Inteligentny agent i jego otoczenie
Przeszukiwanie dla problem-solving

wykład 2

dr inż. Joanna Kołodziejczyk

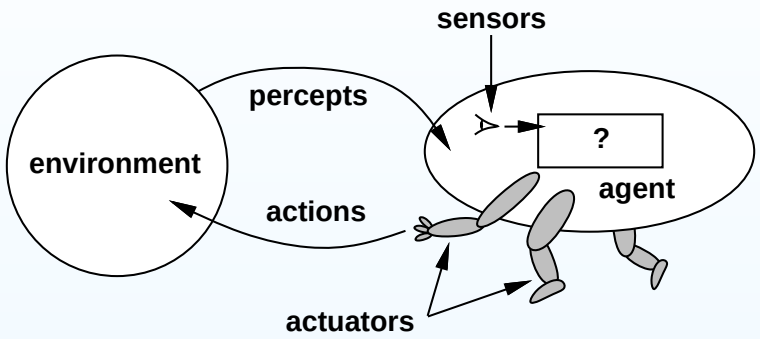
jkolodziejczyk@wi.ps.pl

Zakład Sztucznej Inteligencji ISZiMM

Plan wykładu

1. Część pierwsza
   (a) Agent i jego otoczenie
   (b) Racjonalność
   (c) Ocena działania, otoczenie, czujniki
   (d) Typy środowisk w jakim może pracować agent
   (e) Typy agentów
2. Część druga
   (a) Agent do problem-solving
   (b) Przykłady zadań problem-solving
   (c) Sformułowanie problemu
   (d) Drzewo jako struktura danych dla zadań problem-solving
   (e) Podstawowe algorytmy przeszukiwania

Agent i jego otoczenie

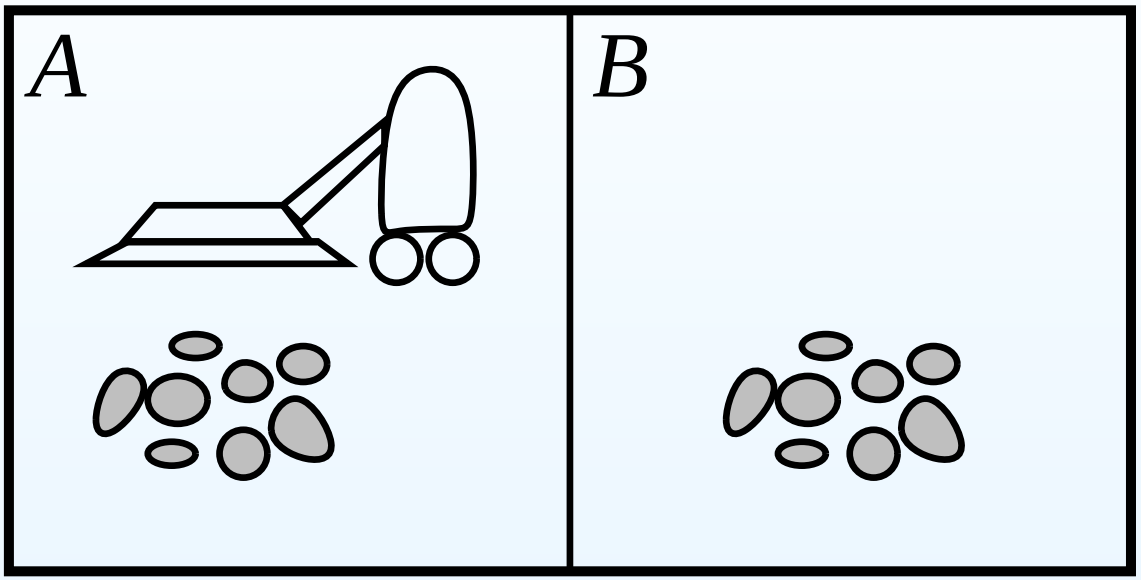


Inteligentny agent to np.: człowiek, robot, softbot, termostat, itp.
Funkcja agenta odwzorowuje percepcję (odczyty z czujników, klawiatury) lub historię takich odczytów na akcje:

$$f : \mathcal{P}^* \rightarrow \mathcal{A}$$

Program agenta uruchamiany jest w urządzeniu (architektura) by wygenerować f. Czyli jest to nic innego jak implementacja funkcji agenta.

Odkurzacz i jego świat



Percepcja: lokalizacja i stan podłogi, np., $[A, Brud]$
Akcja: $Lewo$, $Prawo$, $Odkurz$, Nic

Agent - odkurzacz

Sekwencje percepcji	Działanie
$[A, Czysto]$	$Prawo$
$[A, Brud]$	$Odkurz$
$[B, Czysto]$	$Lewo$
$[B, Brud]$	$Odkurz$
$[A, Czysto], [A, Czysto]$	$Prawo$
$[A, Czysto], [A, Brud]$	$Odkurz$
$[A, Czysto], [B, Czysto]$	$Lewo$
$\vdots$	$\vdots$

```
1  function Reflex-Vacuum-Agent(location,status) return action
2     if status = Brud then return Odkurz
3     else if location = A then return Prawo
4     else if location = B then return Lewo
```

Oczekiwane zachowanie agenta

Funkcja, którą agent ma wypełnić ma dawać takie efekty, iż zachowanie agenta będzie można określić jako „dobre”.

W przybliżeniu dobre akcje, to takie, które przynoszą jak najwięcej korzyści. Zatem należy znaleźć sposób na mierzenie odnoszonych przez agenta korzyści (miarę jego sukcesu).

Pełna specyfikacja agenta obejmuje:

- jak mierzyć korzyści z działania agenta,
- w jakim środowisku (z czym ma kontakt) agent i jak to wpływa na jego zachowanie o ile w ogóle,
- jak i jakie informacje przekazywane są do agenta,
- jak i jakie informacje stanowią informację zwrotną od agenta.

Mierzenie korzyści z działania agaetna

Miara osiągów to kryterium sukcesu. Nie ma uniwersalnego miernika dla agentów. Dla każdego zadania konstruktor określa takową miarę.

Przykład miary osiągów dla agenta-odkurzacza:

- Ile razy odkurzono brud w ciągu ośmiogodzinnej zmiany.
- Jedno pole sprzątane w jednostce czasu T.
- Jeden punkt za posprzątanie pola w jednostce czasu minus jeden punkt za ruch, który powoduje zużycie energii.

Lepiej konstruować miarę osiągów według tego jak agent ma zmieniać środowisko, a nie jak ma się zachowywać w oderwaniu od niego.

Agent racjonalny

Agent racjonalny wybiera taką akcję dla pojawiającej się sekwencji wejściowej, która maksymalizuje oczekiwaną wartość miary osiągów, określoną na podstawie wejść i wiedzy jaka jest w agencie zapamiętana.

