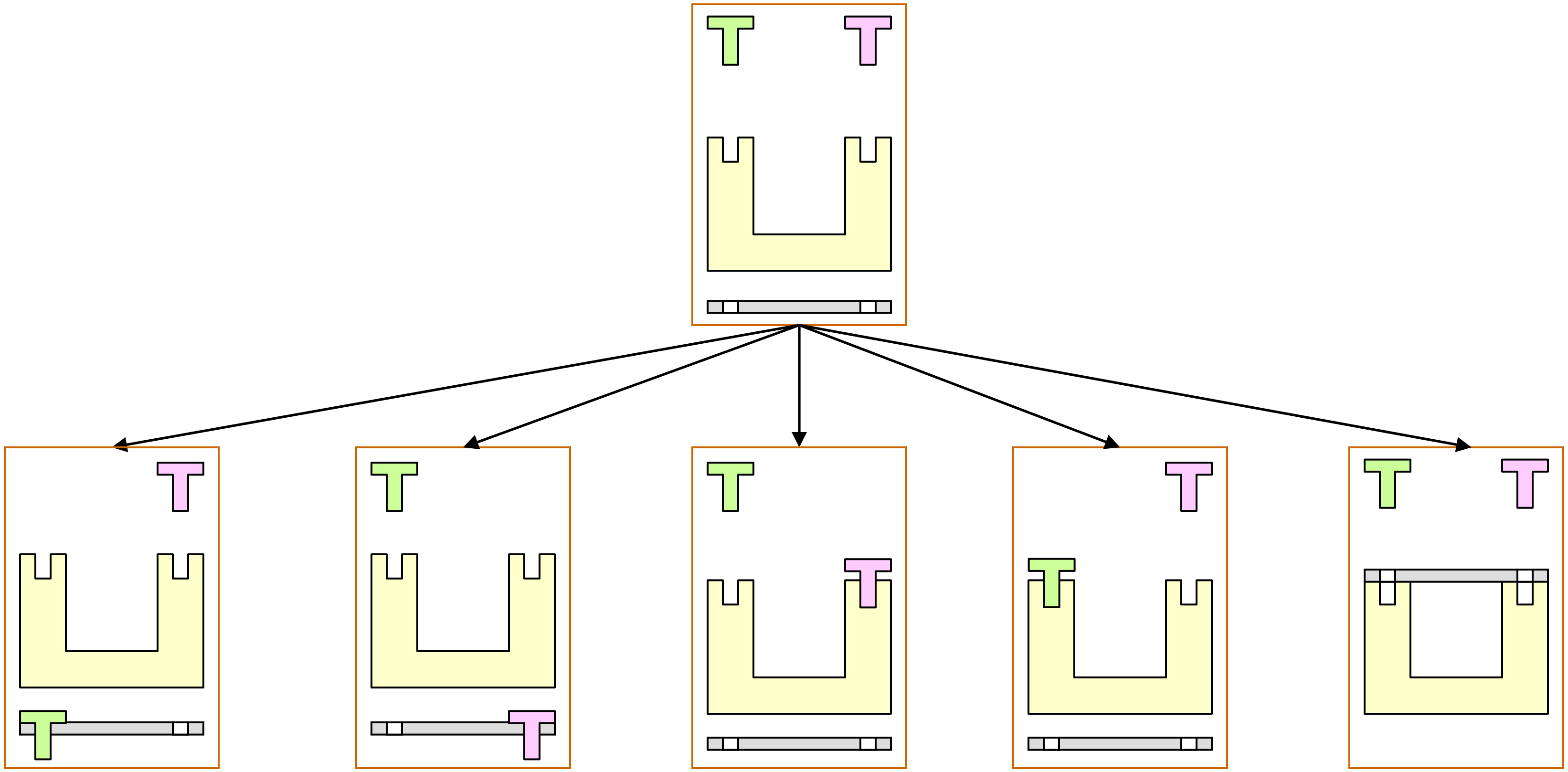
  E.g.: In assembly planning, a state does not define exactly the absolute position of each part



- The state space is discrete. It may be finite, or infinite

Successor Function

- It implicitly represents all the actions that are feasible in each state



Successor Function

- It implicitly represents all the actions that are feasible in each state
- Only the results of the actions (the successor states) and their costs are returned by the function
- The successor function is a "black box": its content is unknown

  E.g., in assembly planning, the successor function may be quite complex (collision, stability, grasping, ...)

Path Cost

- An arc cost is a positive number measuring the "cost" of performing the action corresponding to the arc, e.g.:
  - 1 in the 8-puzzle example
  - expected time to merge two sub-assemblies
- We will assume that for any given problem the cost c of an arc always verifies: $c \geq \varepsilon > 0$, where ε is a constant

Path Cost

- An arc cost is a positive number measuring the "cost" of performing the action corresponding to the arc, e.g.:
  - 1 in the 8-puzzle example
  - expected time to merge two sub-assemblies
- We will assume that for any given problem the cost c of an arc always verifies: $c \geq \varepsilon > 0$, where ε is a constant
  [This condition guarantees that, if path becomes arbitrarily long, its cost also becomes arbitrarily large]

Why is this needed?

Goal State

- It may be explicitly described:

1	2	3
4	5	6
7	8	

- or partially described:

1	a	a
a	5	a
a	8	a

("a" stands for "any" other than 1, 5, and 8)

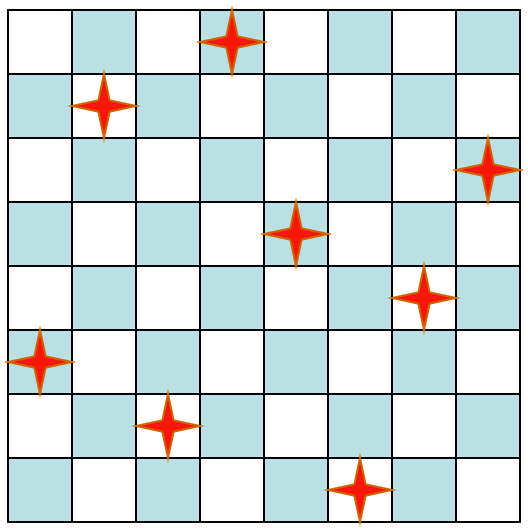
- or defined by a condition,
  e.g., the sum of every row, of every column, and of every diagonal equals 30

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

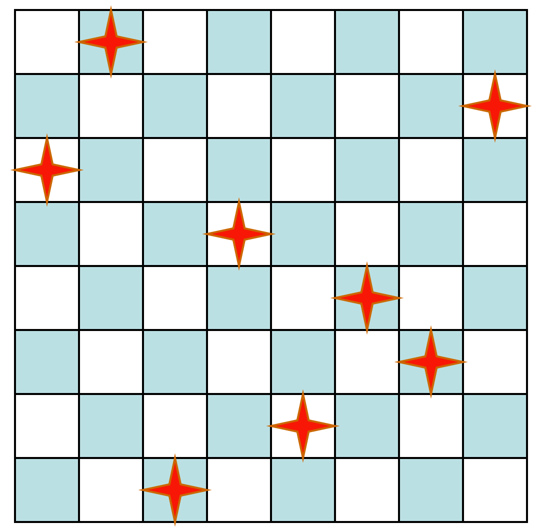
Other examples

8-Queens Problem

Place 8 queens in a chessboard so that no two queens are in the same row, column, or diagonal.

