Continuous Space & Nondeterministic Actions

Outline

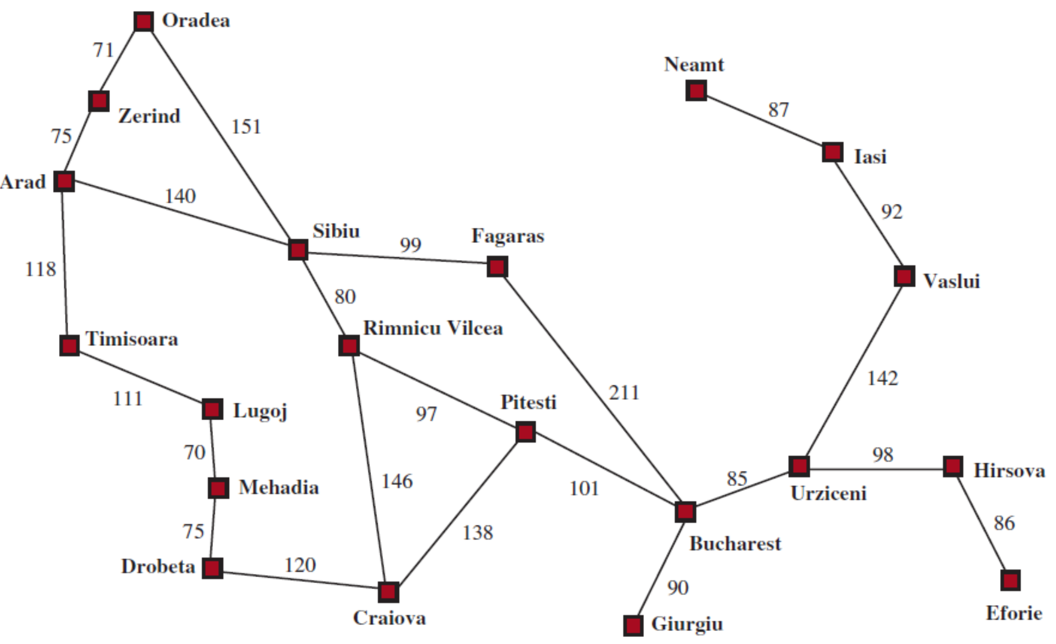
I. Local search in continuous spaces

II. Search with non-deterministic actions

* Figures are from the textbook site (or by the instructor) unless the source is specifically cited.

I. Local Search in Continuous Space

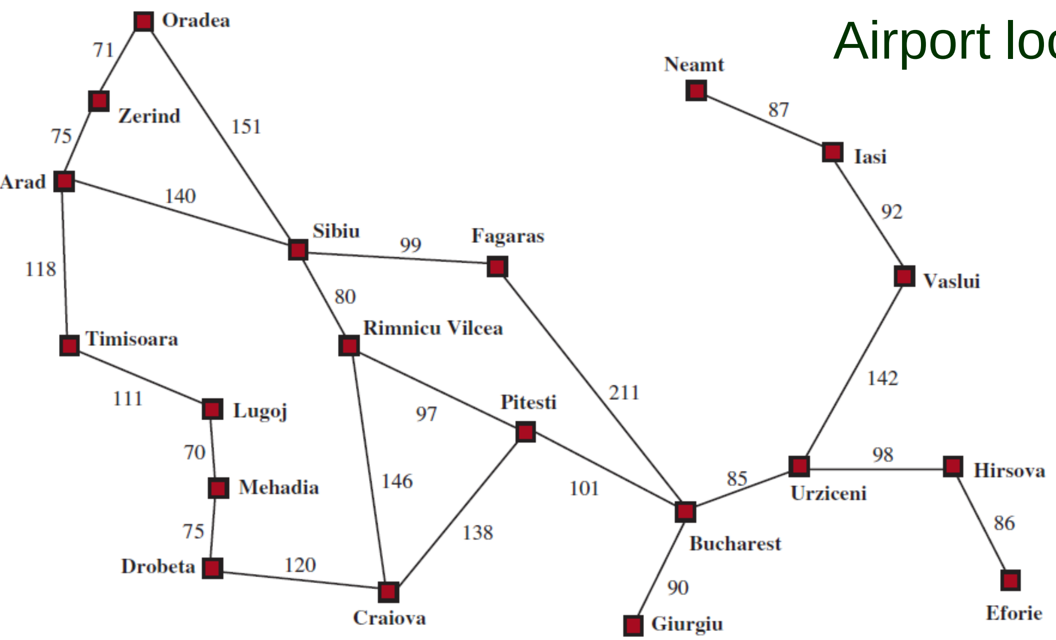
Problem Place three new airports anywhere in Romania to minimize the sum of square straight-line distances from every city to its nearest airport.



I. Local Search in Continuous Space

Problem Place three new airports anywhere in Romania to minimize the sum of square straight-line distances from every city to its nearest airport.

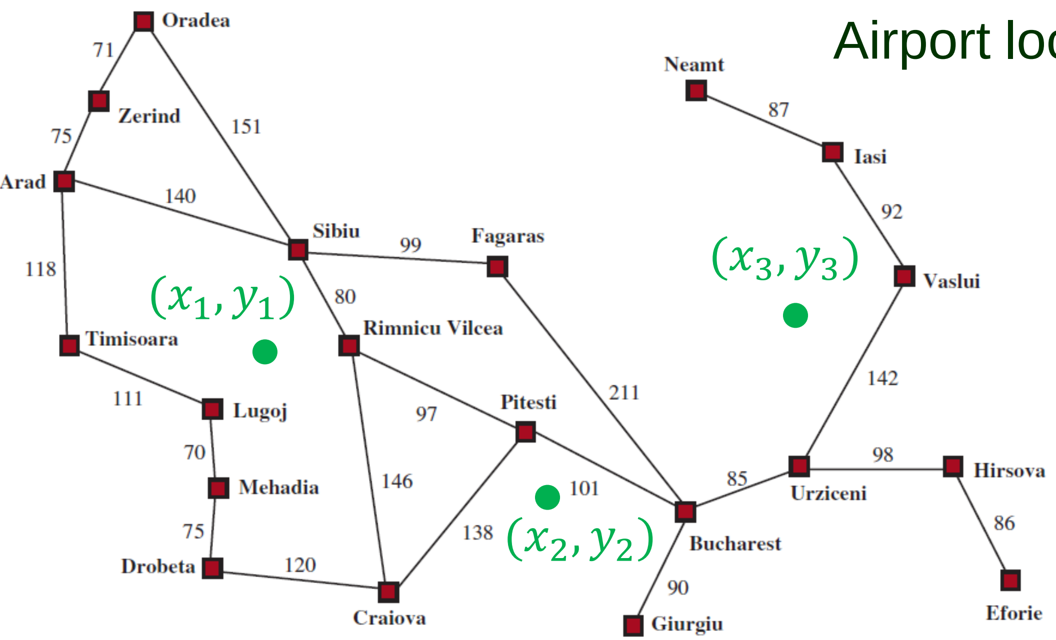
Airport locations: (x_1, y_1), (x_2, y_2), (x_3, y_3)



I. Local Search in Continuous Space

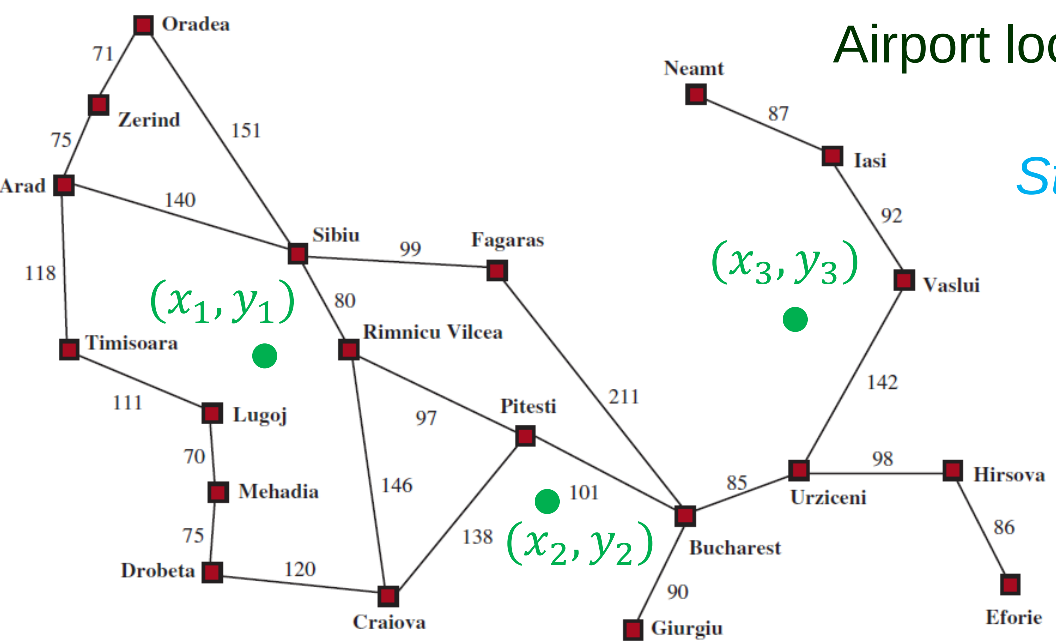
Problem Place three new airports anywhere in Romania to minimize the sum of square straight-line distances from every city to its nearest airport.

Airport locations: (x_1, y_1), (x_2, y_2), (x_3, y_3)



I. Local Search in Continuous Space

Problem Place three new airports anywhere in Romania to minimize the sum of square straight-line distances from every city to its nearest airport.

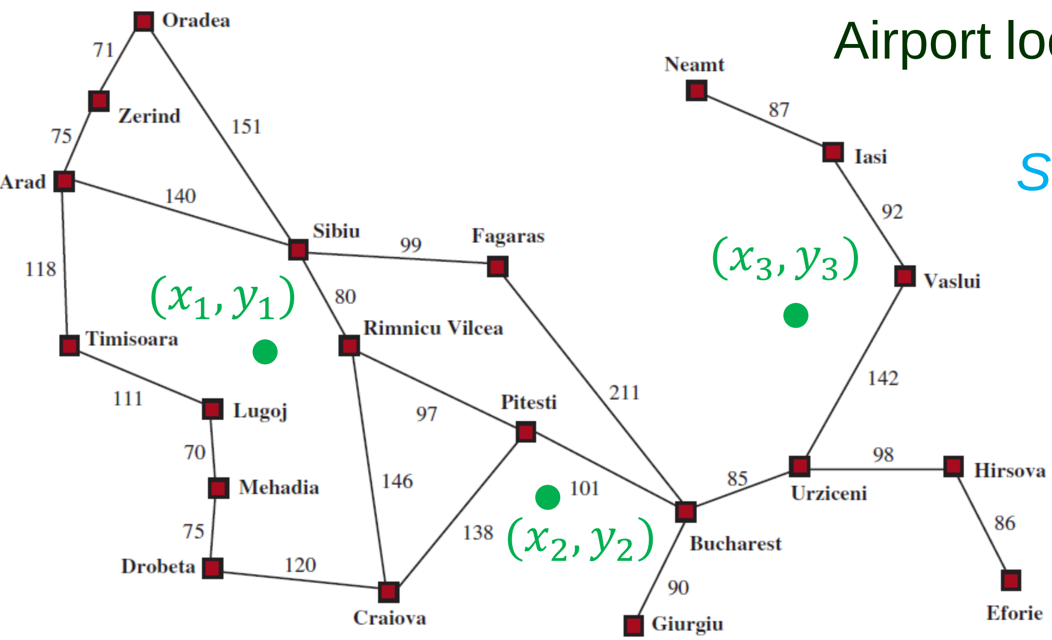


Airport locations: (x_1, y_1), (x_2, y_2), (x_3, y_3)

State $\boldsymbol{x} = (x_1, y_1, x_2, y_2, x_3, y_3)$

I. Local Search in Continuous Space

Problem Place three new airports anywhere in Romania to minimize the sum of square straight-line distances from every city to its nearest airport.