Racjonalny $\neq$ wszechwiedzący
– percepcja nie musi dostarczać całkowitej wymaganej informacji
Racjonalny $\neq$ jasnowidzący
– wynik akcji nie musi być zgodny z oczekiwaniami
Racjonalny $\neq$ odnoszący zawsze sukces
Racjonalny $\rightarrow$ zbiera informacje (zwiększenie szans na nagrodę), uczy się (rozszerzanie wiedzy), jest autonomiczny (więcej niż tylko wbudowana wiedza - zmiana zachowań na podstawie doświadczenia).

Automatyczna taksówka

Aby stworzyć racjonalnego agenta należy określić PEAS, czyli Performace, Environement, Actuators, Sensors. Ważne jest, że agent mysi działać zgodnie z oczekiwaniami czyli miarą osiągów.

Rozważmy agenta - automatyczna taksówka:

1. Ocena osiągów
2. Otoczenie
3. Aktualizatory
4. Czujniki

Automatyczna taksówka

Aby stworzyć racjonalnego agenta należy określić PEAS, czyli Performace, Environement, Actuators, Sensors. Ważne jest, że agent mysi działać zgodnie z oczekiwaniami czyli miarą osiągów.

Rozważmy agenta - automatyczna taksówka:

1. Ocena osiągów — bezpieczeństwo, cel podróży, profit, legalność, komfort
2. Otoczenie — ulice, ruch uliczny, piesi, pogoda

3. Aktualizatory — sterowanie kierunkiem ruchu, gazem, hamulcem, klakson, mikrofon
4. Czujniki — kamera wideo, prędkościomierz, wskaźnik poziomu paliwa, czujniki silnika, klawiatura, GPS

Agent zakupów internetowych

1. Ocena osiągów
2. Otoczenie
3. Aktualizatory
4. Czujniki

Agent zakupów internetowych

1. Ocena osiągów — cena, jakość, zgodność z upodobaniami, efektywność
2. Otoczenie — obecne i przyszłe strony WWW, sprzedawca, dostawca
3. Aktualizatory — wyświetlacz dla użytkownika, śledzenie linków, wypełnianie formularzy
4. Czujniki — tekst, grafika, pismo

Typy otoczenia

	Krzyżówka	Warcaby	Analiza obrazów	taksówka
Obserwowalne				
Deterministyczne				
Epizodyczne				
Statyczne				
Dyskretne				
Pojedynczy agent				

Typy otoczenia

Otoczenie w pełni obserwowalne (czujniki wykrywają wszystkie istotne aspekty, które są niezbędna dla wykonania dobrej akcji) lub częściowo obserwowalne.

	Krzyżówka	Warcaby	Analiza obrazów	taksówka
Obserwowalne	w pełni	w pełni	w pełni	częściowo
Deterministyczne				
Epizodyczne				
Statyczne				
Dyskretne				
Pojedynczy agent				

Typy otoczenia

Deterministyczne lub stochastyczne: jeżeli kolejny stan otoczenia jest całkowicie określony na podstawie stanu bieżącego i wykonanej akcji, to otoczenie jest deterministyczne. Niepewność skutkuje otoczeniem stochatycznym.

	Krzyżówka	Warcaby	Analiza obrazów	taksówka
Obserwowalne	w pełni	w pełni	w pełni	częściowo
Deterministyczne	tak	stochast.	tak	stochast.
Epizodyczne				
Statyczne				
Dyskretne				
Pojedynczy agent				

Typy otoczenia

Epizodyczne lub sekwencyjne: W otoczeniu epizodycznym doświadczenie agenta może być podzielone na atomowe kroki, w których agent postrzega i wykonuje pojedynczą akcję. Wybór działania zależy jedynie od listy epizodów, a nie od działania wykonanego poprzednio.

	Krzyżówka	Warcaby	Analiza obrazów	taksówka
Obserwowalne	w pełni	w pełni	w pełni	częściowo
Deterministyczne	tak	stochast.	tak	stochast.
Epizodyczne	sekwen.	sekwen.	tak	sekwen.
Statyczne				
Dyskretne				
Pojedynczy agent				

Typy otoczenia

Statyczne lub dynamiczne: Jeżeli otoczenie może zmieniać się w czasie kiedy agent wybiera akcję, to środowisko jest dynamiczne. Semi-dynamiczne to takie, gdy miara osiągów agenta zmienia się nawet, gdy otoczenie jest niezmienne w czasie.

	Krzyżówka	Warcaby	Analiza obrazów	taksówka
Obserwowalne	w pełni	w pełni	w pełni	częściowo
Deterministyczne	tak	stochast.	tak	stochast.
Epizodyczne	sekwen.	sekwen.	tak	sekwen.
Statyczne	tak	tak	semi	dynamic.
Dyskretne				
Pojedynczy agent				

Typy otoczenia

Dyskretne lub ciągłe: Zależy od tego jak widziany jest czas zmian stanu otoczenia oraz percepcji i dziania agenta.

	Krzyżówka	Warcaby	Analiza obrazów	taksówka
Obserwowalne	w pełni	w pełni	w pełni	częściowo
Deterministyczne	tak	stochast.	tak	stochast.
Epizodyczne	sekwen.	sekwen.	tak	sekwen.
Statyczne	tak	tak	semi	dynamic.
Dyskretne	tak	tak	ciągłe	ciągłe
Pojedynczy agent				

Typy otoczenia

Pojedynczy lub multiagent: Czy otoczenie zawiera innych agentów, którzy również maksymalizują swoją miarę osiągów? Ich działanie zależy od działania rozpatrywanego agenta.

	Krzyżówka	Warcaby	Analiza obrazów	taksówka
Obserwowalne	w pełni	w pełni	w pełni	częściowo
Deterministyczne	tak	stochast.	tak	stochast.
Epizodyczne	sekwen.	sekwen.	tak	sekwen.
Statyczne	tak	tak	semi	dynamic.
Dyskretne	tak	tak	ciągłe	ciągłe
Pojedynczy agent	tak	multi	tak	multi

Typy otoczenia

	Krzyżówka	Warcaby	Analiza obrazów	taksówka
Obserwowalne	w pełni	w pełni	w pełni	częściowo
Deterministyczne	tak	stochast.	tak	stochast.
Epizodyczne	sekwen.	sekwen.	tak	sekwen.
Statyczne	tak	tak	semi	dynamic.
Dyskretne	tak	tak	ciągłe	ciągłe
Pojedynczy agent	tak	multi	tak	multi

Typ otoczenia silnie wpływa na projekt agenta.

Rzeczywistość jest częściowo obserwowalna, stochastyczna, sekwencyjna, dynamiczna, ciągła, wielo-agentowy.

Struktura agenta

$$agent = architektura + program$$

Architektura, to rodzaj urządzenia zapewniającego sensory i aktualizatory.

Program musi być dostosowany do architektury.

Kurs poświęcony jest projektowaniu programów agentów.