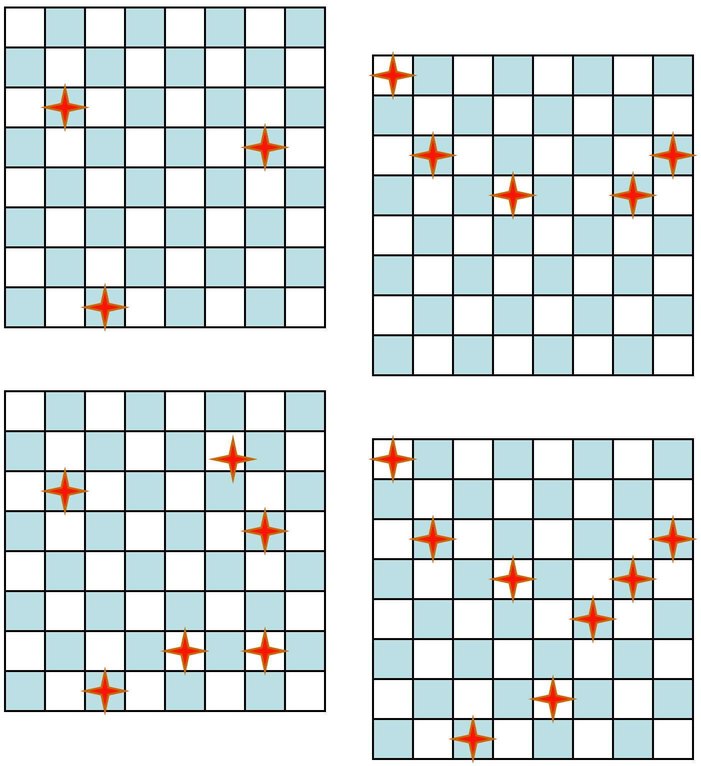


A solution



Not a solution

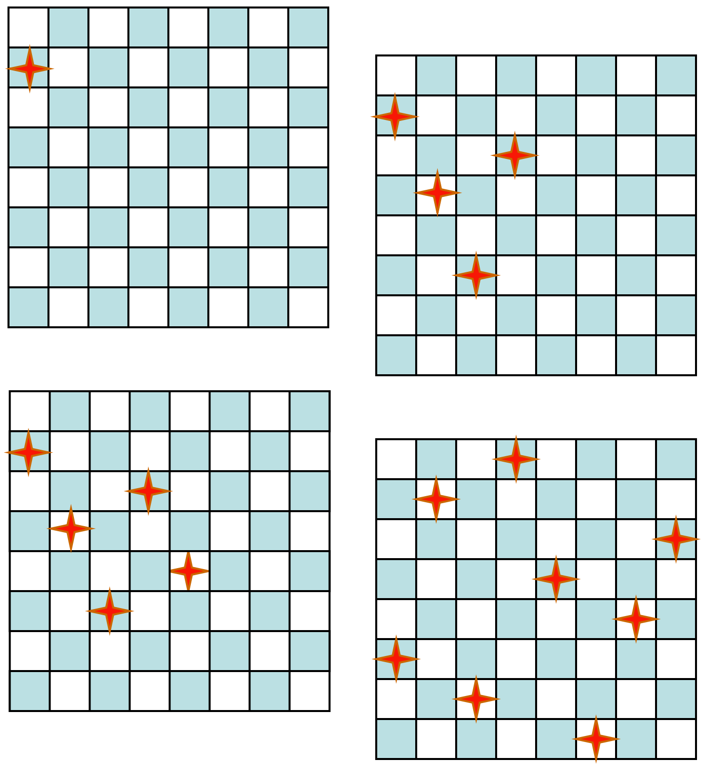
Formulation #1



- **States:** all arrangements of 0, 1, 2, ..., 8 queens on the board
- **Initial state:** 0 queens on the board
- **Successor function:** each of the successors is obtained by adding one queen in an empty square
- **Arc cost:** irrelevant
- **Goal test:** 8 queens are on the board, with no queens attacking each other

→ ~ 64x63x...x57 ~ 3×10^{14} states

Formulation #2



→ 2,057 states

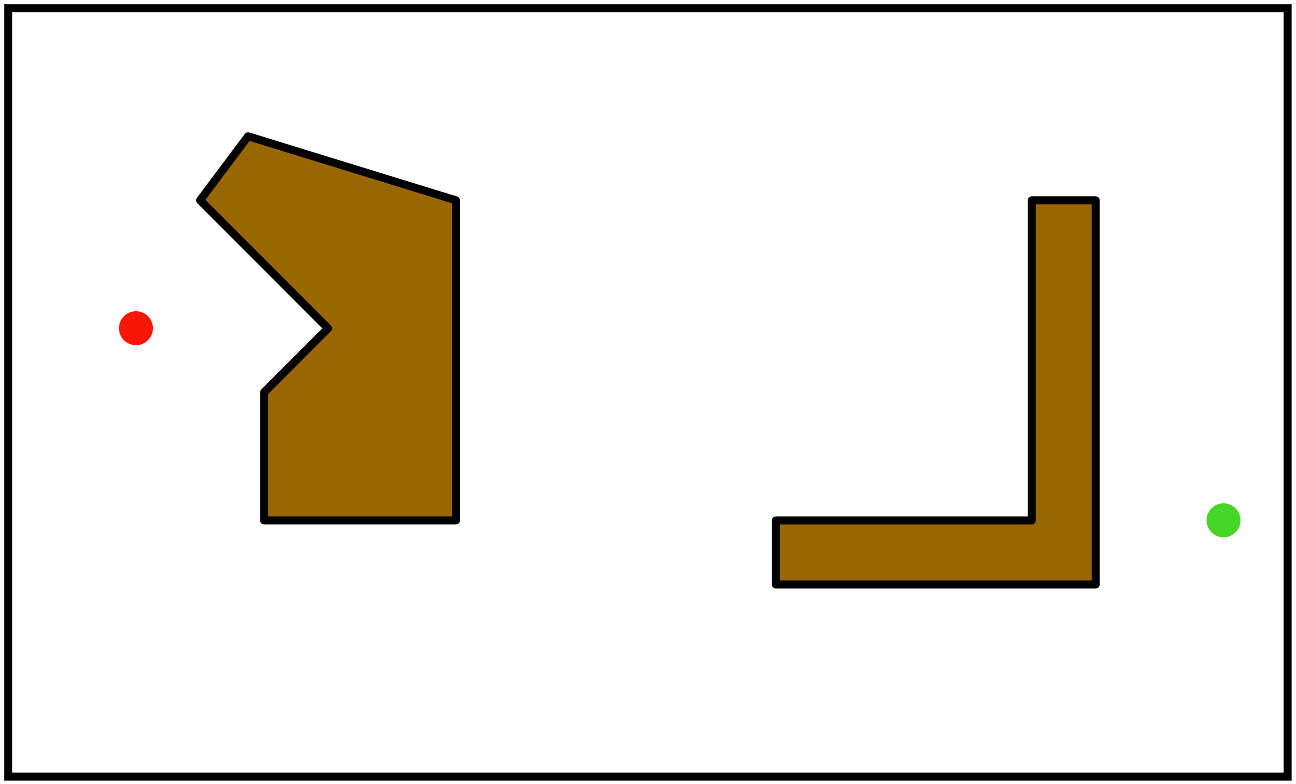
- **States:** all arrangements of k = 0, 1, 2, ..., 8 queens in the k leftmost columns with no two queens attacking each other
- **Initial state:** 0 queens on the board
- **Successor function:** each successor is obtained by adding one queen in any square that is not attacked by any queen already in the board, in the leftmost empty column
- **Arc cost:** irrelevant
- **Goal test:** 8 queens are on the board

n-Queens Problem

- A solution is a goal node, not a path to this node (typical of design problem)
- Number of states in state space:
  - 8-queens → 2,057
  - 100-queens → 10^{52}
- But techniques exist to solve n-queens problems efficiently for large values of n

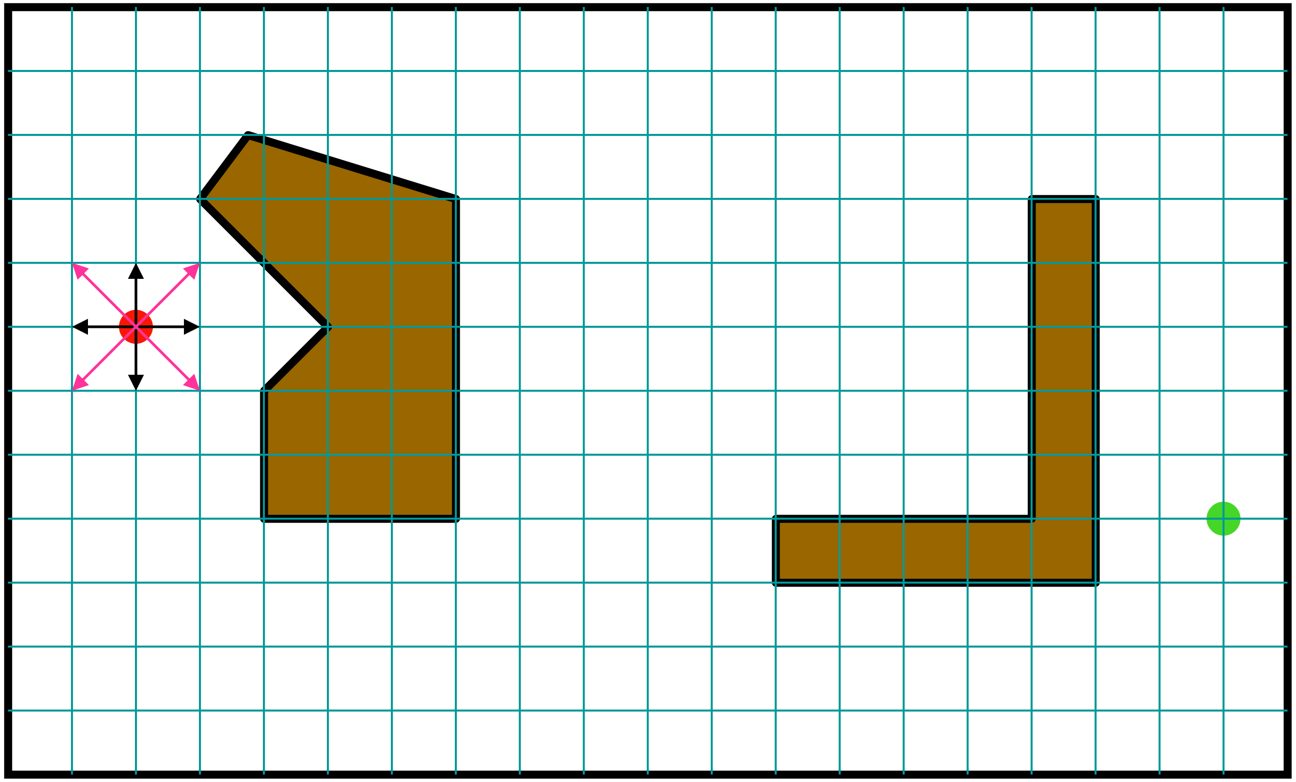
  They exploit the fact that there are many solutions well distributed in the state space