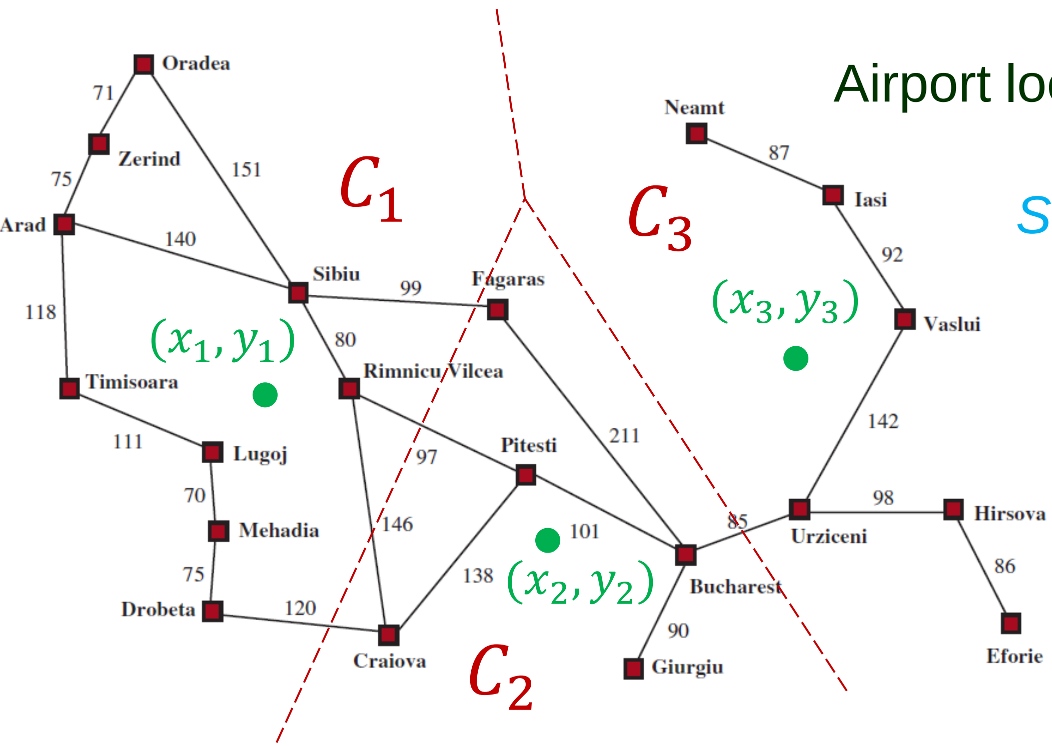
Airport locations: (x_1, y_1), (x_2, y_2), (x_3, y_3)

State $\boldsymbol{x} = (x_1, y_1, x_2, y_2, x_3, y_3)$

C_i: set of cities to which airport i is the closest, for $1 \leq i \leq 3$. Depends on $\boldsymbol{x}$.

I. Local Search in Continuous Space

Problem Place three new airports anywhere in Romania to minimize the sum of square straight-line distances from every city to its nearest airport.



Voronoi diagram

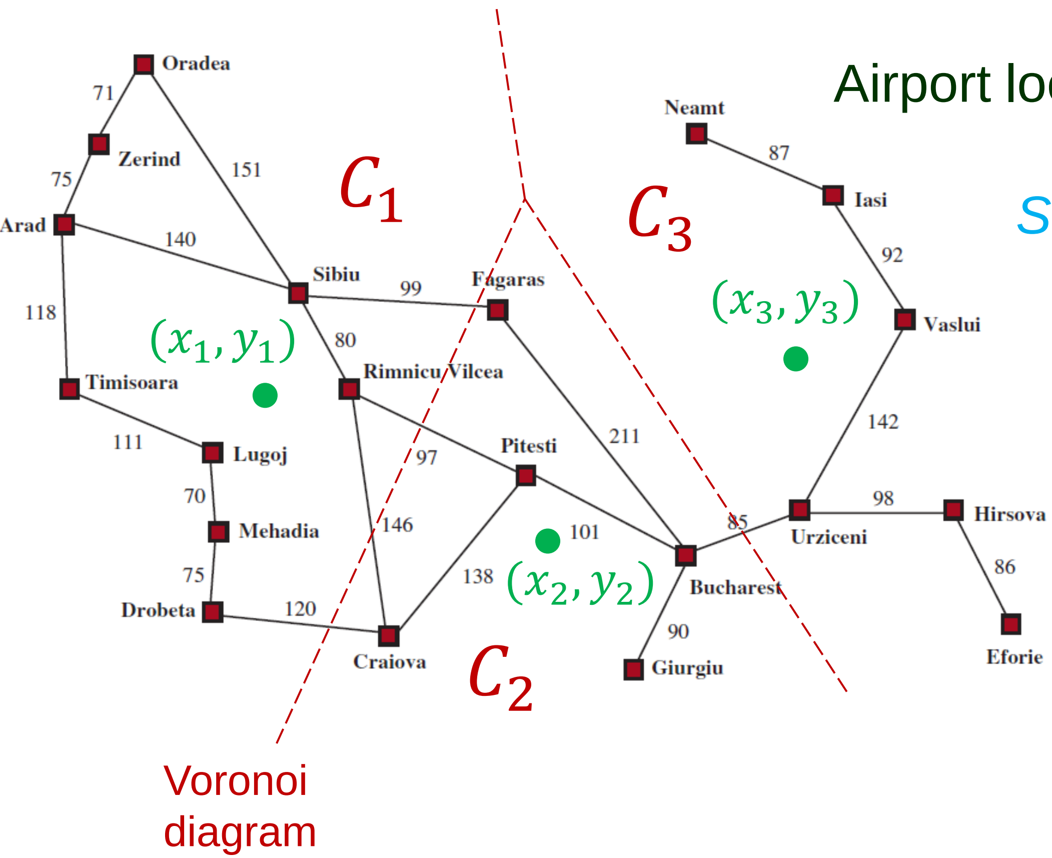
Airport locations: (x_1, y_1), (x_2, y_2), (x_3, y_3)

State $\boldsymbol{x} = (x_1, y_1, x_2, y_2, x_3, y_3)$

C_i: set of cities to which airport i is the closest, for $1 \le i \le 3$. Depends on $\boldsymbol{x}$.

I. Local Search in Continuous Space

Problem Place three new airports anywhere in Romania to minimize the sum of square straight-line distances from every city to its nearest airport.



Airport locations: (x_1, y_1), (x_2, y_2), (x_3, y_3)

State $\boldsymbol{x} = (x_1, y_1, x_2, y_2, x_3, y_3)$

C_i: set of cities to which airport i is the closest, for $1 \le i \le 3$. Depends on $\boldsymbol{x}$.

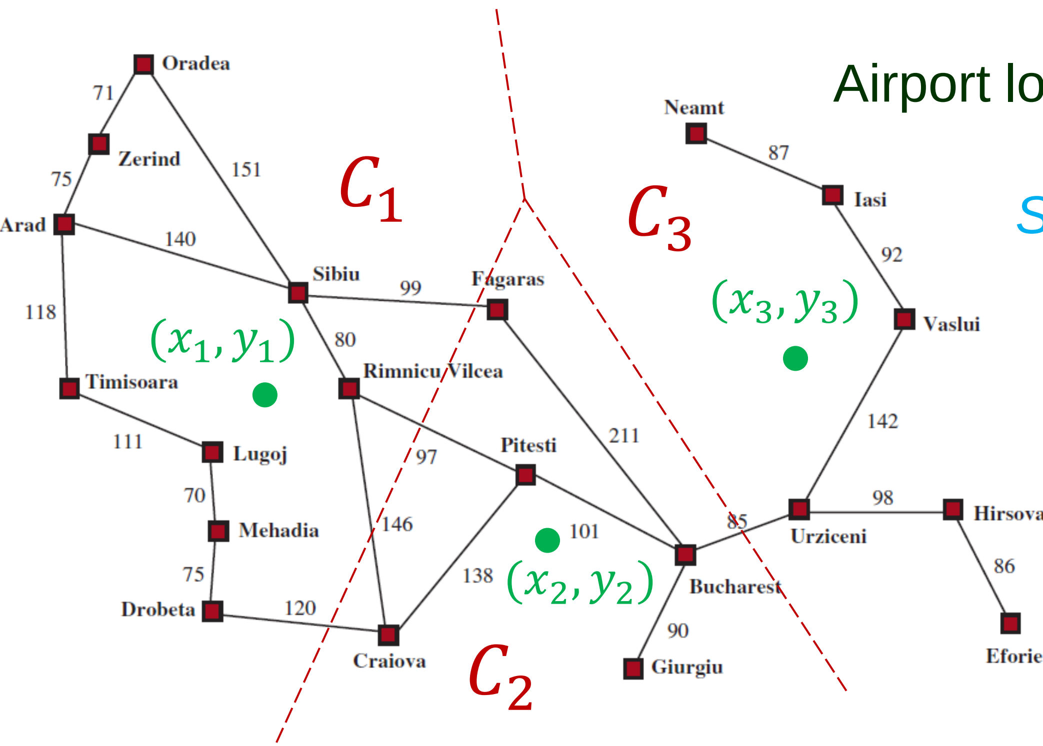
Objective function:

$$f(\boldsymbol{x}) = f(x_1, y_1, x_2, y_2, x_3, y_3)$$

$$= \sum_{i=1}^{3} \sum_{c \in C_i} \left((x_i - x_c)^2 + (y_i - y_c)^2 \right)$$

I. Local Search in Continuous Space

Problem Place three new airports anywhere in Romania to minimize the sum of square straight-line distances from every city to its nearest airport.



Voronoi diagram

Airport locations: (x_1, y_1), (x_2, y_2), (x_3, y_3)

State $\boldsymbol{x} = (x_1, y_1, x_2, y_2, x_3, y_3)$

C_i: set of cities to which airport i is the closest, for $1 \le i \le 3$. Depends on $\boldsymbol{x}$.

Objective function:

$$f(\boldsymbol{x}) = f(x_1, y_1, x_2, y_2, x_3, y_3)$$

$$= \sum_{i=1}^{3} \sum_{c \in C_i} \left((x_i - x_c)^2 + (y_i - y_c)^2 \right)$$

$$\min_{x} f(\boldsymbol{x})$$

Gradient-Based Method

Use the *gradient*: $\nabla f(x_1, \ldots, x_n) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right)$

Gradient-Based Method

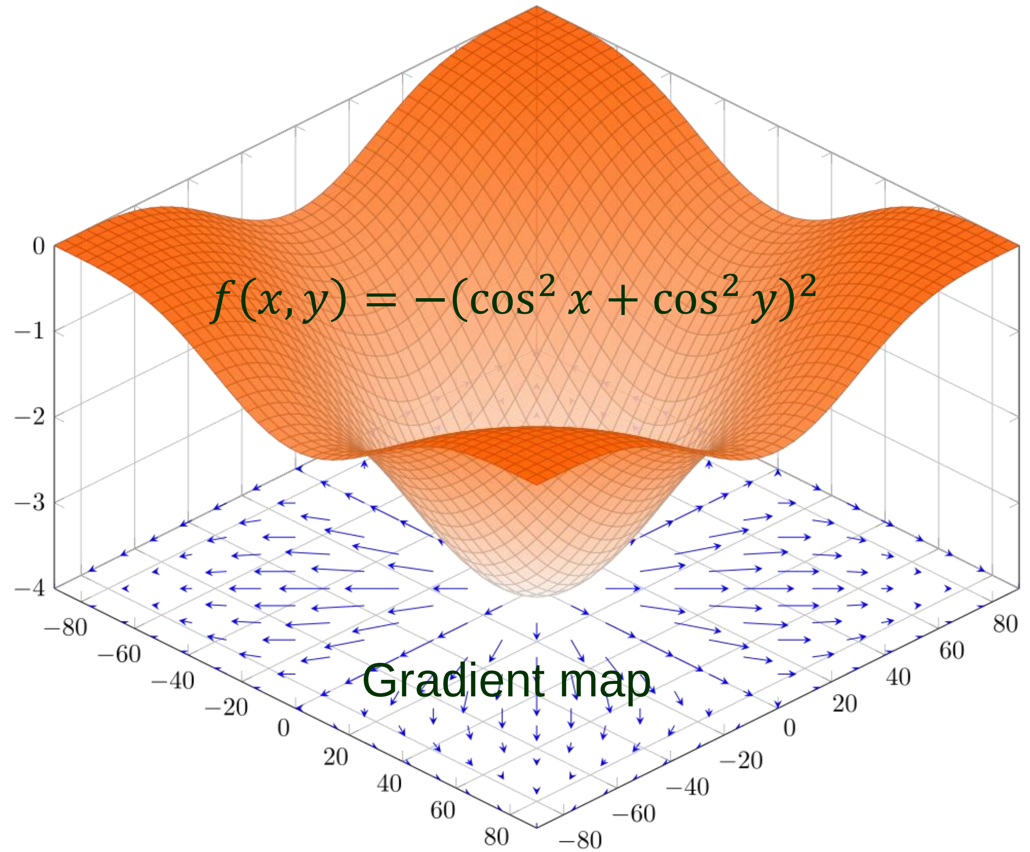
Use the *gradient*: $\nabla f(x_1, \dots, x_n) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right)$