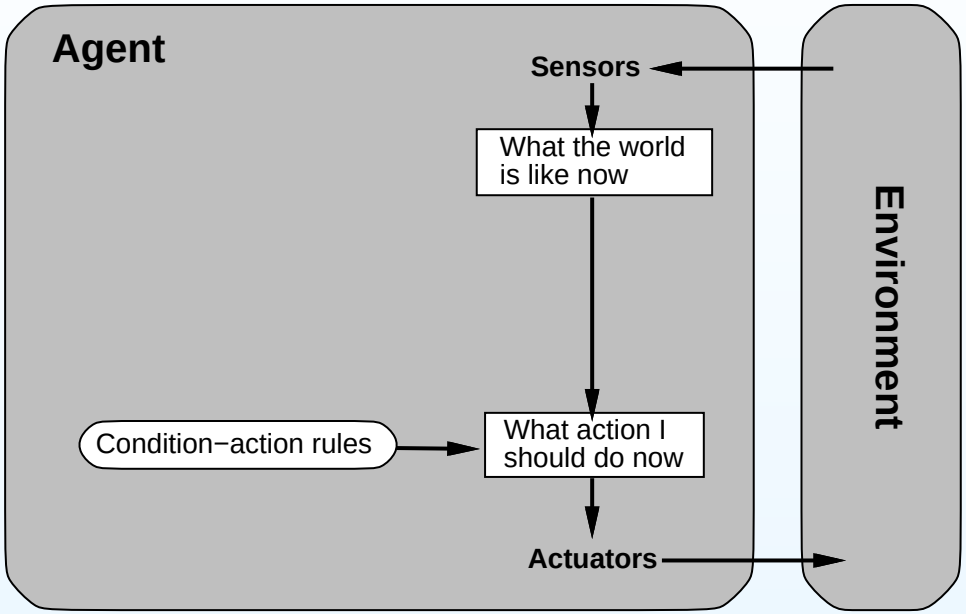
Typy agentów

Cztery główne typy w kolejności zwiększającej się ogólności ich zastosowań

- Prosty agent działający na zasadzie reakcji — reflex agent.
- Agent działający na zasadzie reakcji oparty na modelu — model-based reflex agent.
- Agent ukierunkowany — goal-based agent.
- Agent użyteczny — utility-based agent.

Wszystkie mogą dodatkowo zawierać moduł odpowiedzialny za uczenie się.

Reflex agent

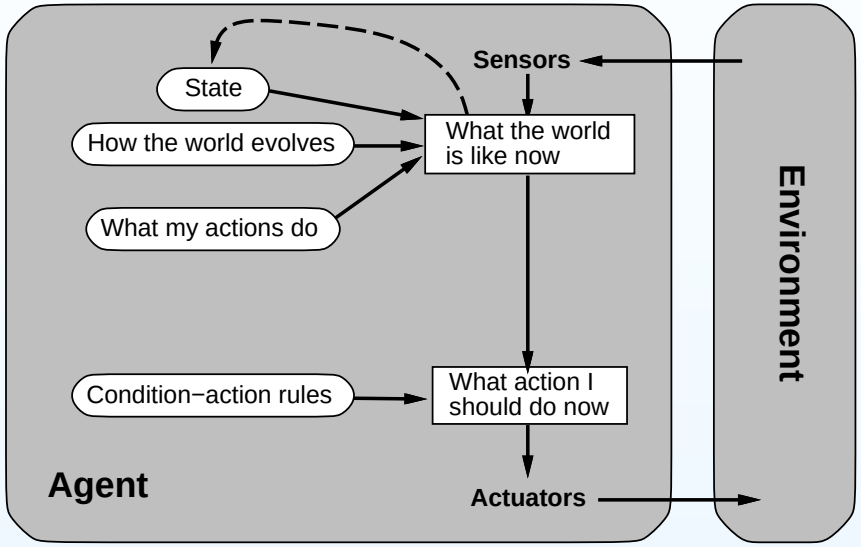



```
1  function Simple-Reflex-Agent(percept) return action
2     static: rules: a set of condition-action rules
3
4     state = InterpretInput(percept)
5     rule = RuleMatch(state, rules)
6     action = RuleAction[rule]
```

Reflex agent

- nie wykorzystuje historii percepcji
- condition-action rule to np.: if samochód przed tobą then zainicjalizuj hamowanie
- może pracować tylko w środowisku w pełni obserwowalnym
- sposobem na uniknięcie nieskończonych pętli zachowań (powstających przy niepełnej obserwowalności) jest stosowanie losowych zmian w akcjach

Model-based reflex agents

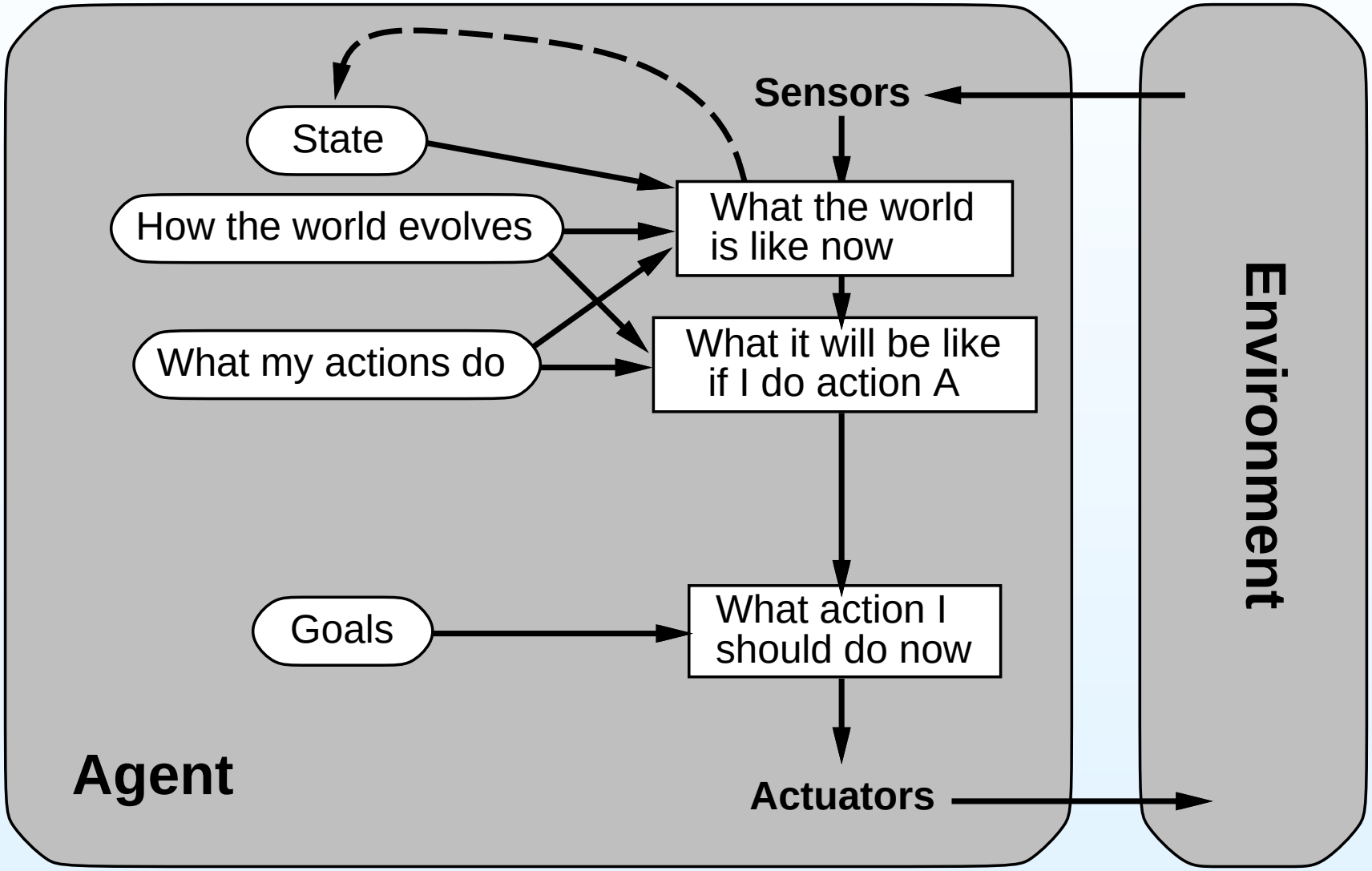



```
1  function Model-based-Reflex-Agent(percept) return action
2     static: rules: a set of condition-action rules
3           state: a description of the current world state
4           action: the most recent action, initially none
5
6     state = UpdateState(state,action,percept)
7     rule = RuleMatch(state, rules)
8     action = RuleAction[rule]
```