Path Planning



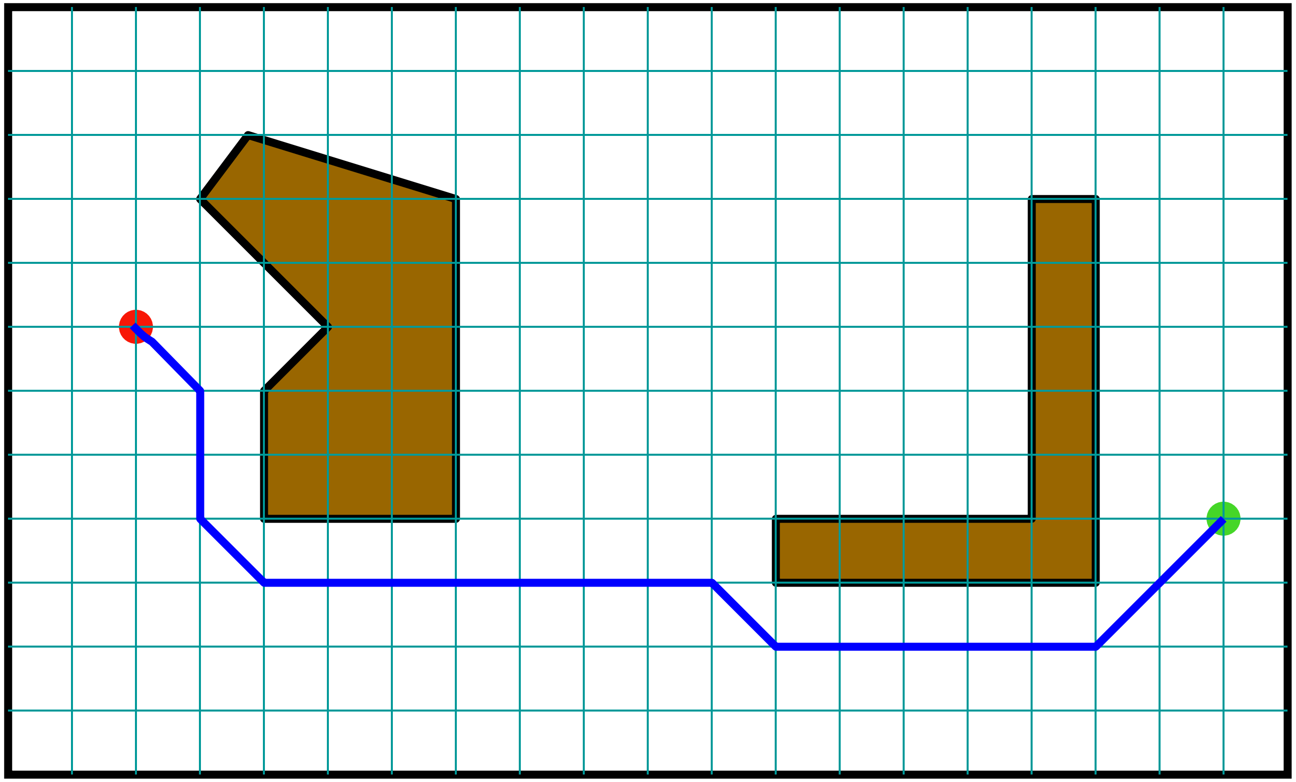
What is the state space?

Formulation #1



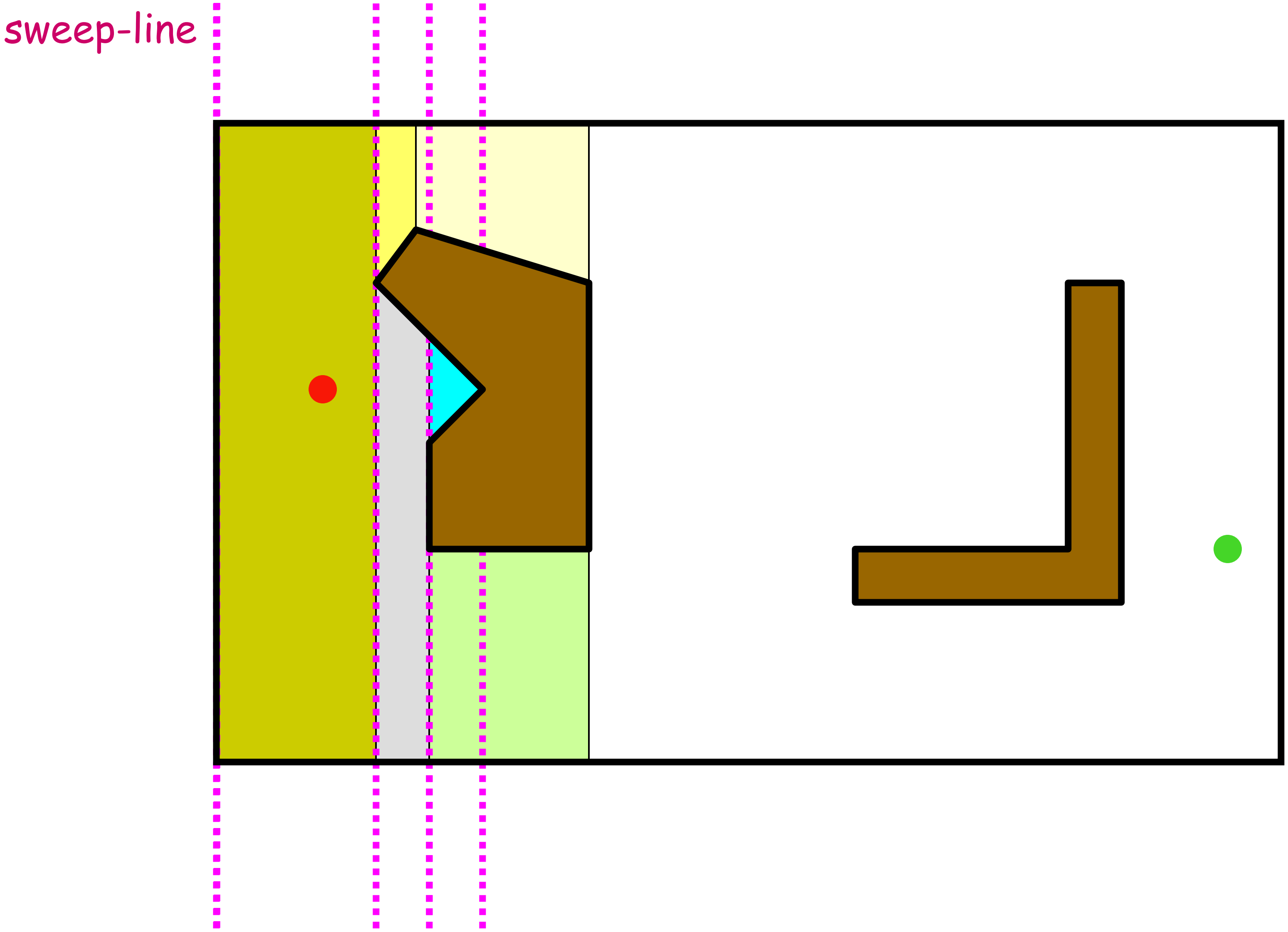
Cost of one horizontal/vertical step = 1
Cost of one diagonal step = $\sqrt{2}$