direction of fastest increase in f value locally

Gradient-Based Method

Use the *gradient*: $\nabla f(x_1, \ldots, x_n) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right)$

direction of fastest increase in f value locally



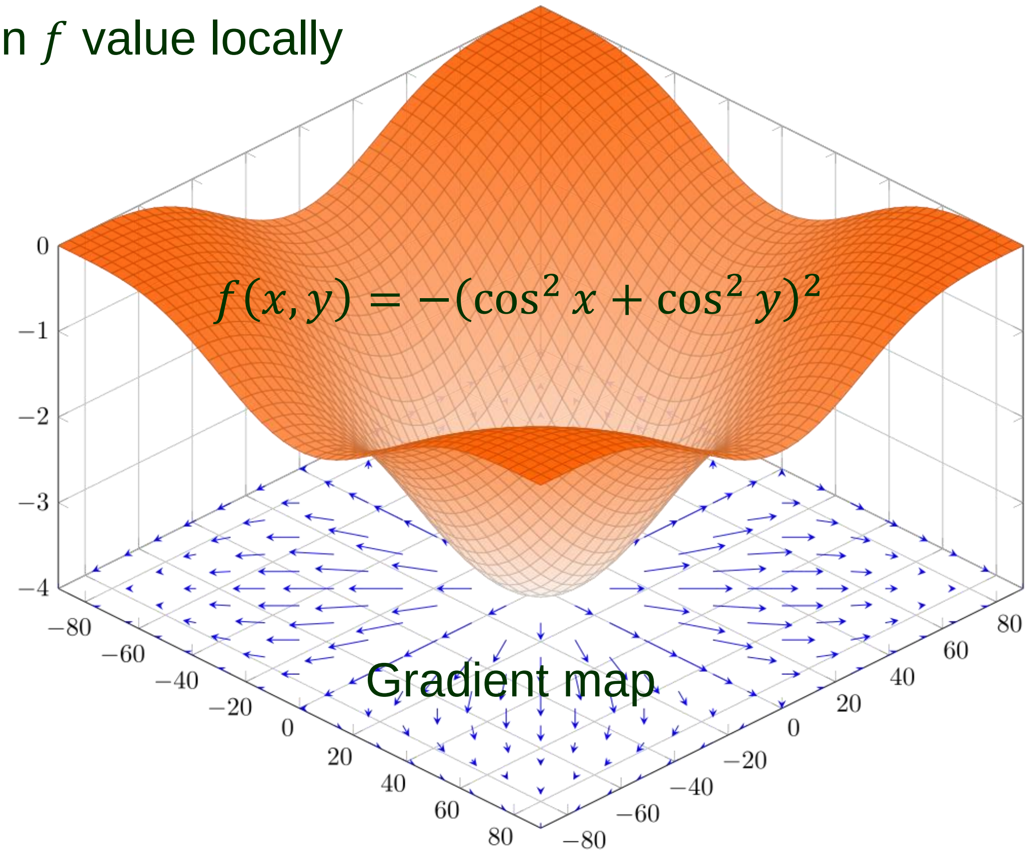
* From https://en.wikipedia.org/wiki/Gradient

Gradient-Based Method

Use the *gradient*: $\nabla f(x_1, \dots, x_n) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n}\right)$

$\uparrow$ direction of fastest increase in f value locally

$-\nabla f$: direction of fastest decrease in f value locally



* From https://en.wikipedia.org/wiki/Gradient

Gradient Descent

Back to airport placements:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)$$

$$f(\boldsymbol{x}) = \sum_{i=1}^{3} \sum_{c \in C_i(\boldsymbol{x})} \left((x_i - x_c)^2 + (y_i - y_c)^2 \right)$$

Gradient Descent

Back to airport placements:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)$$

$$f(\boldsymbol{x}) = \sum_{i=1}^{3} \sum_{c \in C_i(\boldsymbol{x})} \left((x_i - x_c)^2 + (y_i - y_c)^2 \right) \Longrightarrow \quad \frac{\partial f}{\partial x_1} = 2 \sum_{c \in C_1} (x_1 - x_c)$$

Gradient Descent

Back to airport placements:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3}\right)$$

$$f(\boldsymbol{x}) = \sum_{i=1}^{3} \sum_{c \in C_i(\boldsymbol{x})} \left((x_i - x_c)^2 + (y_i - y_c)^2\right) \Longrightarrow \quad \frac{\partial f}{\partial x_1} = 2 \sum_{c \in C_1} (x_1 - x_c)$$

Update: $\boldsymbol{x} \leftarrow \boldsymbol{x} - \alpha \nabla f(\boldsymbol{x})$

Gradient Descent

Back to airport placements:

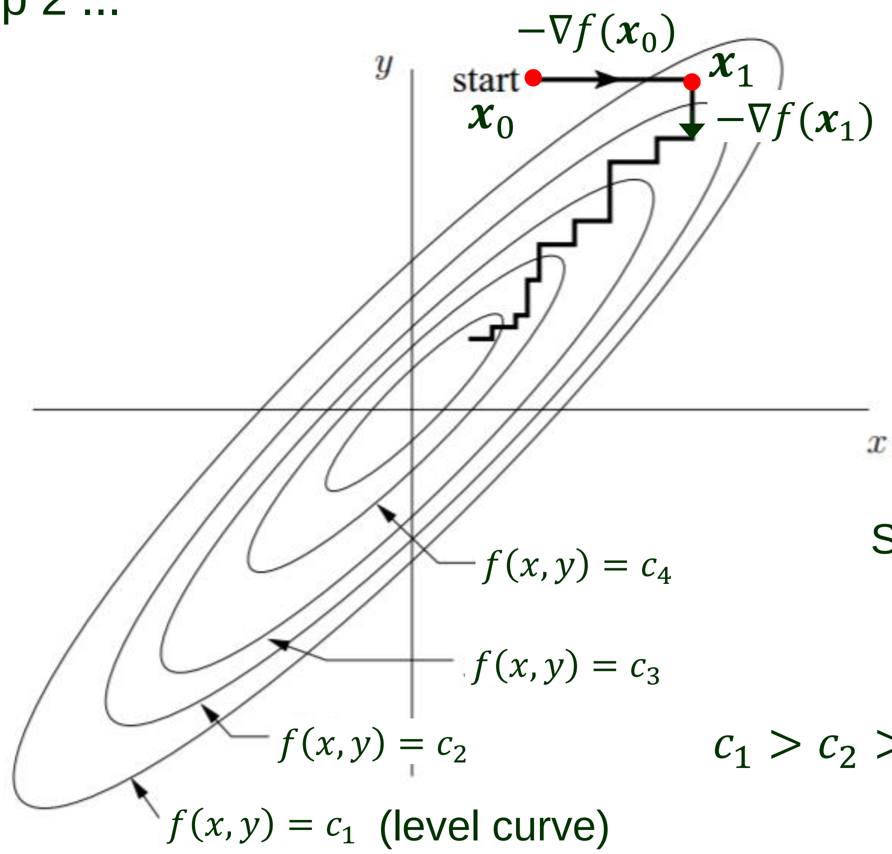
$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)$$

$$f(\boldsymbol{x}) = \sum_{i=1}^{3} \sum_{c \in C_i(\boldsymbol{x})} \left((x_i - x_c)^2 + (y_i - y_c)^2 \right) \Longrightarrow \quad \frac{\partial f}{\partial x_1} = 2 \sum_{c \in C_1} (x_1 - x_c)$$

Update: $\boldsymbol{x} \leftarrow \boldsymbol{x} - \underset{\substack{\uparrow \\ \text{step size}}}{\alpha} \nabla f(\boldsymbol{x})$

Line Search

1. Start at an initial state $\boldsymbol{x} = \boldsymbol{x}_0$.
2. Move along $-\nabla f(\boldsymbol{x})$ until f no longer decreases.
3. $\boldsymbol{x} \leftarrow$ new stopping point.
4. Go back to step 2 ...



Steepest descent (along $-\nabla f$)

$c_1 > c_2 > c_3 > c_4$

Newton-Raphson Method

Solve

$$f(x) = 0 \qquad \text{// one variable}$$

with the iteration formula

$$x \leftarrow x - \frac{f(x)}{f'(x)}$$

Newton-Raphson Method

Solve

$$f(x) = 0 \qquad \text{// one variable}$$

with the iteration formula

$$x \leftarrow x - \frac{f(x)}{f'(x)}$$

To maximize/minimize $f(\boldsymbol{x})$, find $\boldsymbol{x}$ at which $\nabla f(\boldsymbol{x}) = \boldsymbol{0}$.

Newton-Raphson Method

Solve

$$f(x) = 0 \qquad \text{// one variable}$$

with the iteration formula

$$x \leftarrow x - \frac{f(x)}{f'(x)}$$

To maximize/minimize $f(\boldsymbol{x})$, find $\boldsymbol{x}$ at which $\nabla f(\boldsymbol{x}) = \boldsymbol{0}$.

Iteration formula:

$$\boldsymbol{x} \leftarrow \boldsymbol{x} - H_f^{-1}(\boldsymbol{x})(\nabla f(\boldsymbol{x}))^T$$

Newton-Raphson Method

Solve

$$f(x) = 0$$ // one variable

with the iteration formula

$$x \leftarrow x - \frac{f(x)}{f'(x)}$$

To maximize/minimize $f(\boldsymbol{x})$, find $\boldsymbol{x}$ at which $\nabla f(\boldsymbol{x}) = \boldsymbol{0}$.

Iteration formula:

$$\boldsymbol{x} \leftarrow \boldsymbol{x} - H_f^{-1}(\boldsymbol{x})(\nabla f(\boldsymbol{x}))^T$$

Hessian of f: matrix $\left(\frac{\partial^2 f}{\partial x_i \partial x_j}\right)$

More on Continuous Optimization

Com S 477/577 covers more topics:

- Unconstrained nonlinear optimization:

  http://web.cs.iastate.edu/~cs577/handouts/nonlinear-program.pdf

More on Continuous Optimization

Com S 477/577 covers more topics:

- Unconstrained nonlinear optimization:

  http://web.cs.iastate.edu/~cs577/handouts/nonlinear-program.pdf

- Constrained nonlinear optimization (Lagrange multipliers):

  http://web.cs.iastate.edu/~cs577/handouts/lagrange-multiplier.pdf

More on Continuous Optimization

Com S 477/577 covers more topics:

- Unconstrained nonlinear optimization:

  http://web.cs.iastate.edu/~cs577/handouts/nonlinear-program.pdf

- Constrained nonlinear optimization (Lagrange multipliers):

  http://web.cs.iastate.edu/~cs577/handouts/lagrange-multiplier.pdf

- Linear programming (i.e., linear optimization under linear constraints)

  http://web.cs.iastate.edu/~cs577/handouts/linear-program.pdf

  http://web.cs.iastate.edu/~cs577/handouts/simplex.pdf

More on Continuous Optimization

Com S 477/577 covers more topics:

- Unconstrained nonlinear optimization:

  http://web.cs.iastate.edu/~cs577/handouts/nonlinear-program.pdf

- Constrained nonlinear optimization (Lagrange multipliers):

  http://web.cs.iastate.edu/~cs577/handouts/lagrange-multiplier.pdf

- Linear programming (i.e., linear optimization under linear constraints)

  http://web.cs.iastate.edu/~cs577/handouts/linear-program.pdf

  http://web.cs.iastate.edu/~cs577/handouts/simplex.pdf

II. Nondeterministic Actions

- The agent may not know the current state in a partially observable environment.
- The agent may not know the next state after taking an action in a nondeterministic environment.

II. Nondeterministic Actions

- The agent may not know the current state in a partially observable environment.
- The agent may not know the next state after taking an action in a nondeterministic environment.

A *belief state* is a set of physical states believed to be possible by the agent.

II. Nondeterministic Actions

- The agent may not know the current state in a partially observable environment.
- The agent may not know the next state after taking an action in a nondeterministic environment.

A *belief state* is a set of physical states believed to be possible by the agent.

Problem solution: a *conditional plan*.

II. Nondeterministic Actions

- The agent may not know the current state in a partially observable environment.
- The agent may not know the next state after taking an action in a nondeterministic environment.