Model-based reflex agents

- przechowuje pewną wiedzę o stanach np. przeszłych, które teraz nie są obserwowalne
- podjęcie akcji zależy od tego jak wyewoluowało środowisko
- przechowuje informacje jak jego własne akcje wpłyną na otoczenie, czyli przechowuje wiedzę „how the world works”, co nazywa się modelem świata
- funkcja uaktualniania stanu jest odpowiedzialna za stworzenie wewnętrznego opisu stanu jak i interpretację w świetle wiedzy o świecie i tym jak się zmienił

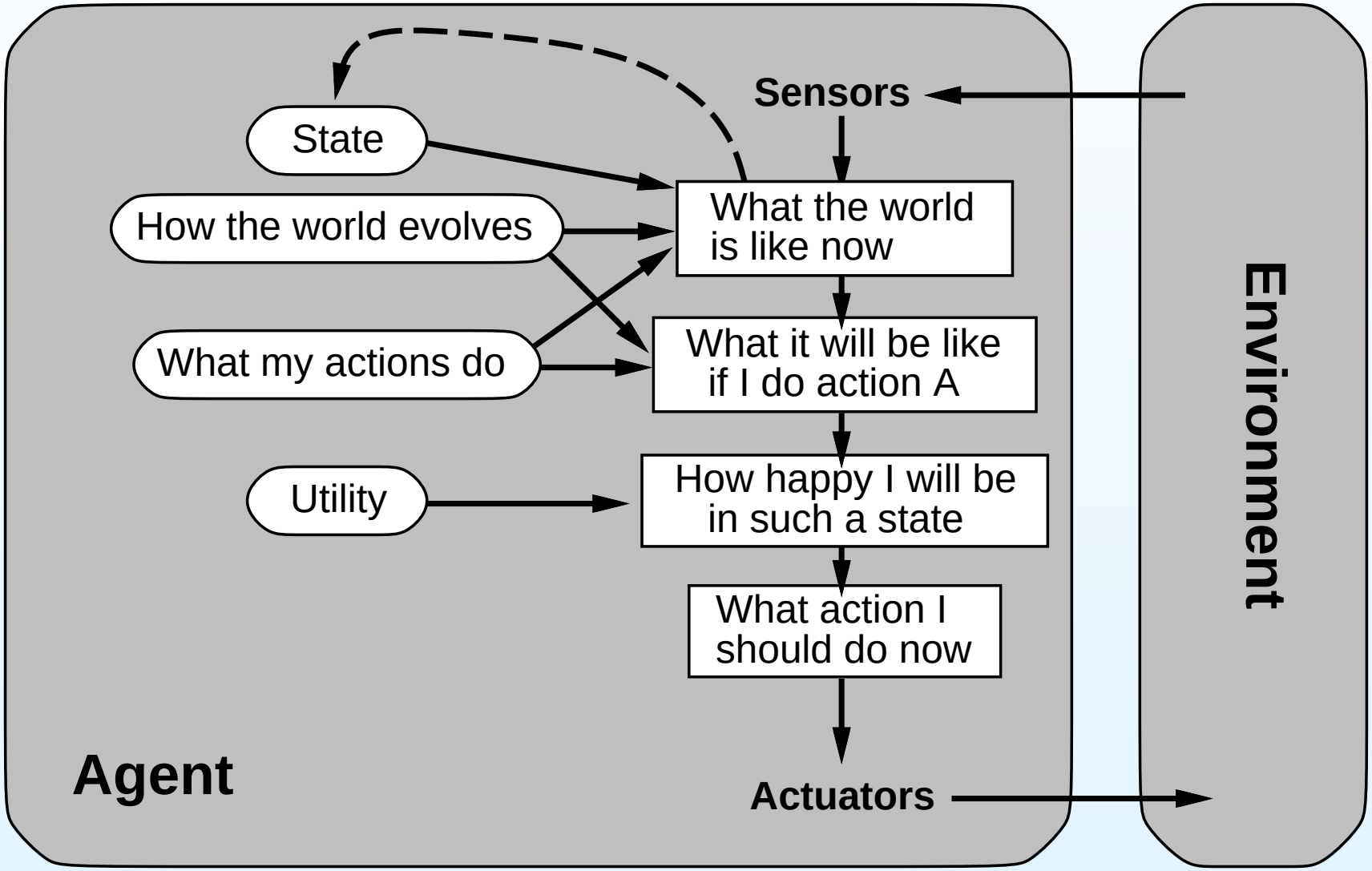
Goal-based agents



Goal-based agents

- podejmowana decyzja może być zależna nie tylko od bieżącego stanu, ale od celu jaki agent ma postawiony
- jeżeli cel osiągnie się przez wykonanie pojedynczej akcji, to zadanie jest proste
- w rzeczywistych zadaniach należy znaleźć długi i nie zawsze łatwy do znalezienie ciąg akcji prowadzący do celu
- przeszukiwani i planowanie, to poddziedziny w których wykorzystuje się goal-based agentów
- wiedza, na podstawie której podejmuje decyzje jest zewnętrzna i dlatego łatwiej ją zmieniać, a agent jest przenośny