Optimal Solution

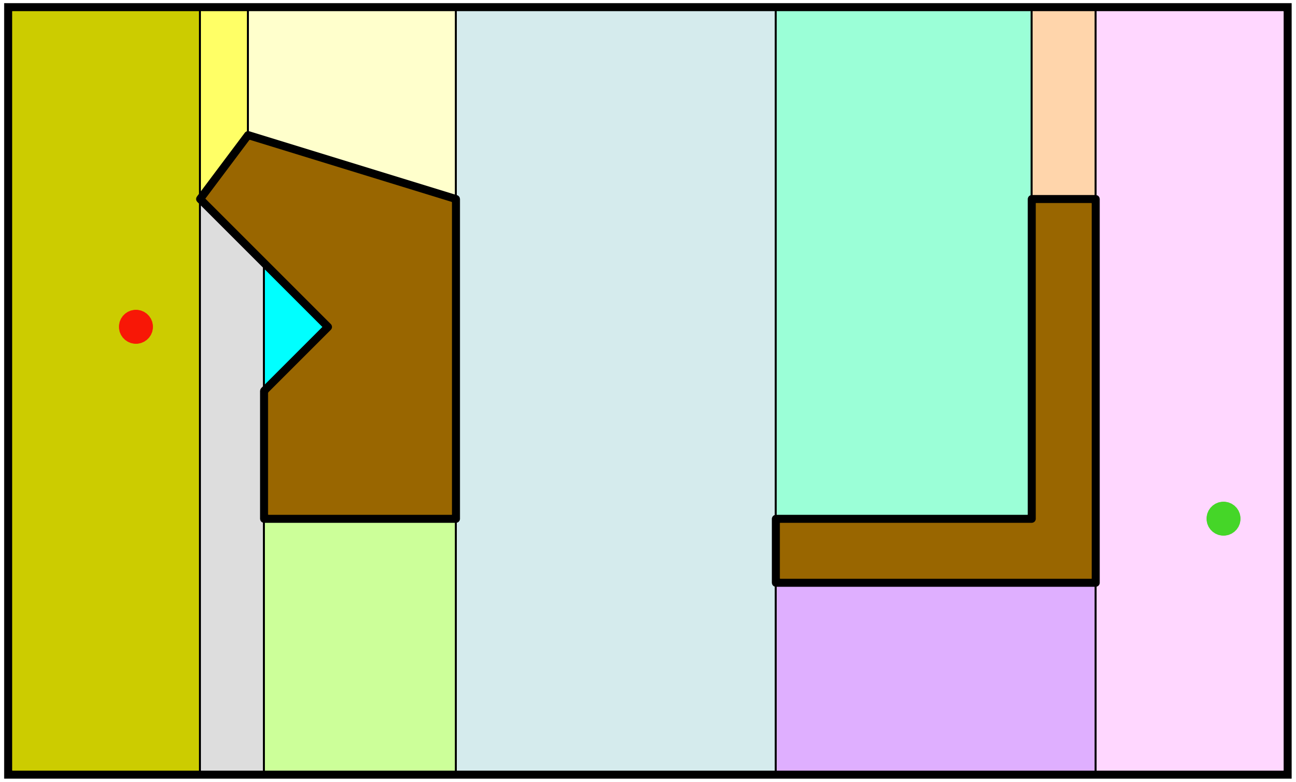


This path is the shortest in the discretized state space, but not in the original continuous space