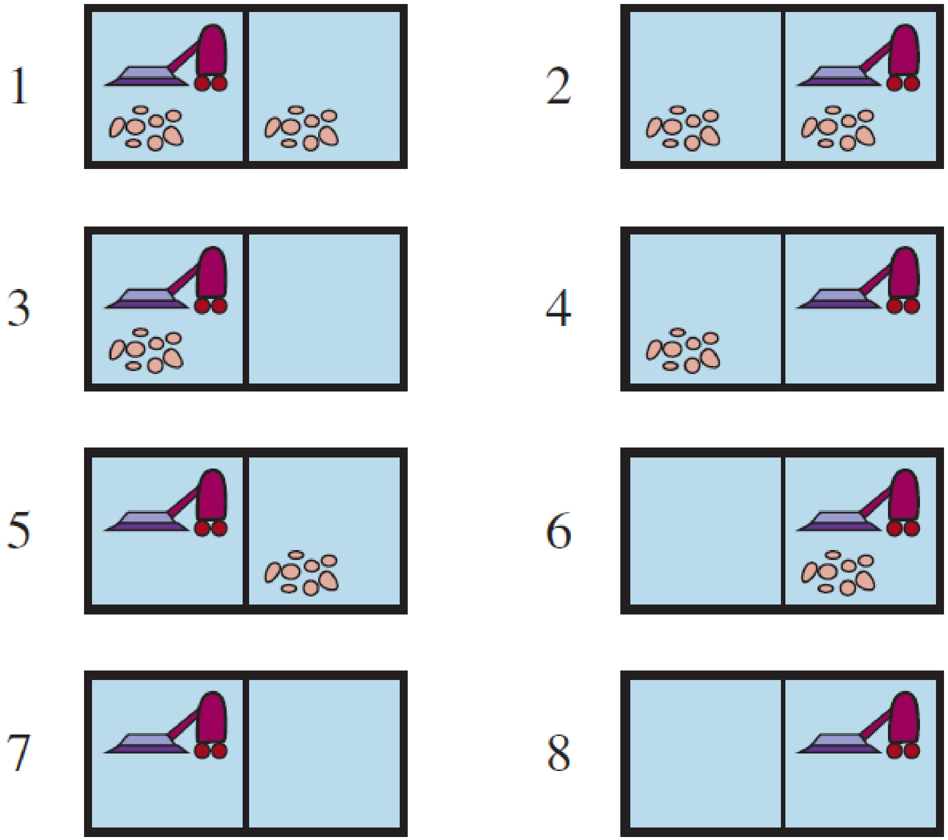
A *belief state* is a set of physical states believed to be possible by the agent.

Problem solution: a *conditional plan*.

To specify what to do depending on what percepts the agent receives while executing the plan.

Perfect Vacuum World

Fully observable, deterministic, and completely known

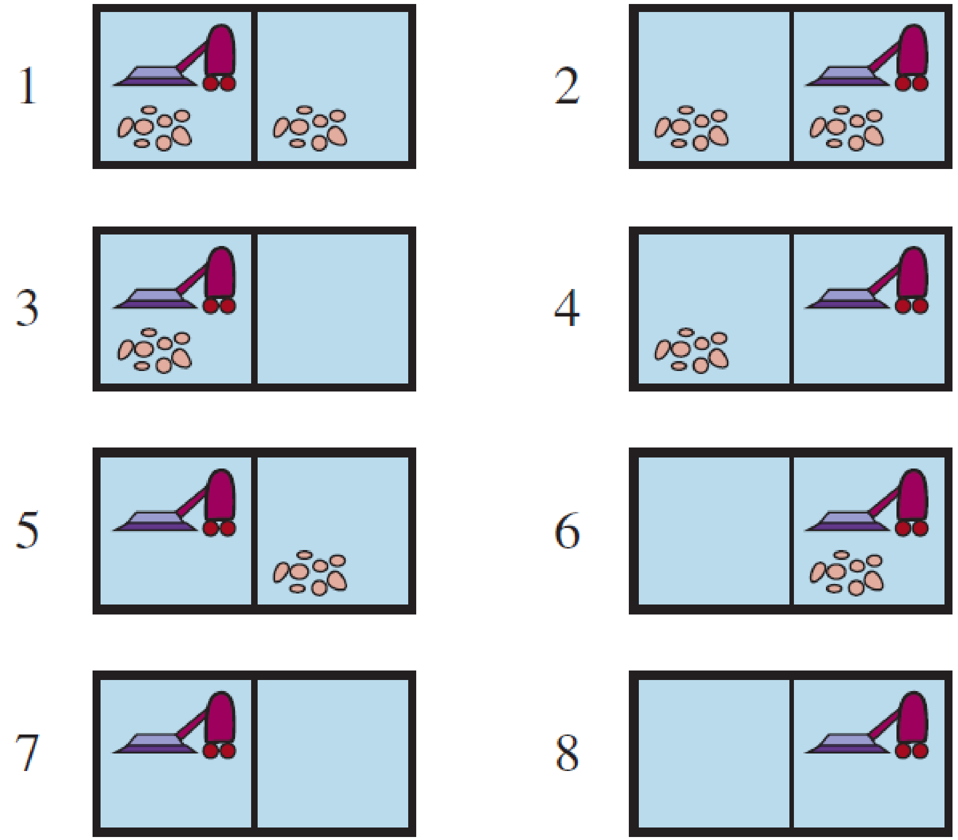


8 possible states

Perfect Vacuum World

Fully observable, deterministic, and completely known

Solution: *Suck, Right, Suck* ⇐

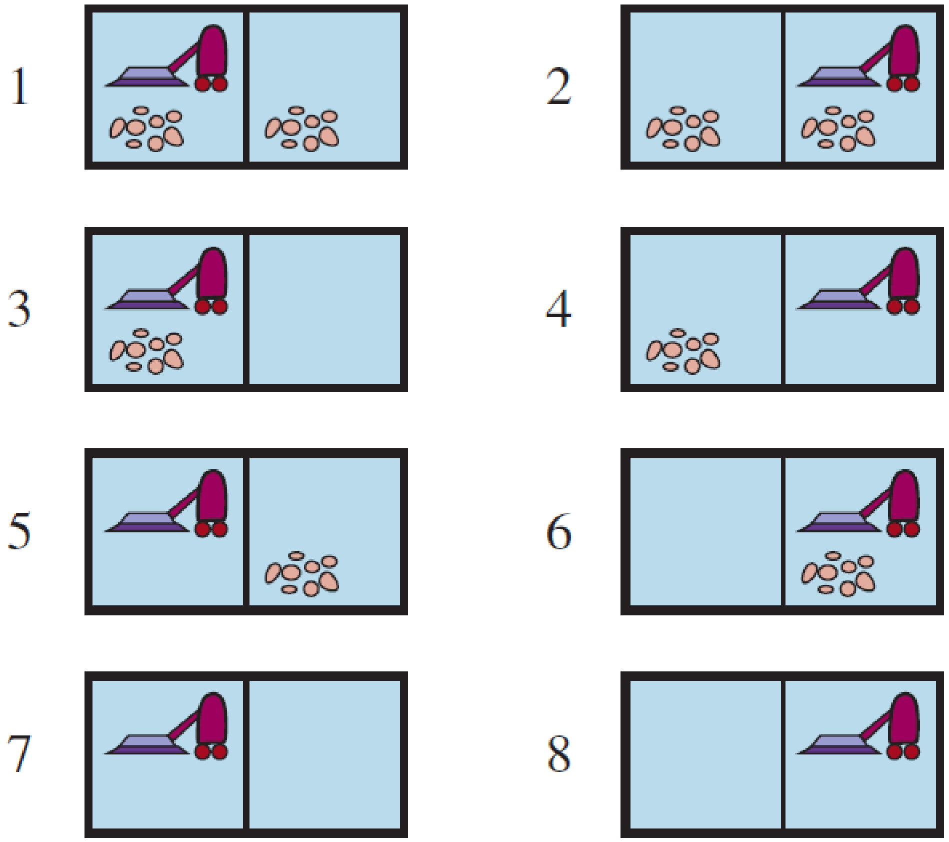


8 possible states

Erratic Vacuum World

Suck:

- Applied to a dirty square
  - clean the square
  - sometimes clean up dirt in an adjacent square

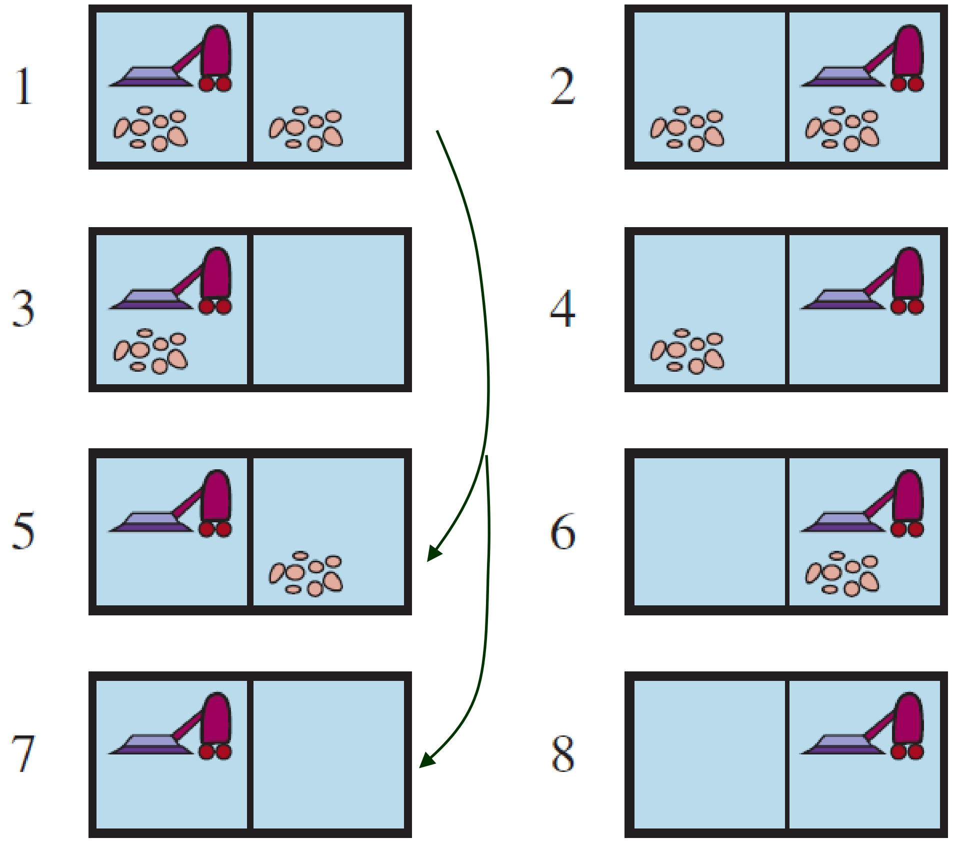


Erratic Vacuum World

Suck:

- Applied to a dirty square
  - clean the square
  - sometimes clean up dirt in an adjacent square

Either state 5 or 7 after applied to state 1:



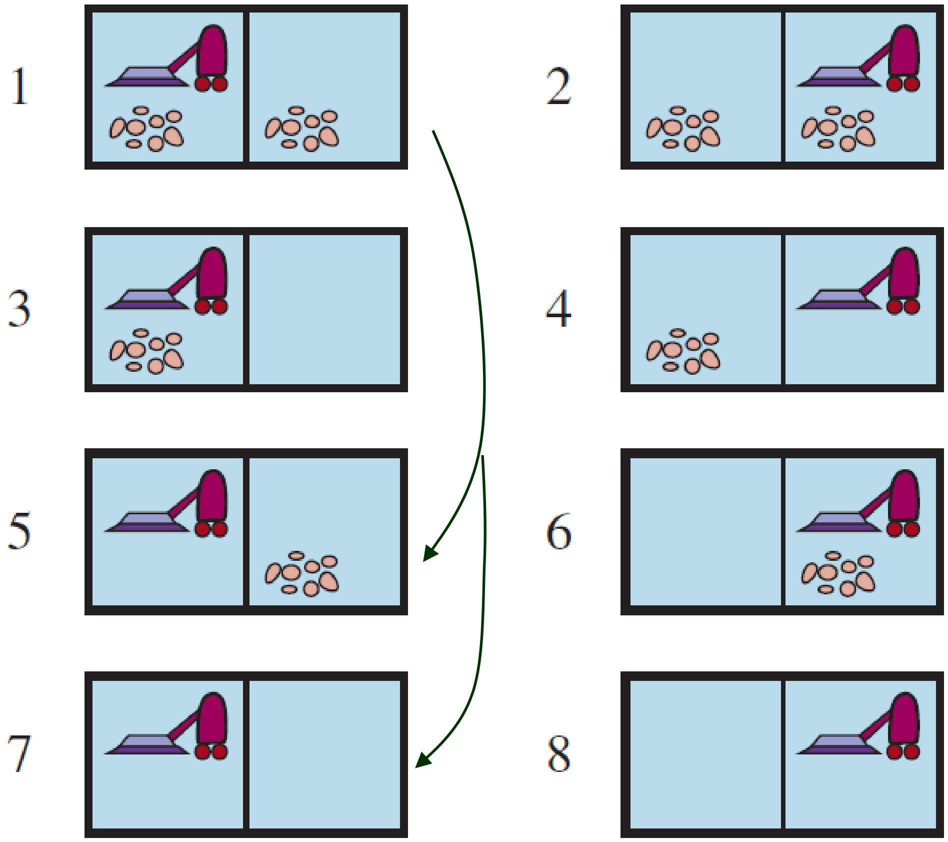
Erratic Vacuum World

Suck:

- Applied to a dirty square
  - clean the square
  - sometimes clean up dirt in an adjacent square

Either state 5 or 7 after applied to state 1:

RESULTS(1, *Suck*) = {5, 7}



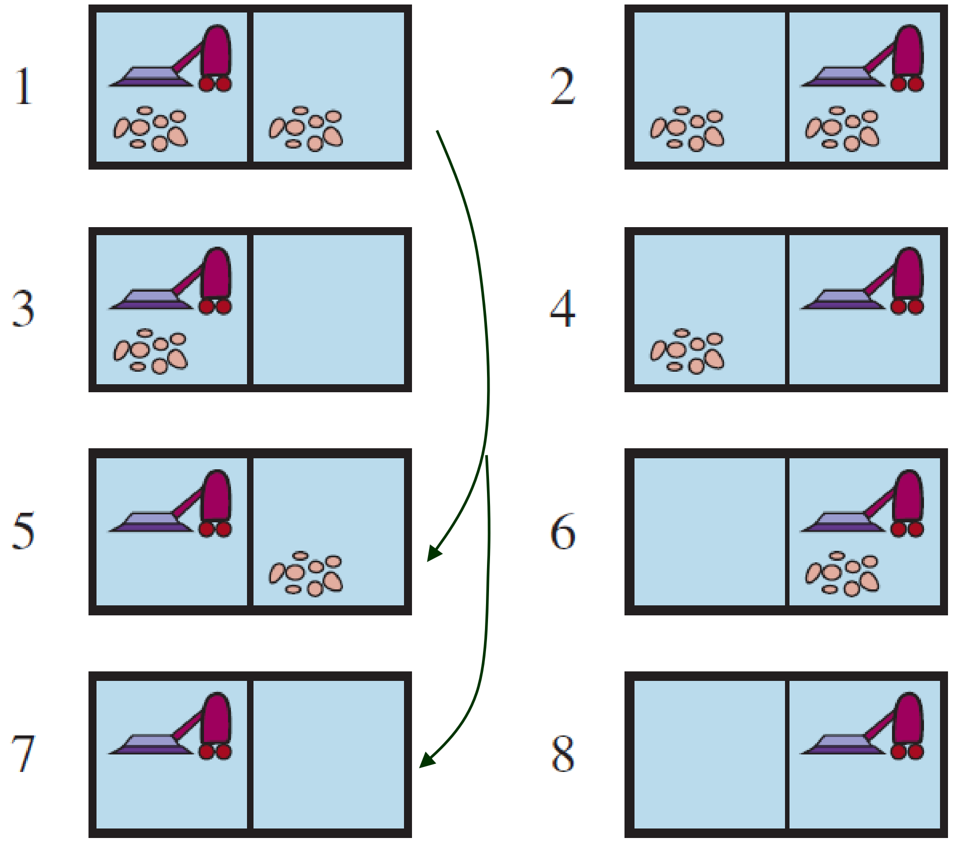
Erratic Vacuum World

Suck:

- Applied to a dirty square
  - clean the square
  - sometimes clean up dirt in an adjacent square
- Applied to a clean square
  - sometimes deposit dirt on the carpet

Either state 5 or 7 after applied to state 1:

RESULTS(1, *Suck*) = {5, 7}



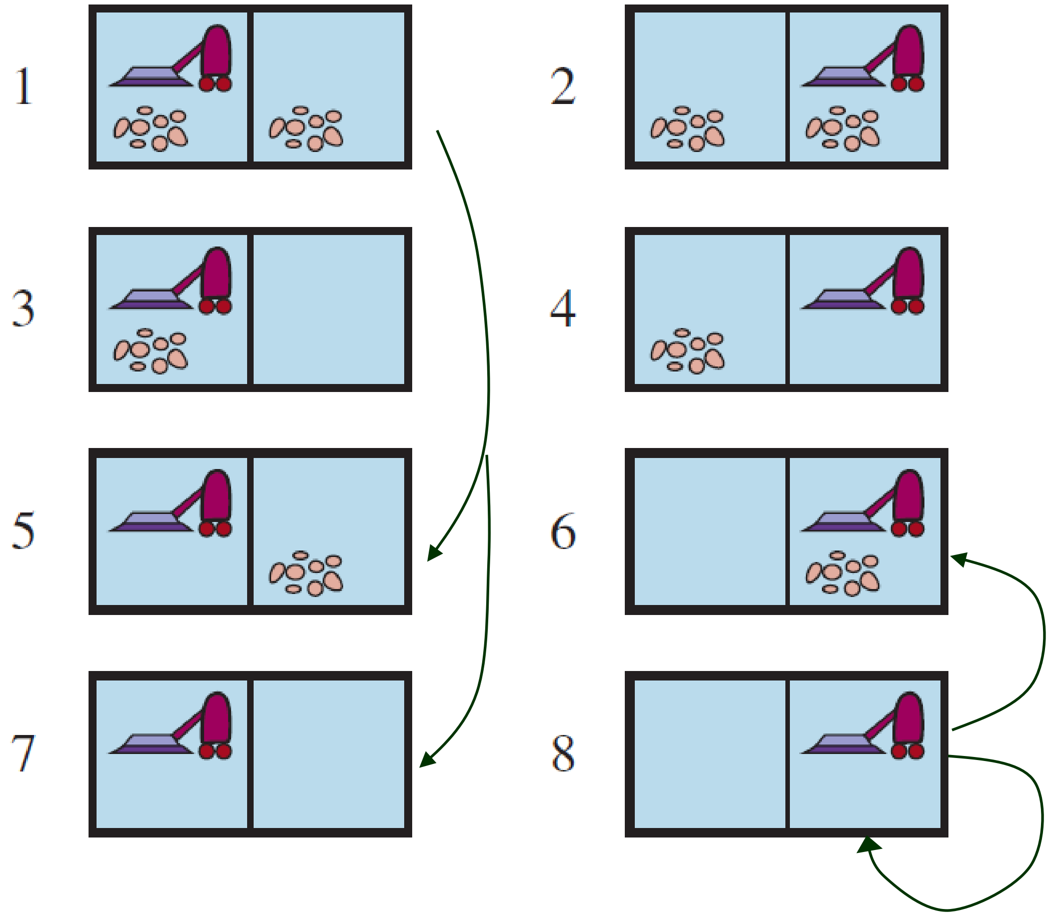
Erratic Vacuum World

Suck:

- Applied to a dirty square
  - clean the square
  - sometimes clean up dirt in an adjacent square
- Applied to a clean square
  - sometimes deposit dirt on the carpet

Either state 5 or 7 after applied to state 1:

RESULTS(1, *Suck*) = {5, 7}

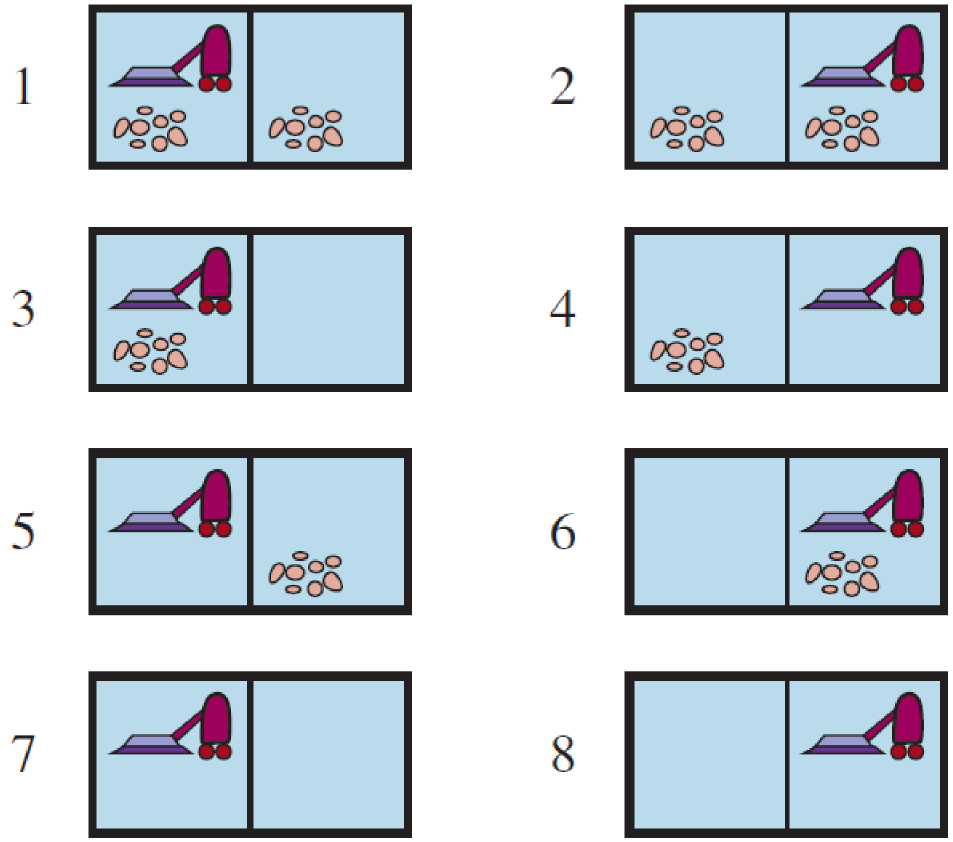


Still Solvable?

Suck:

- Applied to a dirty square
  - clean the square
  - sometimes clean up dirt in an adjacent square

No single sequence of actions solves the problem.



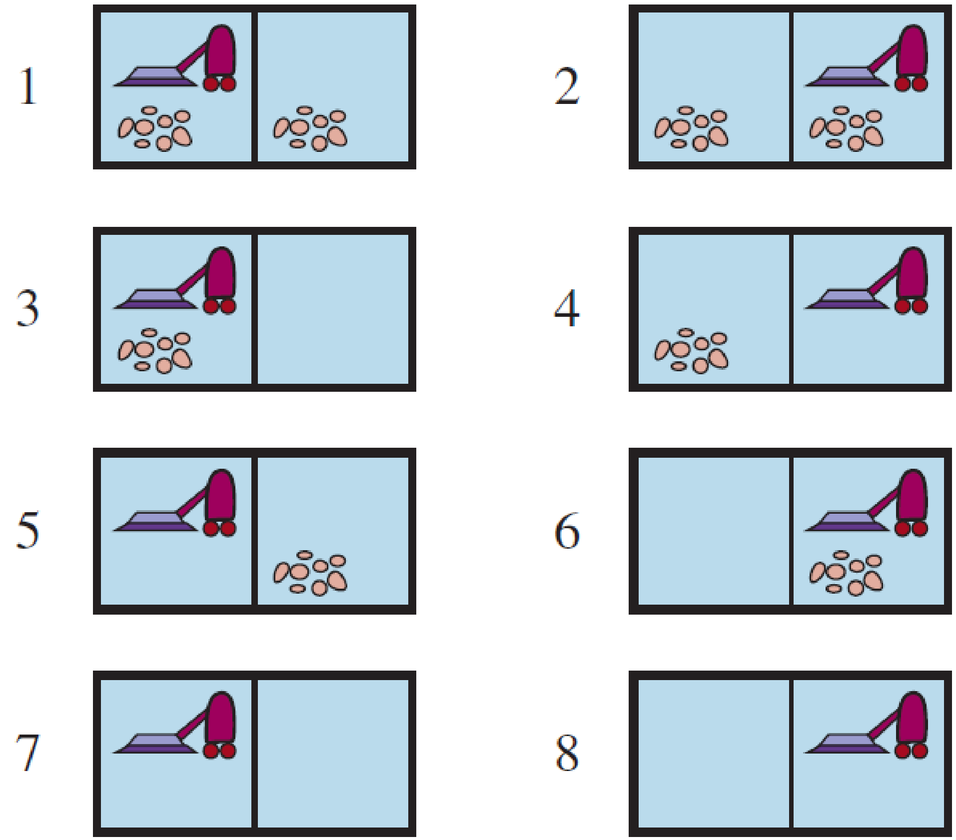
Still Solvable?