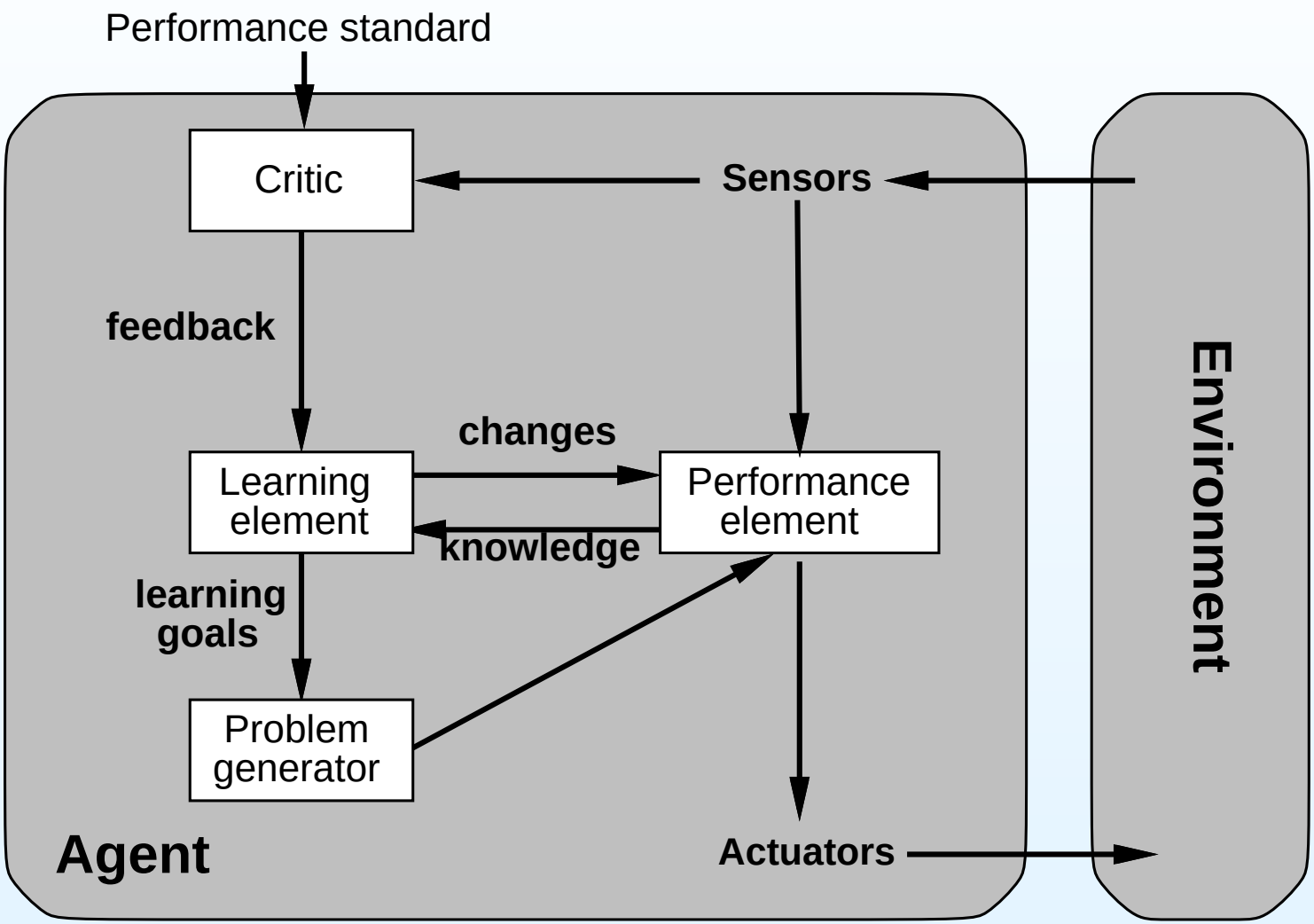
Utility-based agents



Utility-based agents

- sam cel nie zawsze wystarczy, można również oceniać jak osiągnąć go wydajniej
- tworzy się funkcję użyteczności, która oszacowuje jakość bieżącego stanu
- w wypadku kilku akceptowalnych sekwencji prowadzących do celu pozwala wybrać taką, o najlepszych parametrach użyteczności

Learning agents



Learning agents

- learning element — odpowiedzialny za postępy
- performance element — wybór akcji (wcześniej prezentowane typy agentów dokładnie tym się zajmowały)
- critic — ocenia zachowanie bieżące agenta, by ewentualnie skorygować jego zachowanie w przyszłości i przekazuje te informacje do moduły uczącego się
- problem generator — sugeruje akcje prowadzące do nowych doświadczeń

Rozwiązywanie problemów przez przeszukiwanie

czyli jak zbudować agenta, który znajdzie sekwencję akcji prowadzącą do oczekiwanego celu (lub celów).

Problem-solving agent to agent typu goal-based. Problem, to elementy wśród których szuka się elementu solution.

Problem-solving agent

Cztery główne kroki w zadaniach (problem-solving)