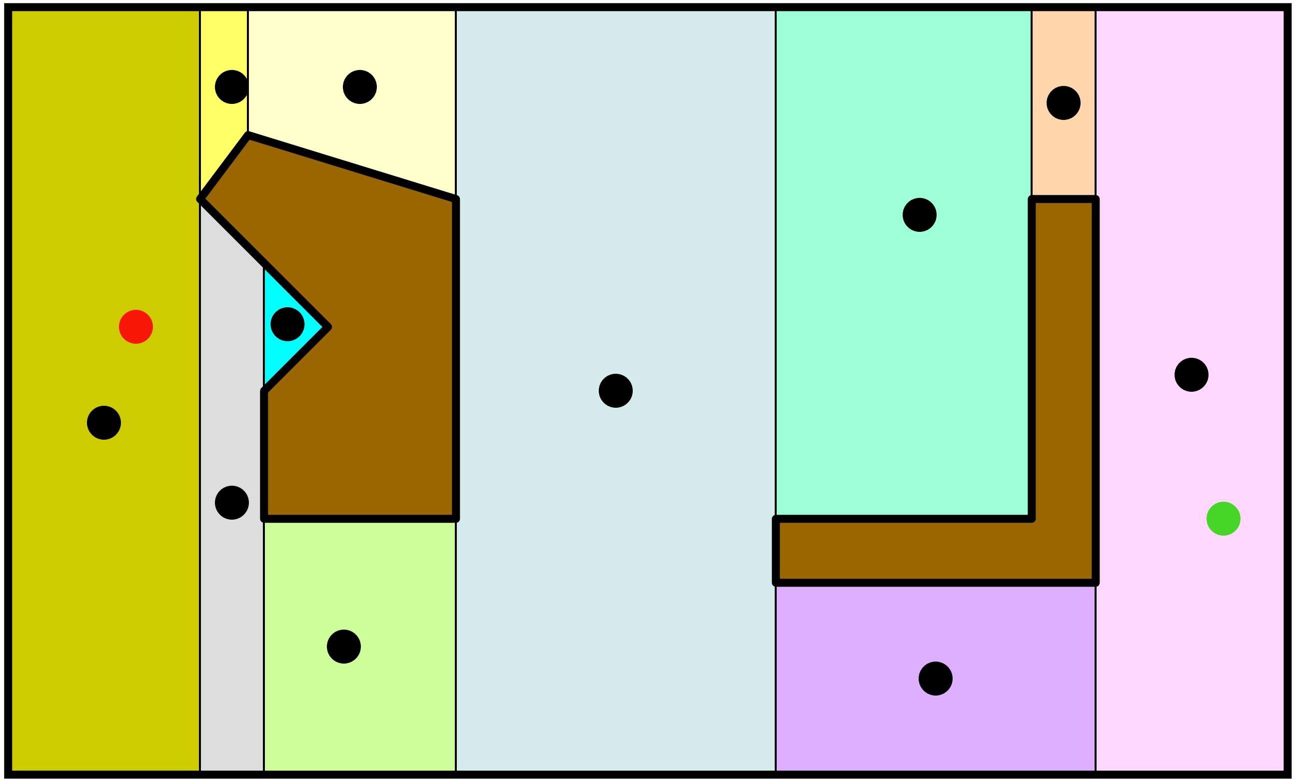
Formulation #2



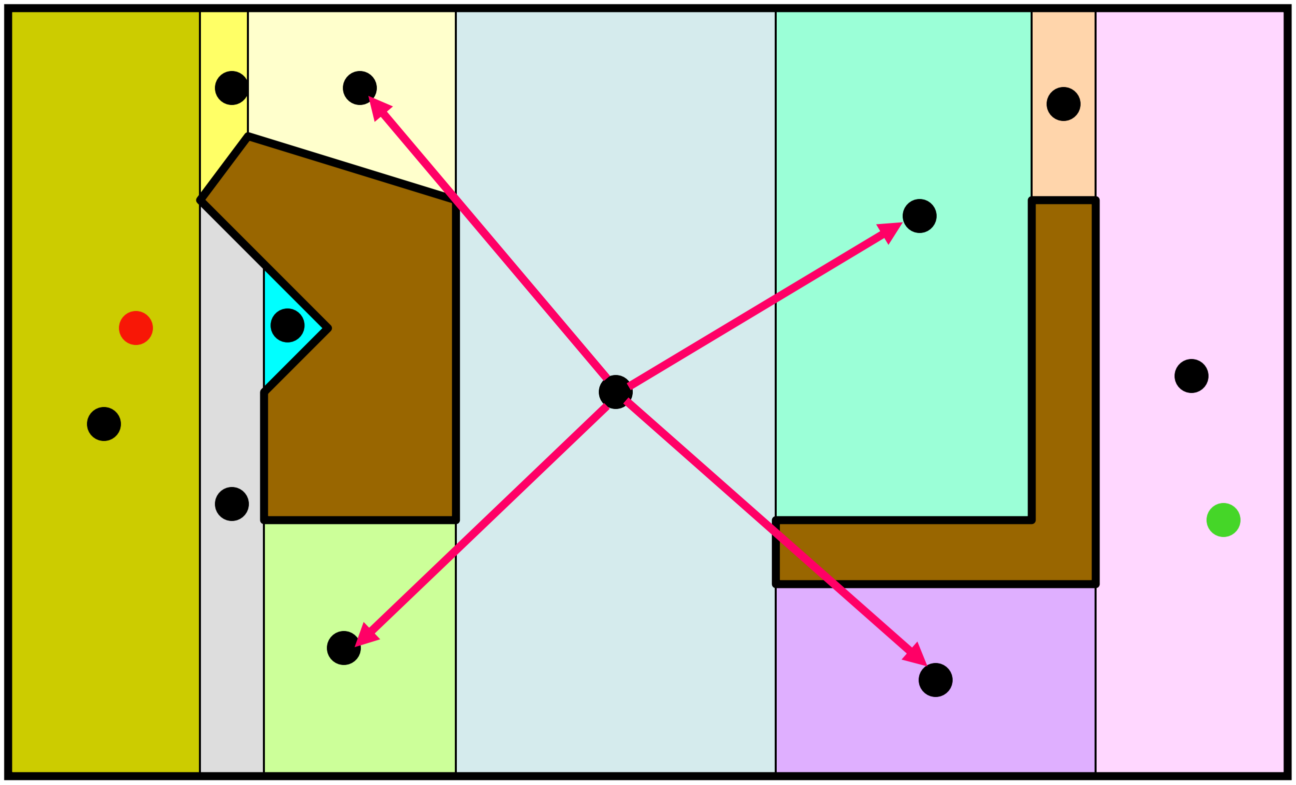
Formulation #2



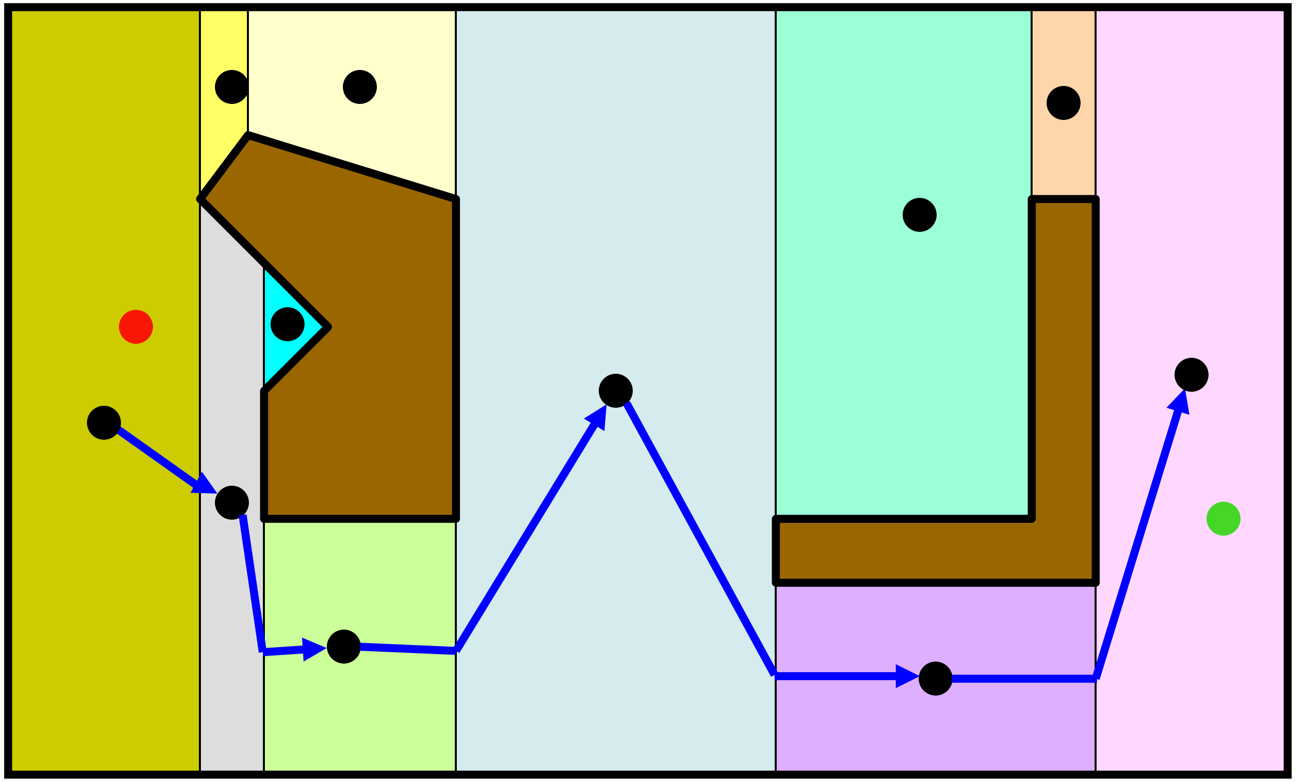
States



Successor Function

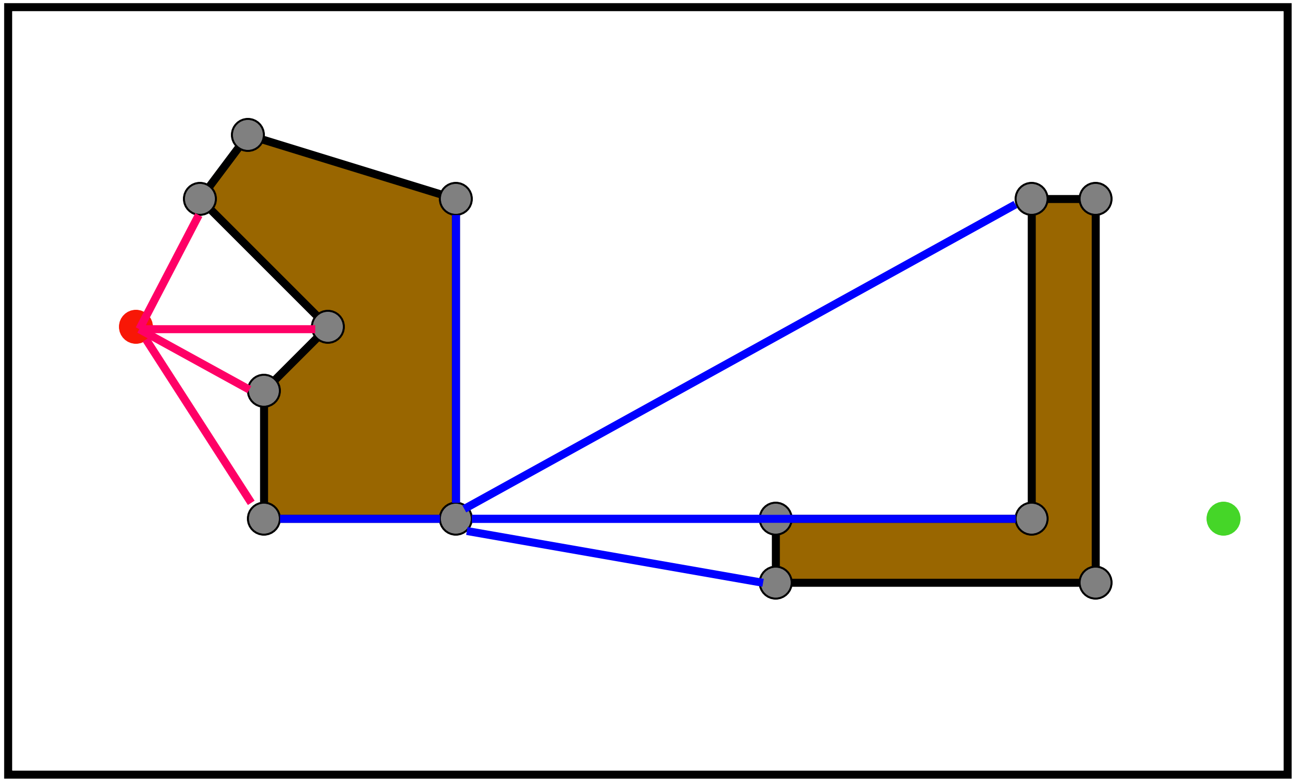


Solution Path



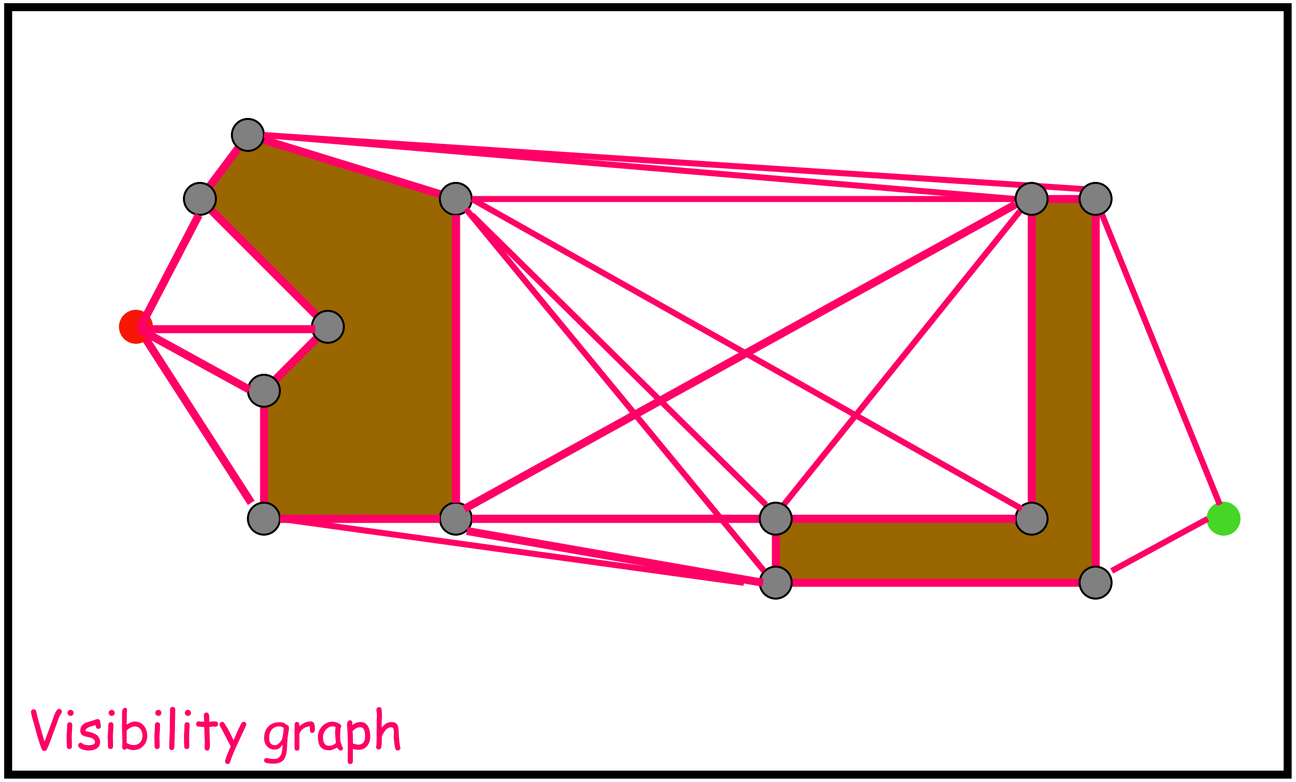
A path-smoothing post-processing step is usually needed to shorten the path further