Suck:
- Applied to a dirty square
  - clean the square
  - sometimes clean up dirt in an adjacent square

No single sequence of actions solves the problem.

Solution still exists as a *conditional plan* (suppose at State 1):

[*Suck*, **if** State = 5 **then**
[*Right, Suck*] **else** []]



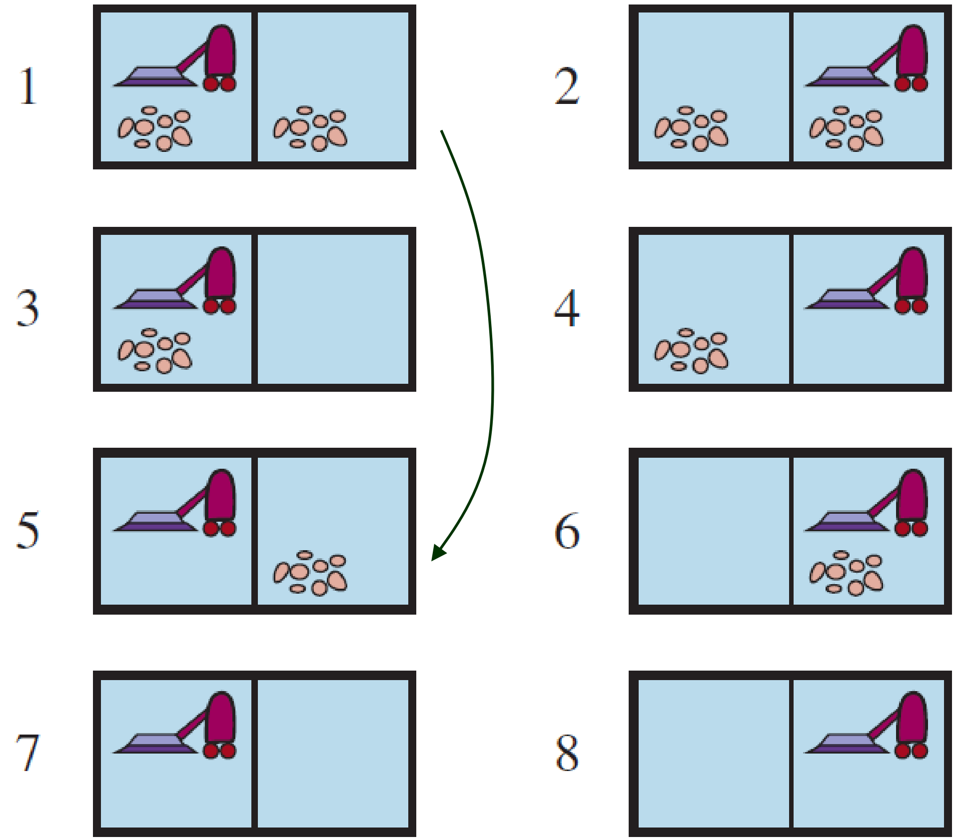
Still Solvable?

Suck:
- Applied to a dirty square
  - clean the square
  - sometimes clean up dirt in an adjacent square

No single sequence of actions solves the problem.

Solution still exists as a *conditional plan* (suppose at State 1):

[*Suck*, **if** State = 5 **then**
[*Right*, *Suck*] **else** []]



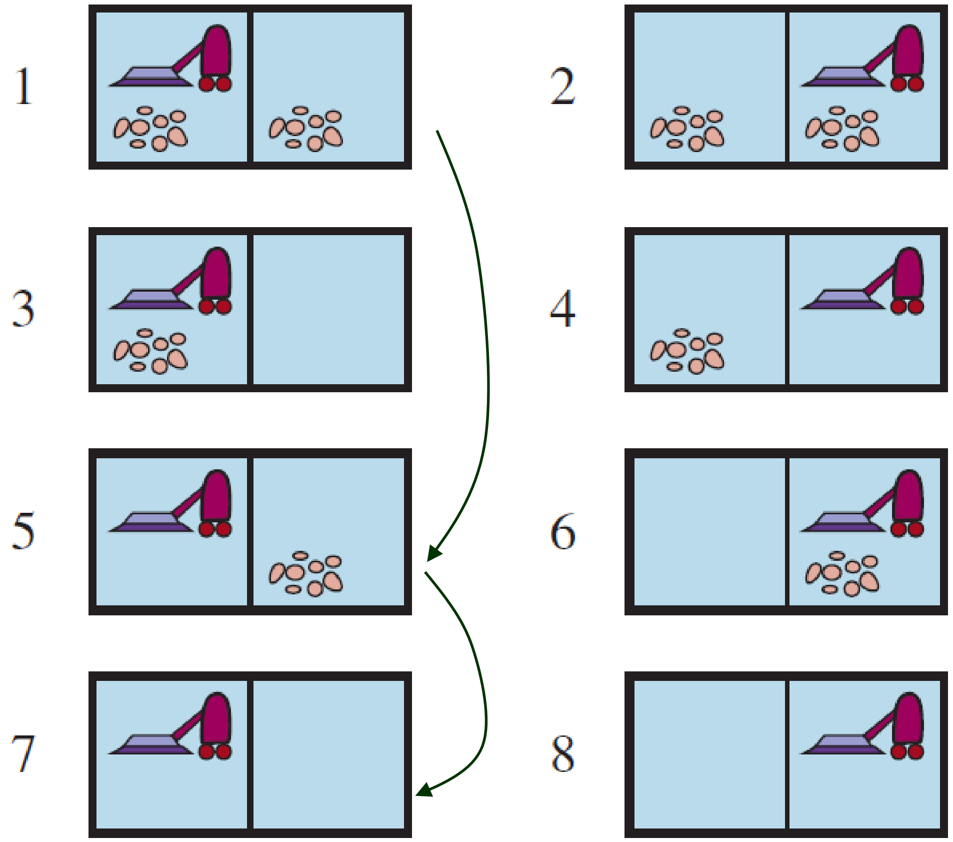
Still Solvable?

Suck:
- Applied to a dirty square
  - clean the square
  - sometimes clean up dirt in an adjacent square

No single sequence of actions solves the problem.

Solution still exists as a *conditional plan* (suppose at State 1):

[*Suck*, **if** State = 5 **then**
[*Right, Suck*] **else** []]



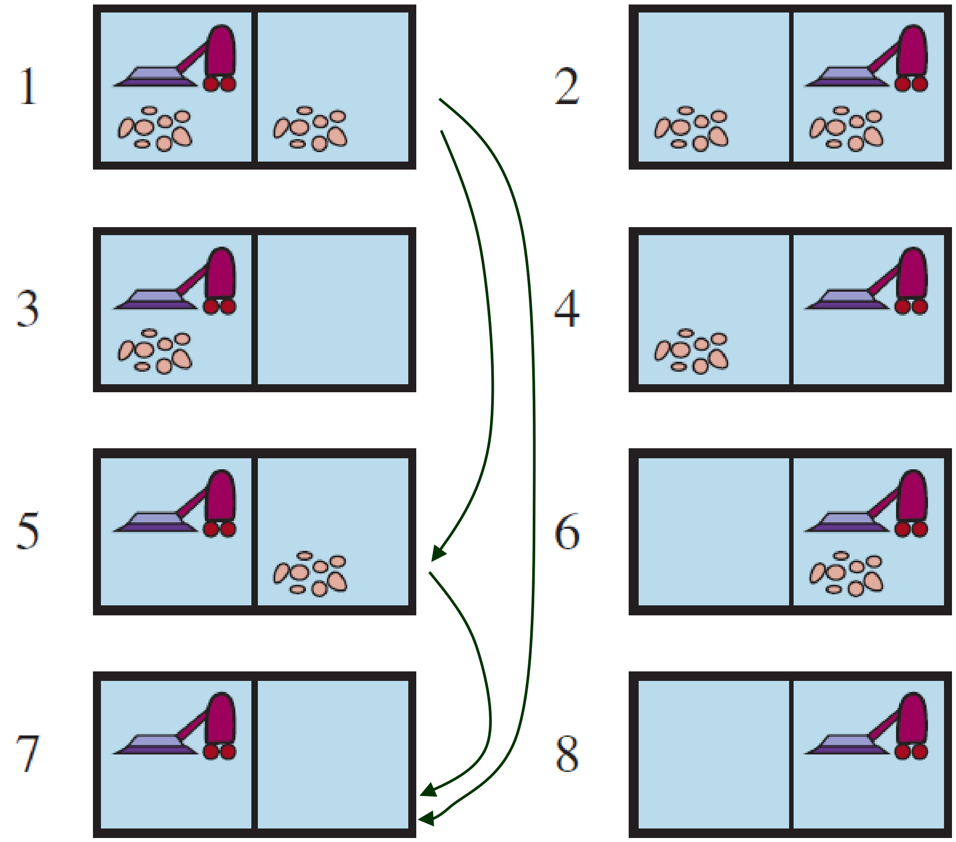
Still Solvable?

Suck:
- Applied to a dirty square
  - clean the square
  - sometimes clean up dirt in an adjacent square

No single sequence of actions solves the problem.

Solution still exists as a *conditional plan* (suppose at State 1):

[*Suck*, **if** State = 5 **then**
[*Right, Suck*] **else** []]



Still Solvable?

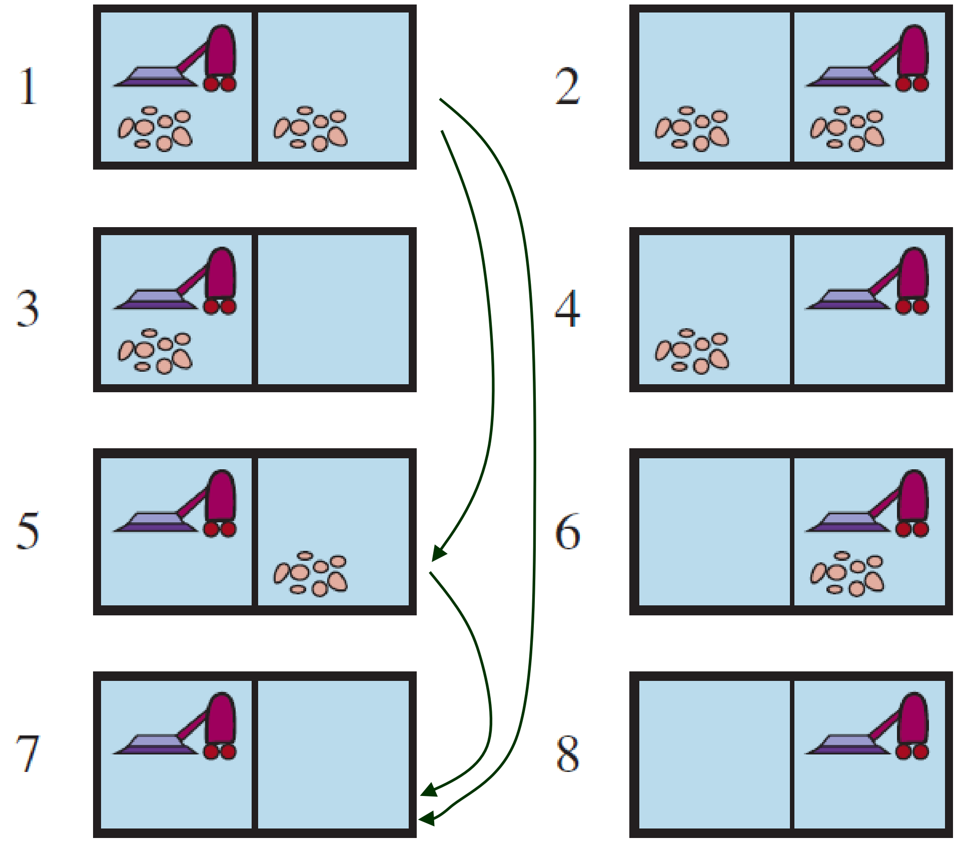
Suck:

- Applied to a dirty square
  - clean the square
  - sometimes clean up dirt in an adjacent square

No single sequence of actions solves the problem.

Solution still exists as a *conditional plan* (suppose at State 1):

[*Suck*, **if** State = 5 **then**
[*Right, Suck*] **else** []]



Solution is a tree – of a different character!