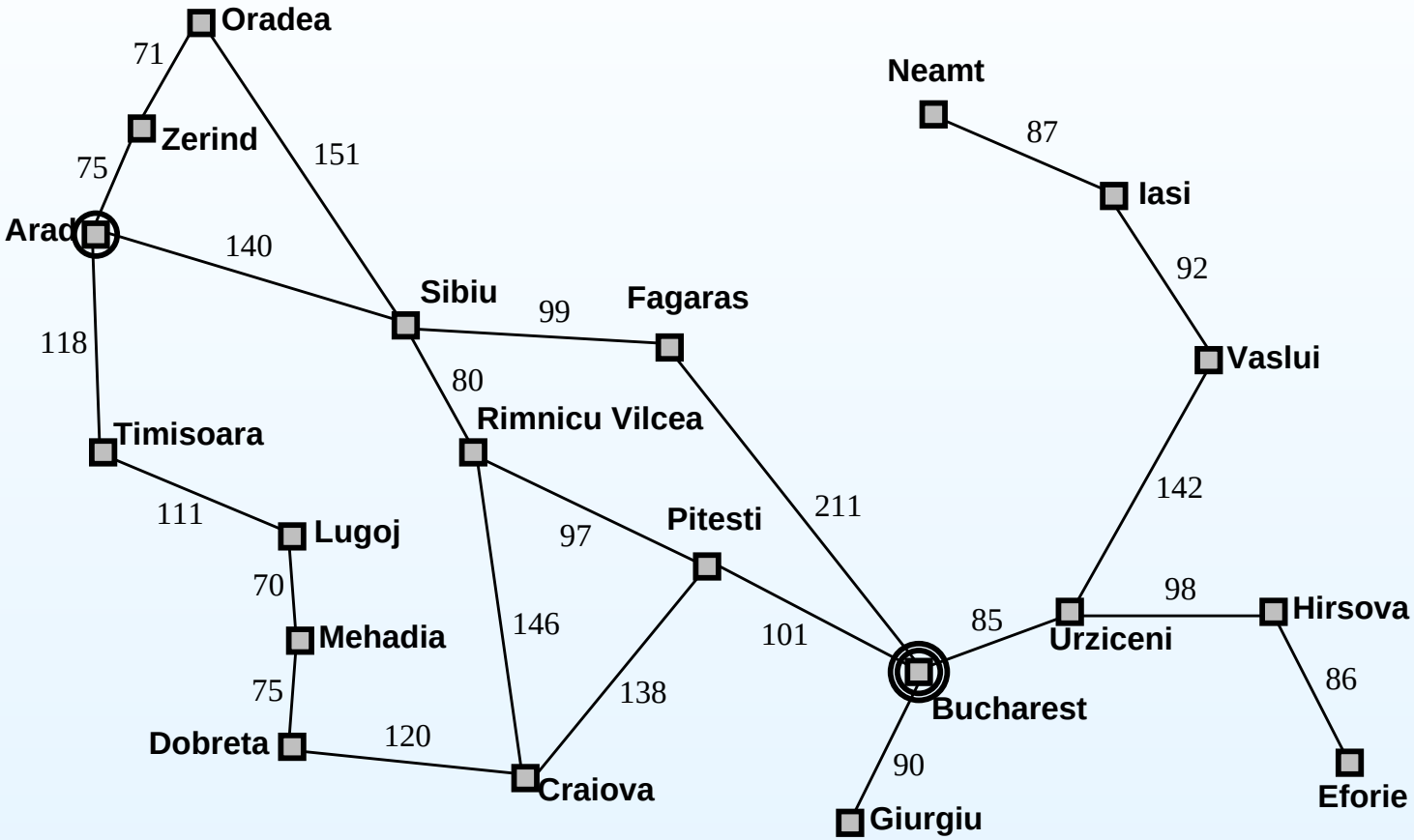
1. Sformułowanie celu: Jaki stan dla rozpatrywanego mikroświata jest stanem oczekiwanym (zwycięskim)?
2. Określenie problemu: Jakie stany i akcje należy rozważyć by osiągnąć cel?
3. Poszukiwanie rozwiązania: Określenie możliwej sekwencji akcji, która prowadzi do stanów o znanych wartościach i wybór najlepszej sekwencji.
4. Wykonanie: Podaj rozwiązanie jako ciąg wykonanych akcji.

Problem-solving agent

```
1  function Simple-Problem-Solving-Agent(percept) return action
2     input: pecept, a percept
3     static: seq: an action sequence, initially empty
4           state: some description of the current world state
5           goal: a goal, initially null
6           problem: a problem formulation
7     state = Update-State(state, percept)
8     if seq is empty then
9        goal = Formulate-Goal(state)
10       problem = Formulate-Problem(state, goal)
11       seq = Search( problem)
12    action = First(seq)
13    seq = Rest(seq)
14    return action
```

Uwaga: Rozpatrywane problem-solving jest oparte na pełnej wiedzy o problemie.

Przykład: Rumunia



Przykład: Rumunia

Jesteś na wakacjach w Ruminii. Obecnie w Aradzie. Jutro masz samolot z Bukaresztu.

Określenie celu: dotrzeć do Bukaresztu

Określenie problemu:

- stany: różne miasta
- akcje: podróż pomiędzy miastami

Określenie rozwiązania: porządek odwiedzanych miast np. Arad, Sibiu, Fagaras, Bukareszt

W jakim środowisku pracuje problem-solving agent?

- Statyczne — określenie problem-solving (stanów i rozwiązania) jest określane bez udziału wpływu środowiska i zmian jakie mogą w nim nastąpić.
- W pełni obserwowalne — stan inicjalizujący jest dany.
- Dyskretne — alternatywne akcje mają charakter wyliczeniowy.
- Deterministyczne — rozwiązanie jest sekwencją akcji i nie ma możliwości wystąpienia nieprzewidzianych wydarzeń.

Poprawne określanie problem-solving

Problem definiuje się przez :

- Stan początkowy: np. Arad
- Akcja - funkcja generująca potomka: $S(X)$ zbiór par $< akcja, stannastpny >$ np. $S(Arad) = \{< Arad \rightarrow Zerind, Zerind >, \ldots\}$
- Stan końcowy - cel:
  - Wyraźny: np. $X = Bukareszt$
  - Ukryty: np. $szach - mat(X)$
- Koszt ścieżki: funkcja przypisująca numeryczną wartości do sekwencji stanów prowadzących do rozwiązania lub innego stanu pośredniego (informacja dodatkowa) np. suma odległości, numer wykonanej akcji, itp. $c(x, a, y)$ jest kosztem wykonania kroku i zakłada się, że ≥ 0

Rozwiązanie: To ciąg, w którym zapisane są kolejne akcje, jakie wykonano przechodząc od stany początkowego do stanu końcowego. Optymalne rozwiązanie ma najmniejszy koszt ścieżki.