Formulation #3



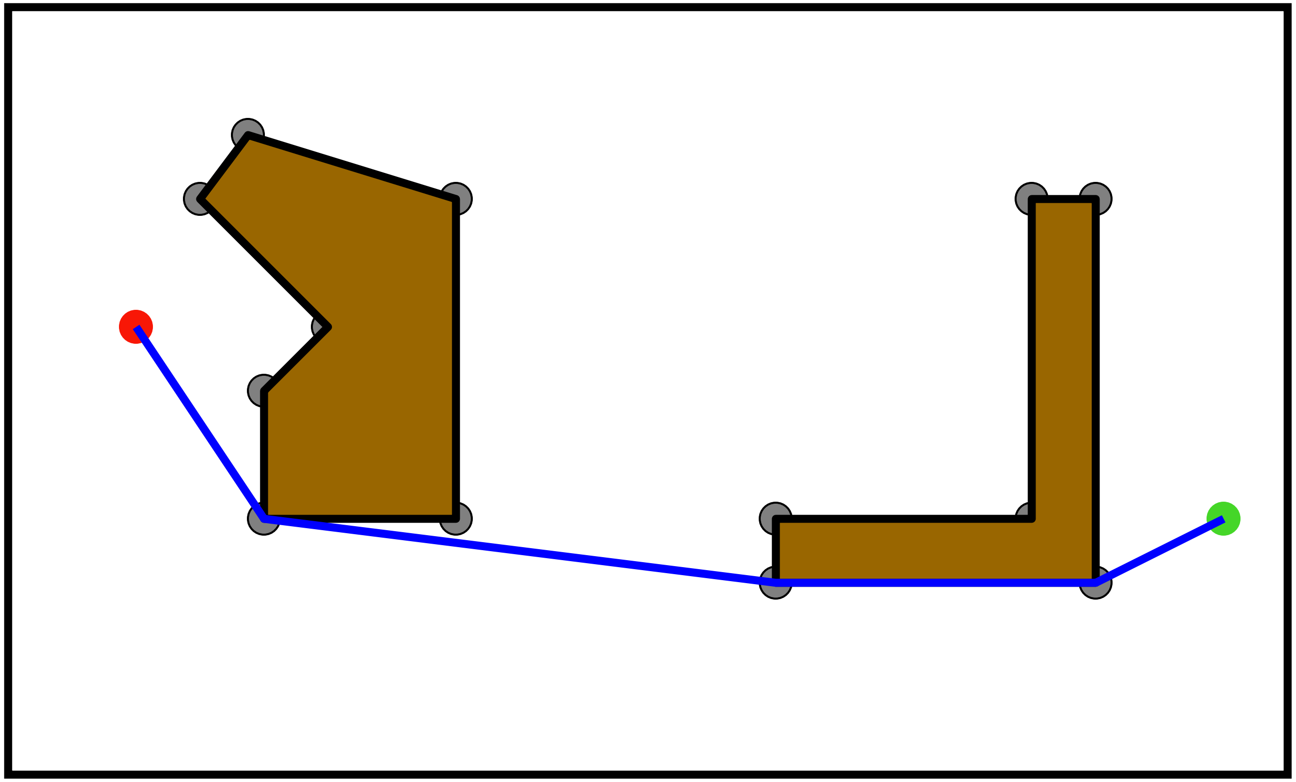
Cost of one step: length of segment

Formulation #3



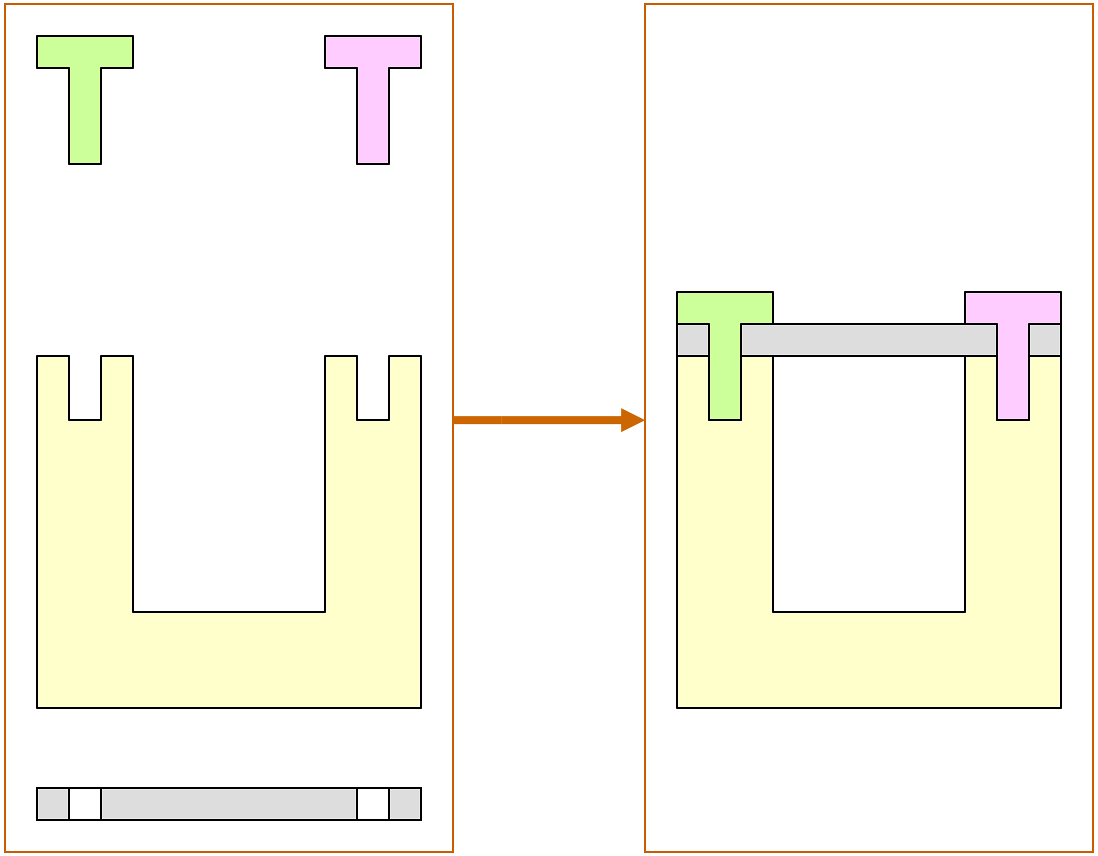
Cost of one step: length of segment

Solution Path



The shortest path in this state space is also the shortest in the original continuous space