AND-OR Search Tree

OR-node (*deterministic*): the agent chooses an action.

e.g., *Left*, *Right*, or *Suck*

AND-OR Search Tree

OR-node (*deterministic*): the agent chooses an action.

e.g., *Left*, *Right*, or *Suck*

AND-node (*non-deterministic*): the environment "chooses" to have an outcome for each action.

e.g., *Suck* in state 1 results in the belief state {5, 7}.

AND-OR Search Tree

OR-node (*deterministic*): the agent chooses an action.

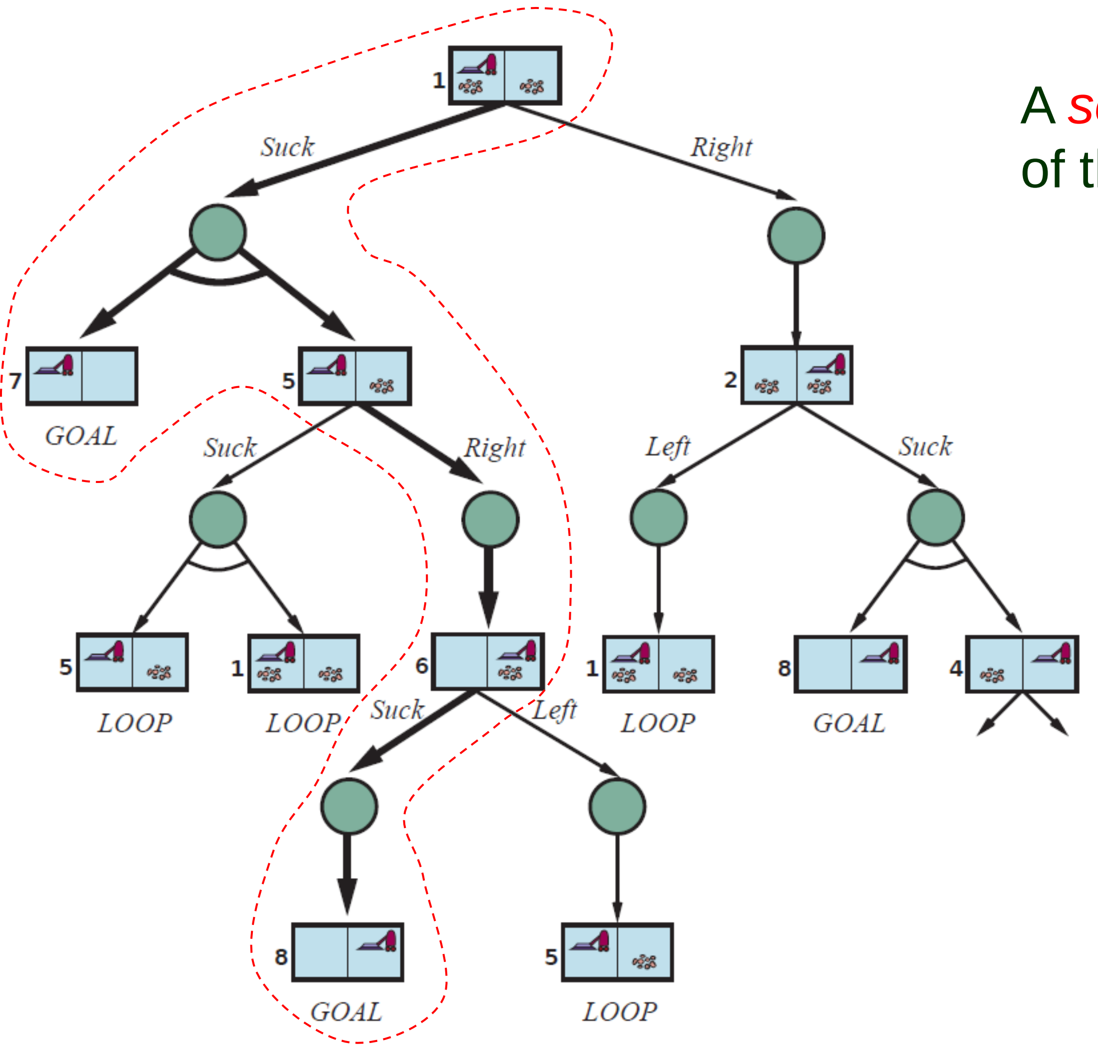
e.g., *Left*, *Right*, or *Suck*

AND-node (*non-deterministic*): the environment "chooses" to have an outcome for each action.

e.g., *Suck* in state 1 results in the belief state {5, 7}.

OR- and AND-nodes alternate in the tree.

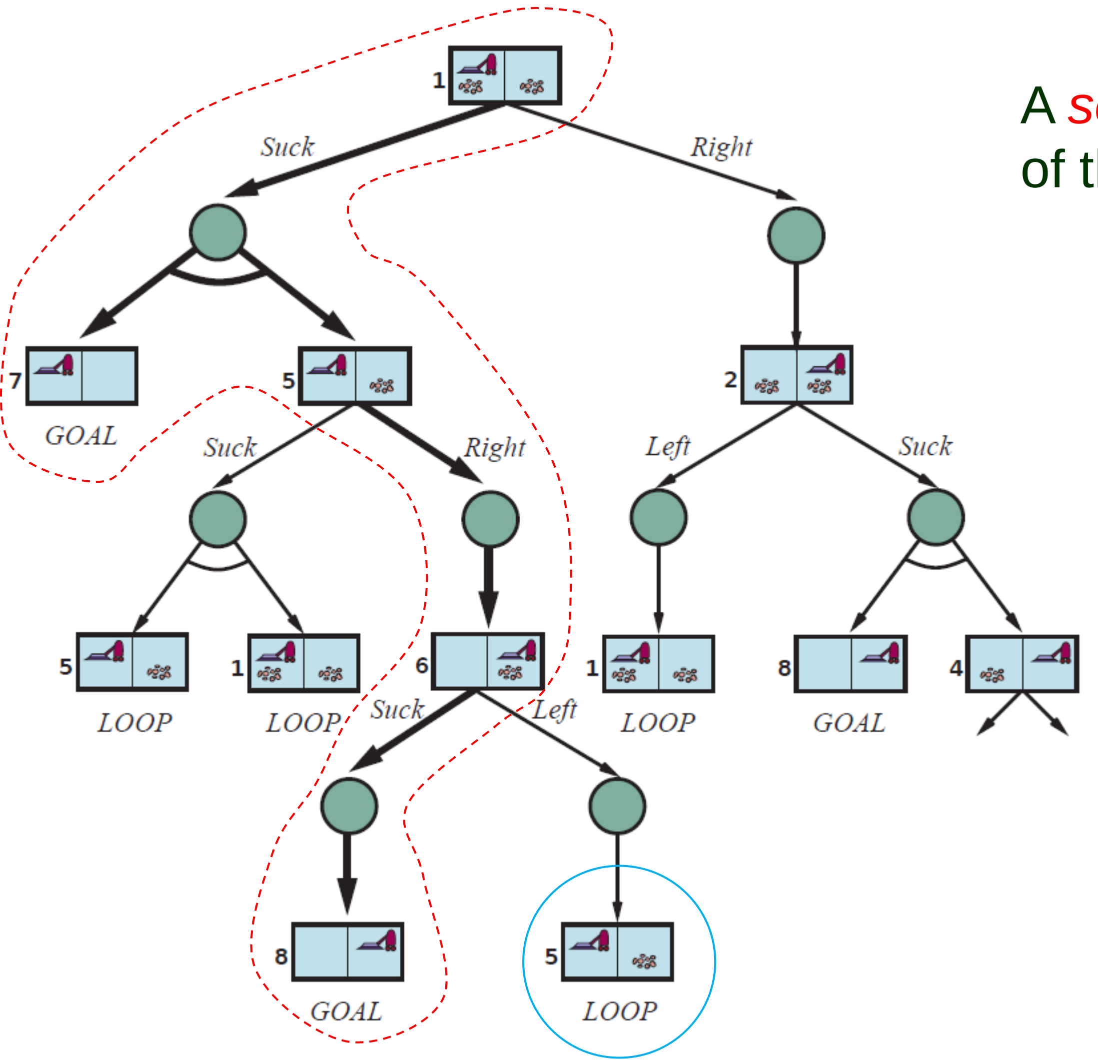
Example of a (Partial) Tree



A *solution* is a connected portion of the AND-OR tree such that

- its root is the tree's root;
- every OR node has exactly one child (i.e., one of the actions);
- every AND node has all children (possible outcomes) from the corresponding action;
- all the leaves are goal nodes.

Example of a (Partial) Tree



A *solution* is a connected portion of the AND-OR tree such that

- its root is the tree's root;
- every OR node has exactly one child (i.e., one of the actions);
- every AND node has all children (possible outcomes) from the corresponding action;
- all the leaves are goal nodes.

DFS Implementation of AND-OR Tree Search

```
function AND-OR-SEARCH(problem) returns a conditional plan, or failure
   return OR-SEARCH(problem, problem.INITIAL, [])

function OR-SEARCH(problem, state, path) returns a conditional plan, or failure
   if problem.IS-GOAL(state) then return the empty plan
   if IS-CYCLE(path) then return failure  // ignore a solution with a cycle. such a solution would
   for each action in problem.ACTIONS(state) do  // imply the existence of a non-cyclic solution
       plan ← AND-SEARCH(problem, RESULTS(state, action), [state] + path])  // which can
       if plan ≠ failure then return [action] + plan]  // be found.
   return failure

function AND-SEARCH(problem, states, path) returns a conditional plan, or failure
   for each s_i in states do
       plan_i ← OR-SEARCH(problem, s_i, path)
       if plan_i = failure then return failure
   return [if s_1 then plan_1 else if s_2 then plan_2 else ... if s_{n-1} then plan_{n-1} else plan_n]
```

Solution plan

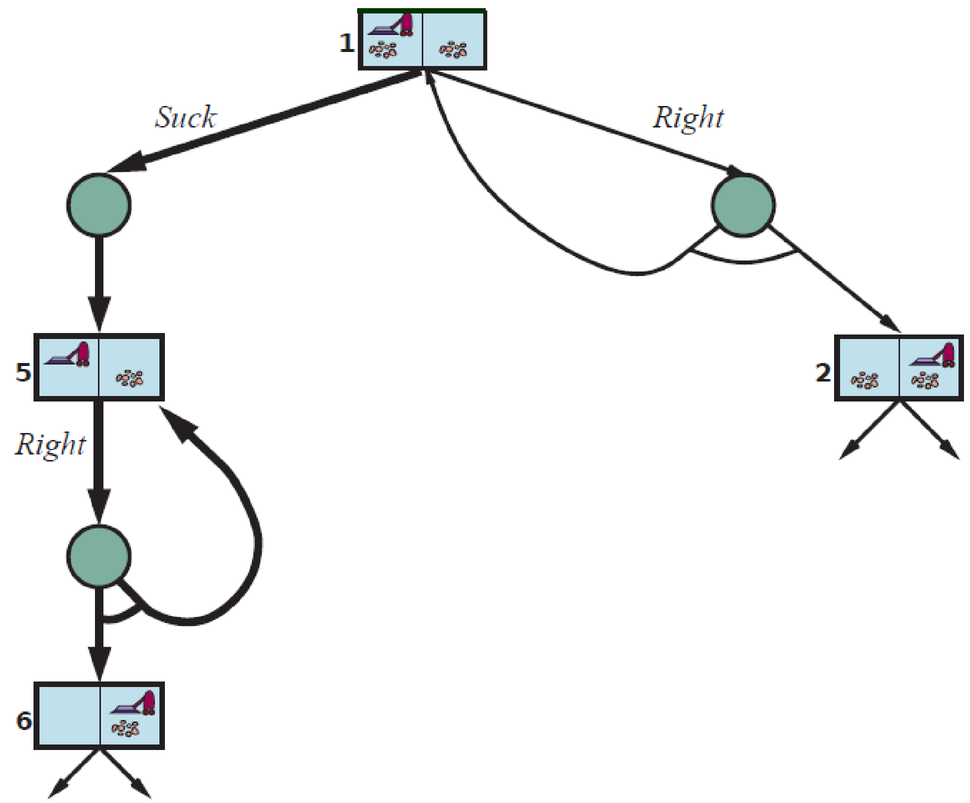
Cyclic Plan for a Solution

What if an action fails and the state is not changed?