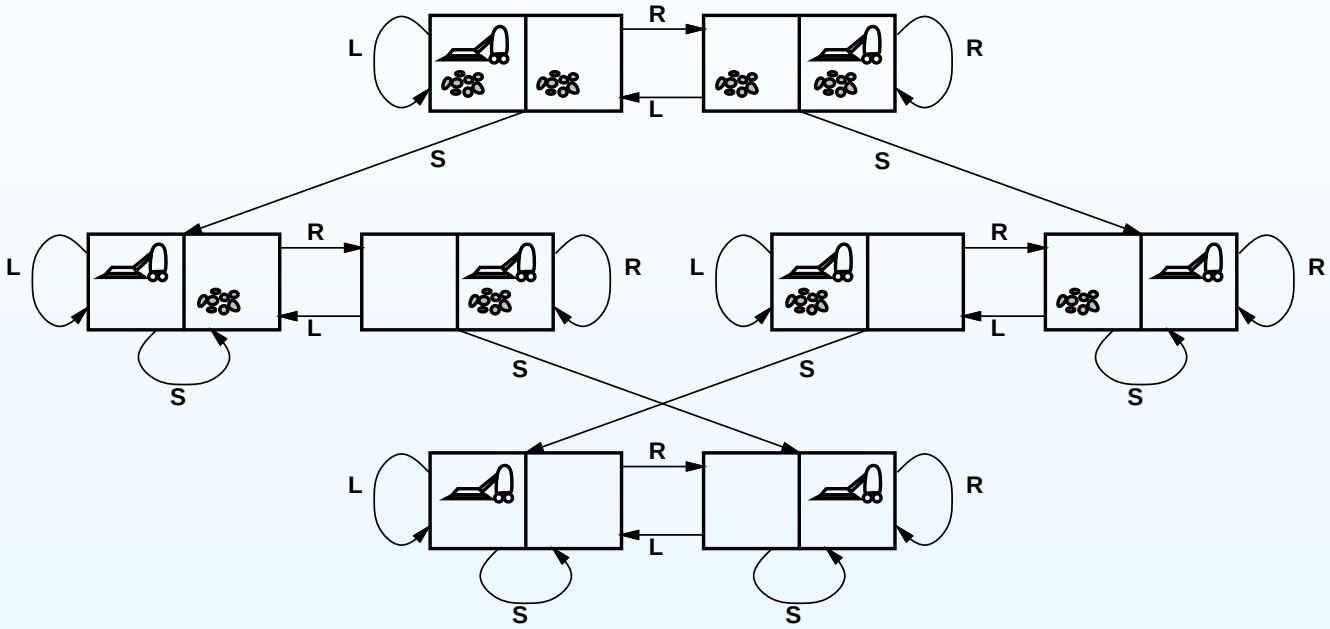
Szczegóły o określaniu problem-solving

Rzeczywistość jest szalenie skomplikowana i wymaga uproszczenia (abstraction) (wycięcia szczegółowych informacji) reprezentacji by możliwe było rozpatrywanie przestrzeni stanów.

- (Uproszczony) stan: zbiór stanów rzeczywistych
- (Uproszczona) akcja: kombinacja rzeczywistych działań
  — np: $Arad \rightarrow Zerind$ reprezentuje złożony zbiór możliwych dróg, miejsc na odpoczynek, itp.
  — Uproszczenie jest dopuszczalne jeżeli ścieżka odzwierciedli rzeczywisty świat.
- (Uproszczone) rozwiązanie: zbiór ścieżek, które występują w świecie rzeczywistym.

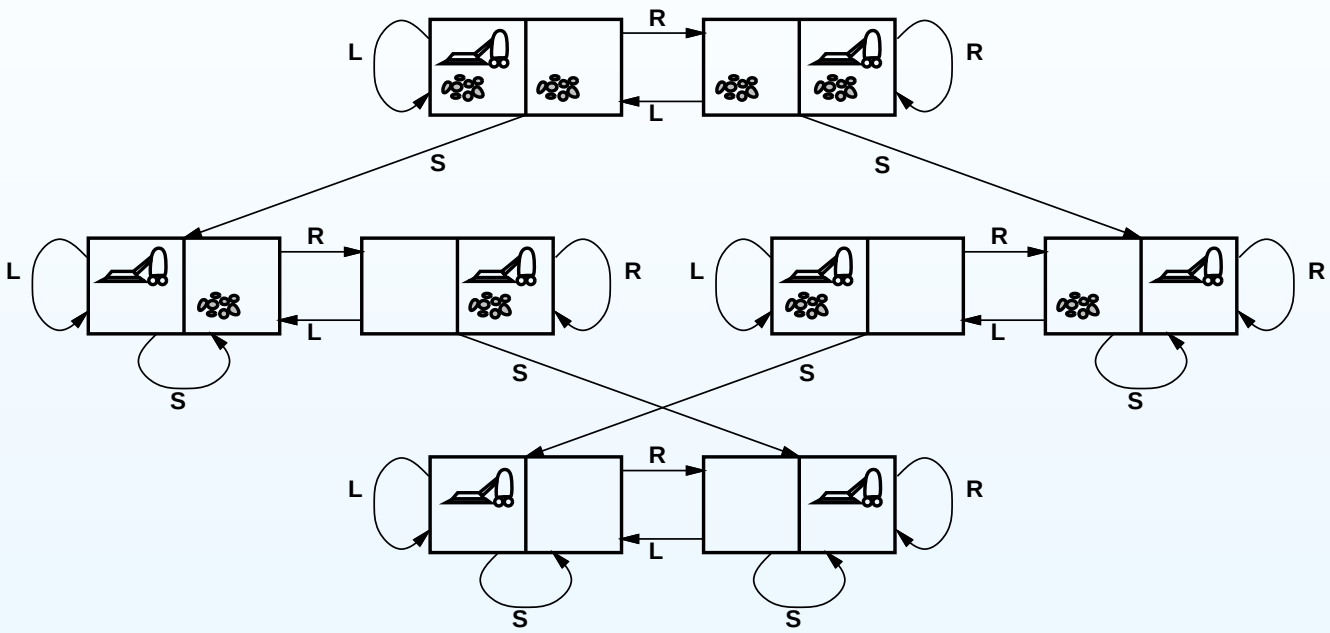
Każda uproszczona akcja jest łatwiejsza niż problem rzeczywisty.

Przykłady określania problem-solving: odkurzacz



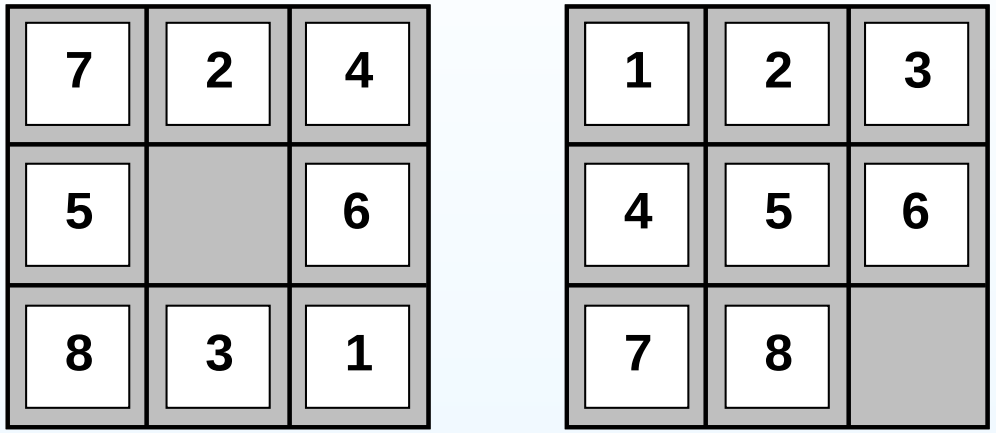
- Stany:
- Stan początkowy:
- Akcja:
- Jak sprawdzić cel:
- Koszt ścieżki:

Przykłady określania problem-solving: odkurzacz



- Stany: dwa pola z brudem lub bez: $2 \cdot 2^2 = 8$ stanów.
- Stan początkowy: Każdy stan może nim być.
- Akcja: $\{Lewo(L), Prawo(R), Odkurzaj(S)\}$
- Jak sprawdzić cel: Czy oba pola czyste?
- Koszt ścieżki: Liczba wykonanych akcji do osiągnięcia celu. Koszt akcji np. 1