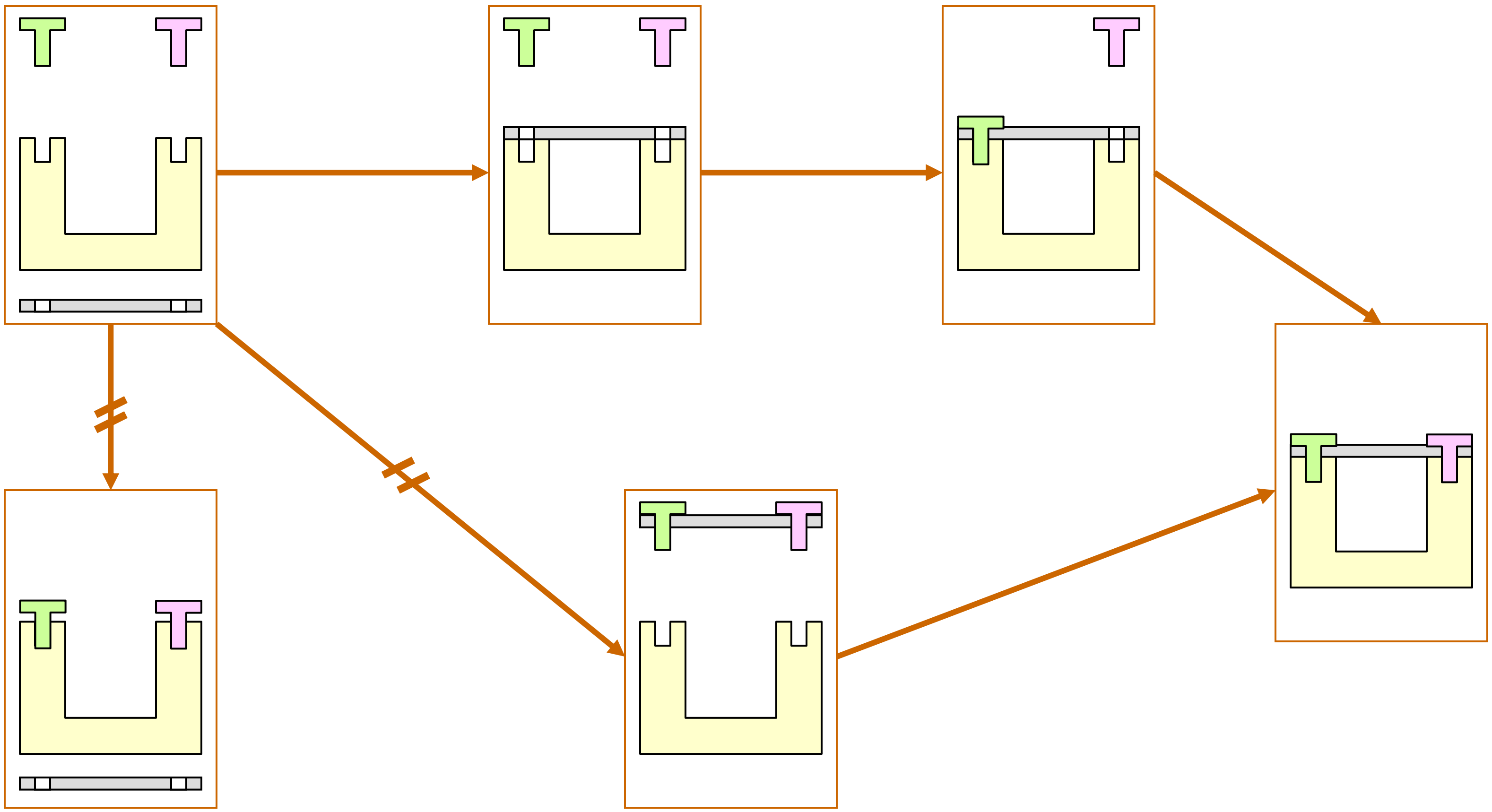
Assembly (Sequence) Planning



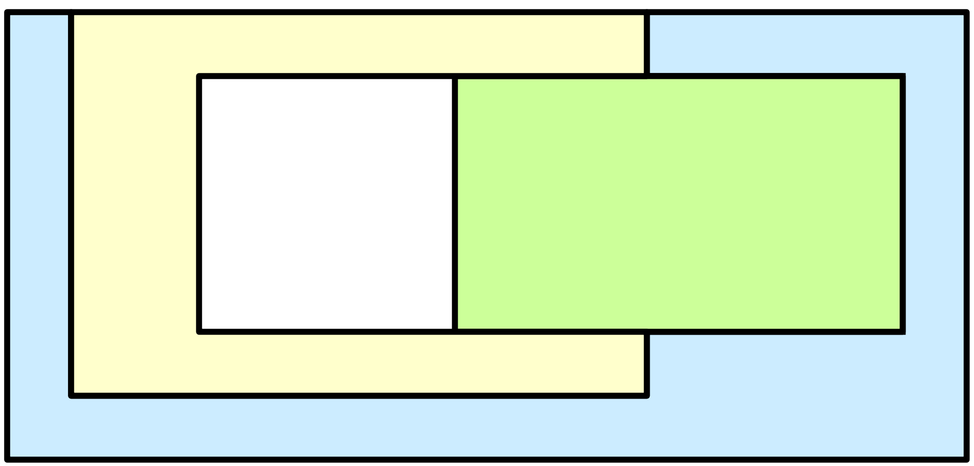
Possible Formulation

- **States:** All decompositions of the assembly into subassemblies (subsets of parts in their relative placements in the assembly)
- **Initial state:** All subassemblies are made of a single part
- **Goal state:** Un-decomposed assembly
- **Successor function:** Each successor of a state is obtained by merging two subassemblies (the successor function must check if the merging is feasible: collision, stability, grasping, ...)
- **Arc cost:** 1 or time to carry the merging

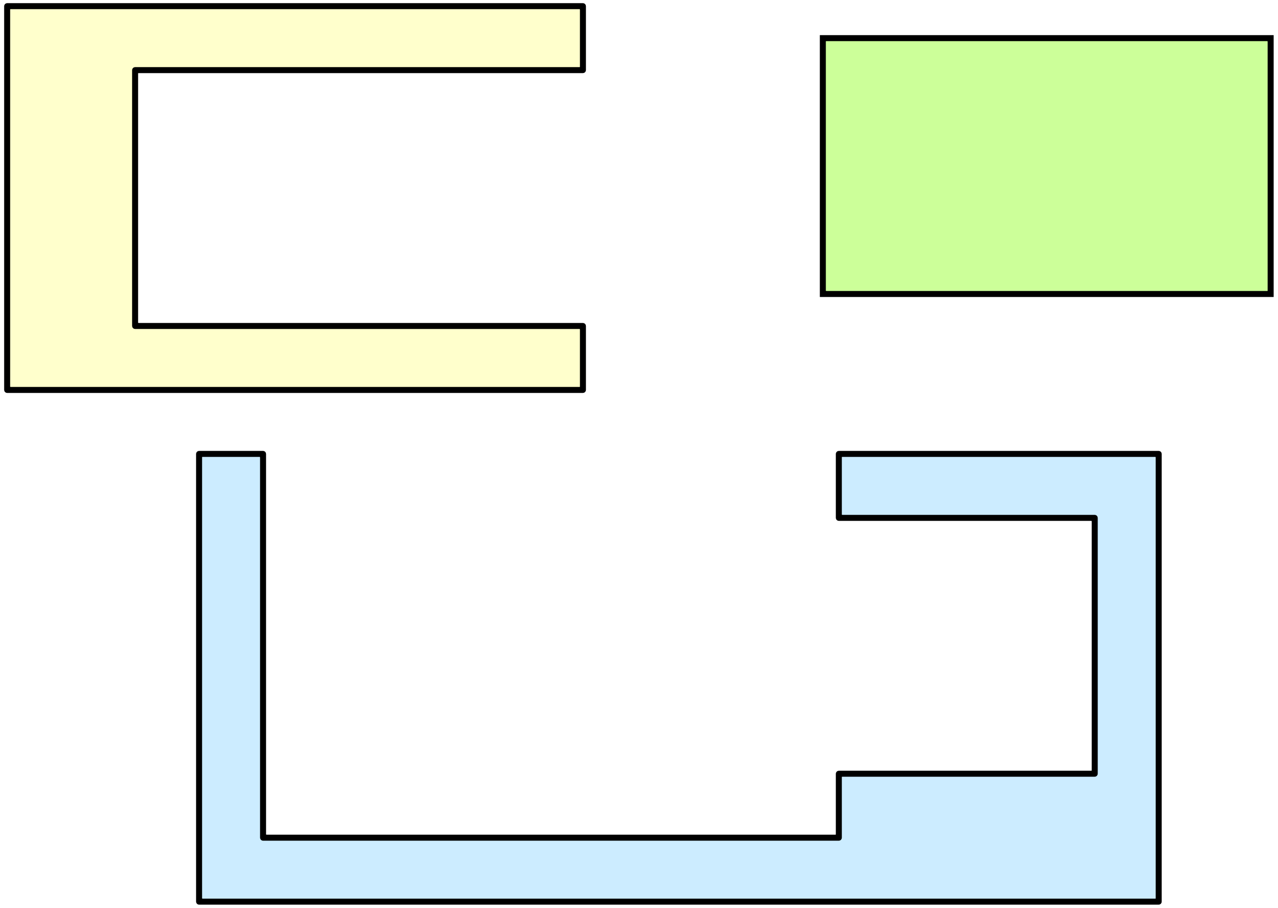
A Portion of State Space



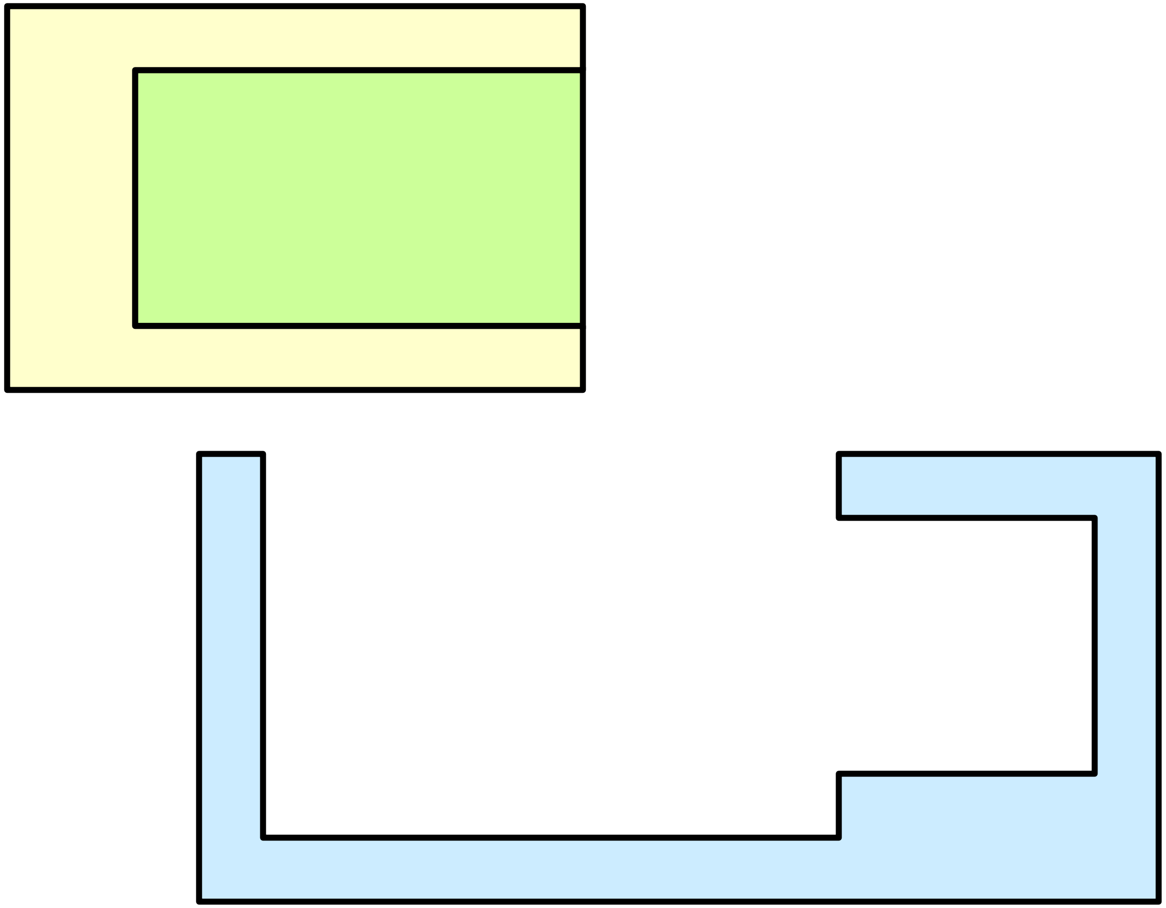
But the formulation rules out "non-monotonic" assemblies