Slippery vacuum world.

$$\text{State 1} \xrightarrow{\textit{Right}} \{1, 2\}$$

$$\text{State 5} \xrightarrow{\textit{Right}} \{5, 6\}$$



Cyclic Plan for a Solution

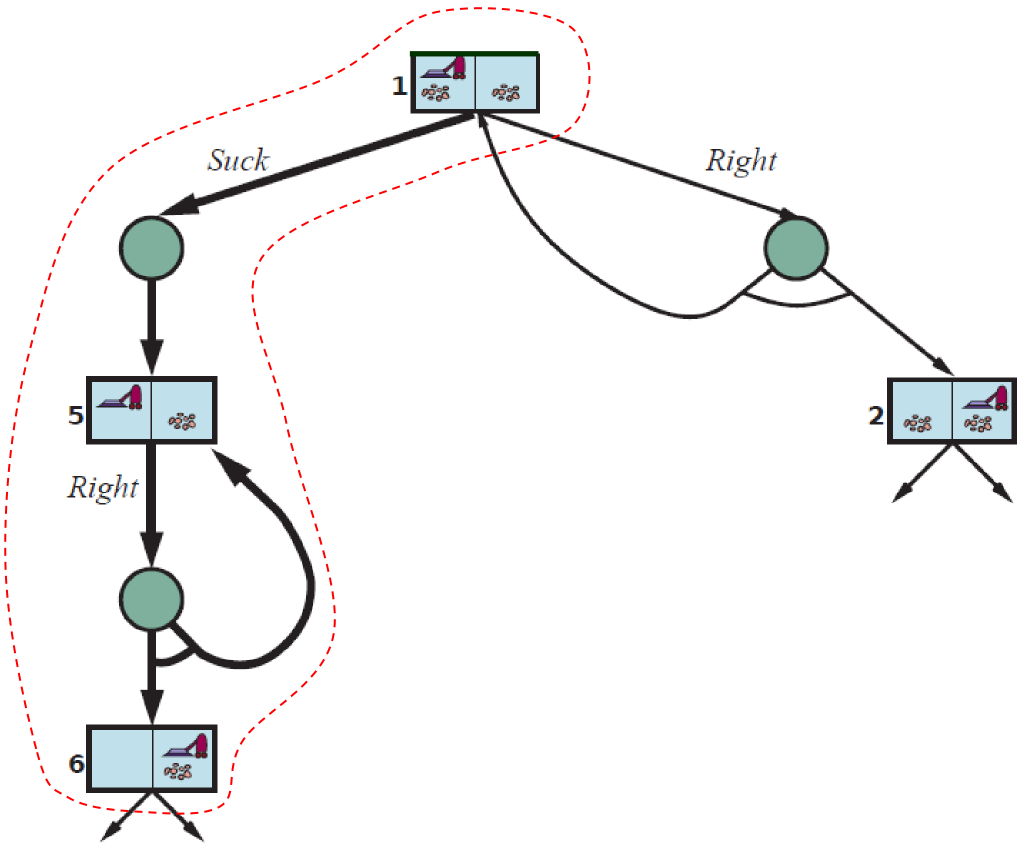
What if an action fails and the state is not changed?

Slippery vacuum world.

$$\text{State 1} \xrightarrow{Right} \{1, 2\}$$

$$\text{State 5} \xrightarrow{Right} \{5, 6\}$$

Cyclic solution



Cyclic Plan for a Solution

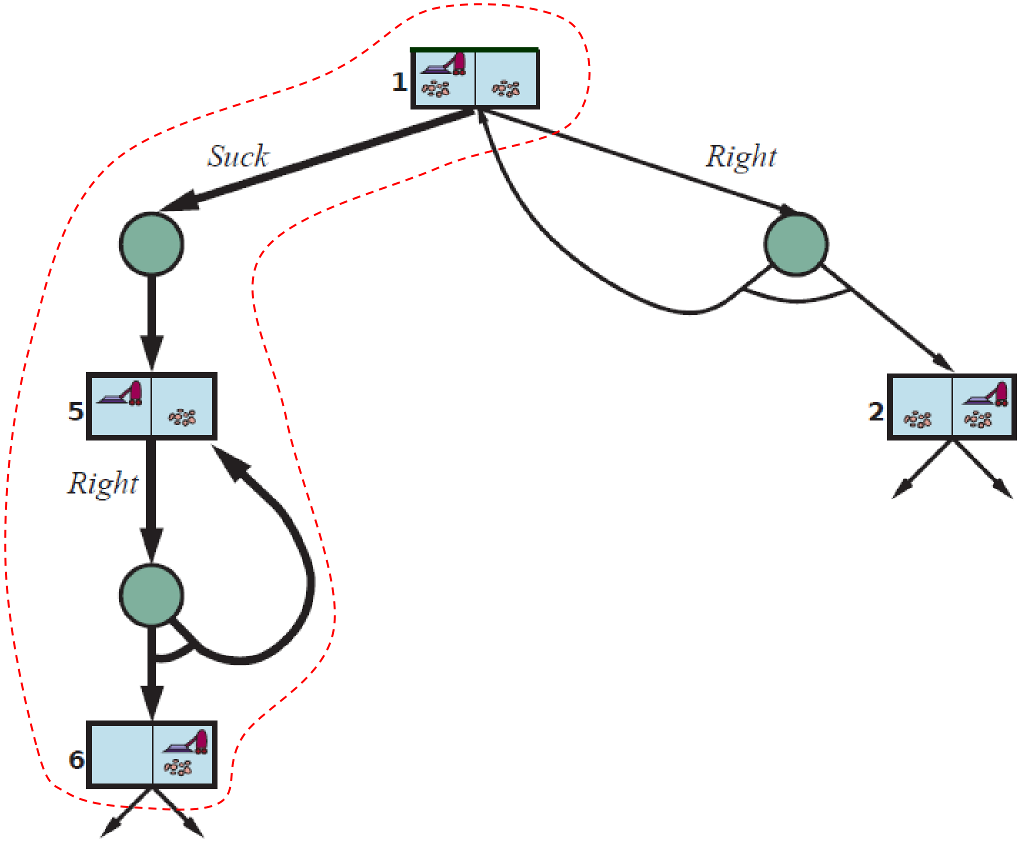
What if an action fails and the state is not changed?

Slippery vacuum world.

$$\text{State 1} \xrightarrow{\textit{Right}} \{1, 2\}$$

$$\text{State 5} \xrightarrow{\textit{Right}} \{5, 6\}$$

Cyclic solution



do
Suck;
if State = 5 then *Right*

Cyclic Plan for a Solution

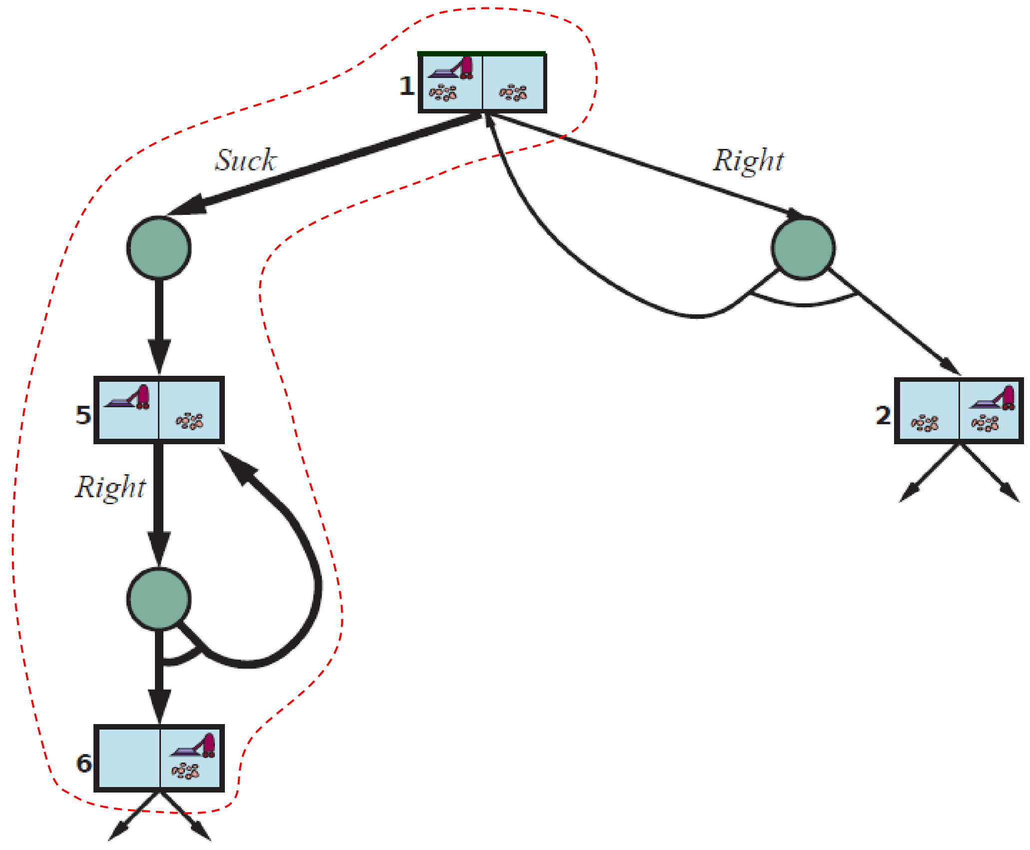
What if an action fails and the state is not changed?

Slippery vacuum world.

$$\text{State 1} \xrightarrow{\textit{Right}} \{1, 2\}$$

$$\text{State 5} \xrightarrow{\textit{Right}} \{5, 6\}$$

Cyclic solution →



do
  Suck;
  if State = 5 then *Right*

The goal will be reached provided that each outcome of a nondeterministic action eventually occurs.

Cyclic Plan for a Solution

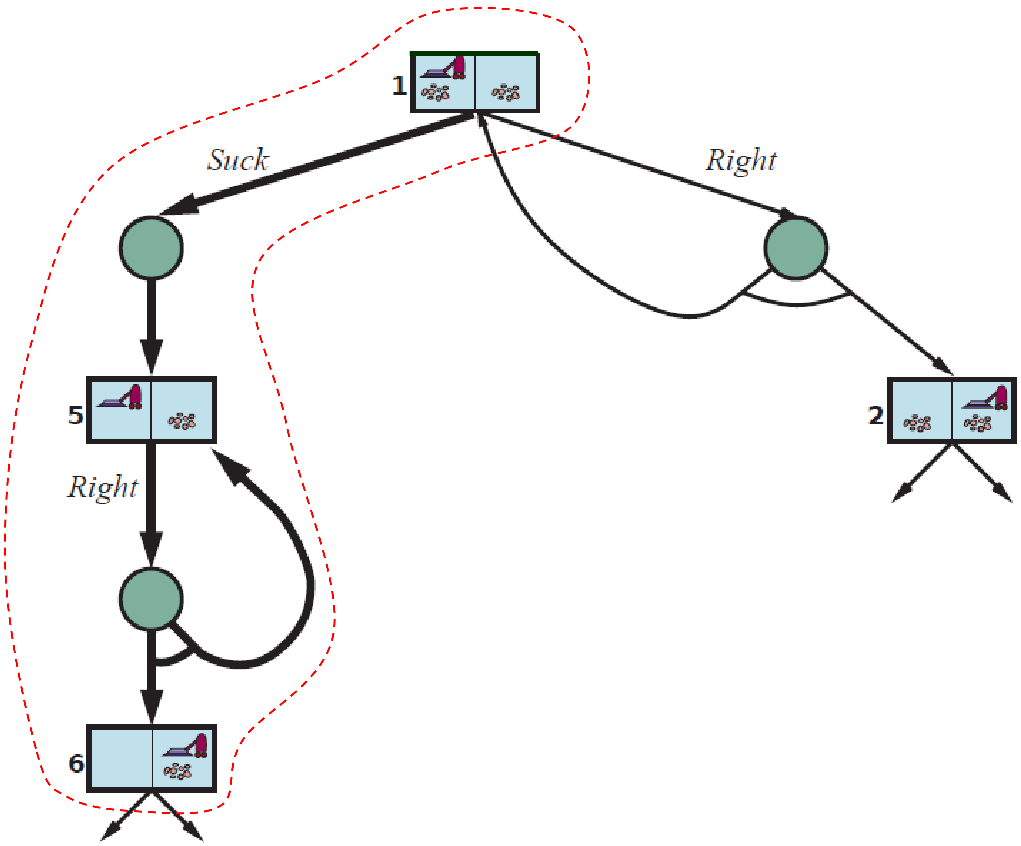
What if an action fails and the state is not changed?

Slippery vacuum world.

$$\text{State 1} \xrightarrow{\textit{Right}} \{1, 2\}$$

$$\text{State 5} \xrightarrow{\textit{Right}} \{5, 6\}$$

Cyclic solution



do
Suck;
if State = 5 then *Right*

The goal will be reached provided that each outcome of a nondeterministic action eventually occurs.