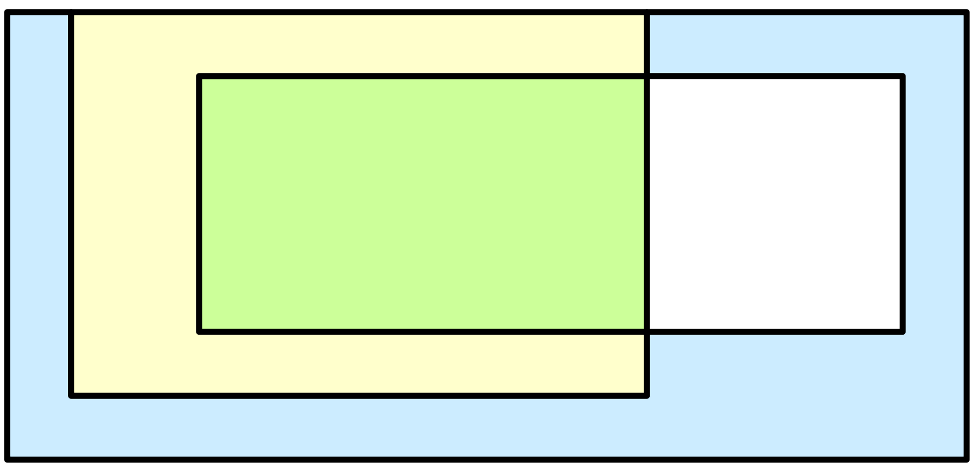
But the formulation rules out "non-monotonic" assemblies



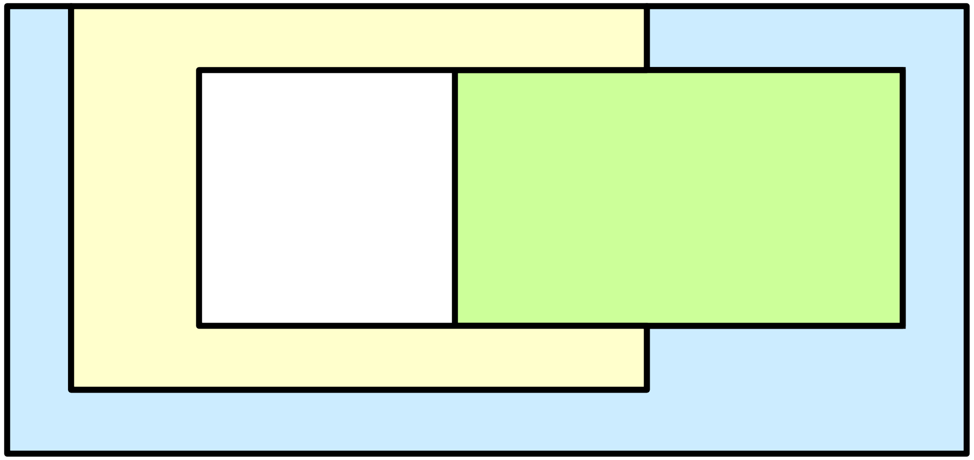
But the formulation rules out “non-monotonic” assemblies



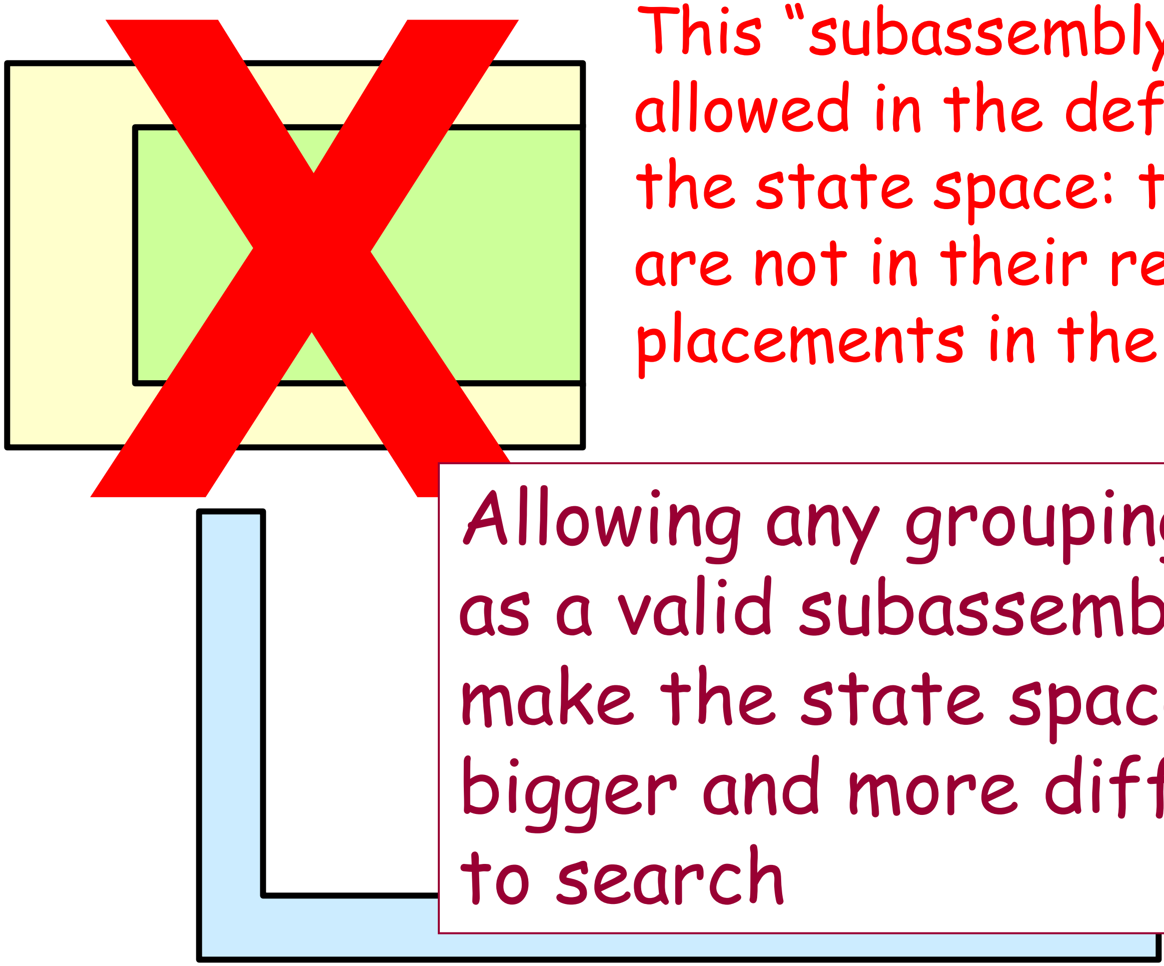
But the formulation rules out “non-monotonic” assemblies



But the formulation rules out "non-monotonic" assemblies



But the formulation rules out "non-monotonic" assemblies



This "subassembly" is not allowed in the definition of the state space: the 2 parts are not in their relative placements in the assembly

Allowing any grouping of parts as a valid subassembly would make the state space much bigger and more difficult to search

Assumptions in Basic Search

- The world is static
- The world is discretizable
- The world is observable
- The actions are deterministic

But many of these assumptions can be removed, and search still remains an important problem-solving tool

Search and AI

- Search methods are ubiquitous in AI systems. They often are the backbones of both core and peripheral modules
- An autonomous robot uses search methods:
  - to decide which actions to take and which sensing operations to perform,
  - to quickly anticipate collision,
  - to plan trajectories,
  - to interpret large numerical datasets provided by sensors into compact symbolic representations,
  - to diagnose why something did not happen as expected,
  - etc...
- Many searches may occur concurrently and sequentially

Applications

Search plays a key role in many applications, e.g.:

- Route finding: airline travel, networks
- Package/mail distribution
- Pipe routing, VLSI routing
- Comparison and classification of protein folds
- Pharmaceutical drug design
- Design of protein-like molecules
- Video games