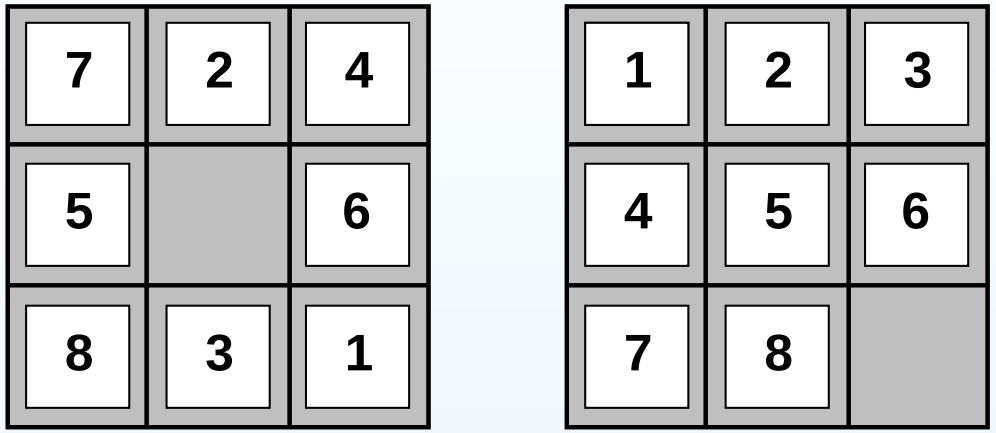
Przykłady cd: układanka ośmioelementowa



Start State **Goal State**

- Stany:
- Stan początkowy:
- Akcja:
- Jak sprawdzić cel:
- Koszt ścieżki:

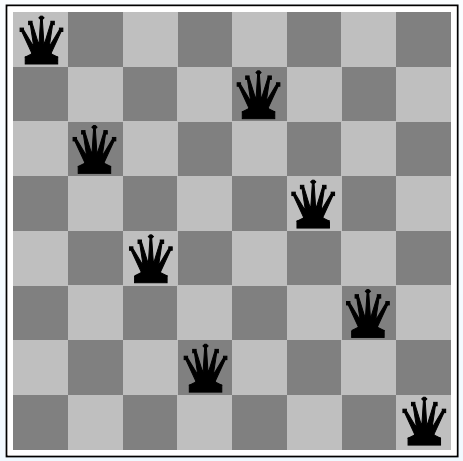
Przykłady cd: układanka ośmioelementowa



Start State **Goal State**

- Stany: określa położenie każdego klocka i pustego pola.
- Stan początkowy: Każdy stan może nim być.
- Akcja: $\{Left, Right, Up, Down\}$
- Jak sprawdzić cel: Sprawdź czy położenie końcowe zostało uzyskane.
- Koszt ścieżki: Liczba wykonanych akcji do osiągnięcia celu. Koszt kroku np. 1.

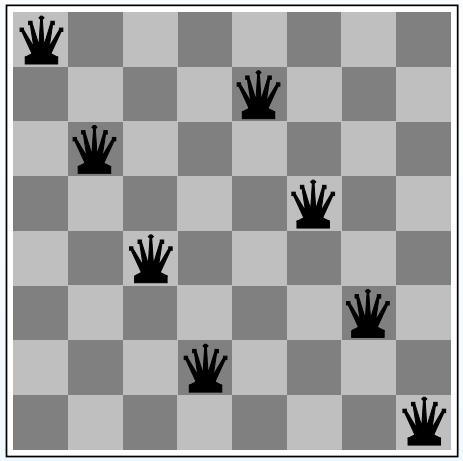
Przykłady cd: 8 hetmanów



Podejście inkrementacyjne

- Stany:
- Stan początkowy:
- Akcja:
- Jak sprawdzić cel:
- Koszt ścieżki:

Przykłady cd: 8 hetmanów

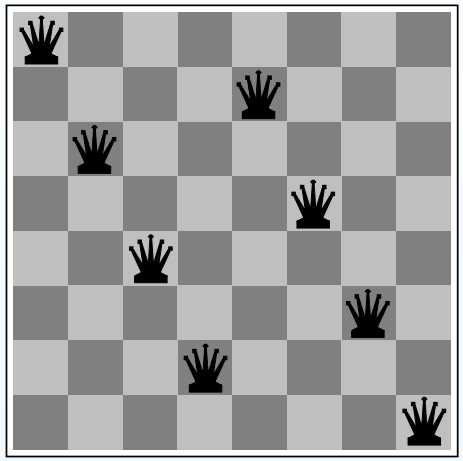


Podejście inkrementacyjne

- Stany: dowolne położenie od 0 do 8 hetmanów na szachownicy.
- Stan początkowy: Pusta szachownica.
- Akcja: Dodaj hetmana na pustym polu.
- Jak sprawdzić cel: Czy ustawione na szachownicy hetmany nie atakują się?
- Koszt ścieżki: zero - nie interesuje nas.

Istnieje 64^8 możliwych sekwencji do przebadania.

Przykłady cd: 8 hetmanów

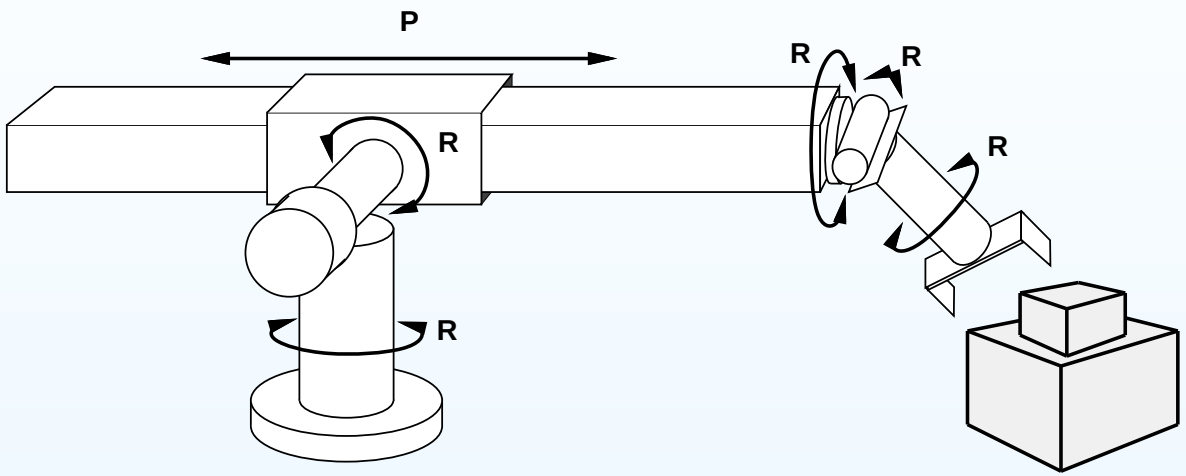


Podejście inkrementacyjne

- Stany: Ustawienie od 0 do 8 hetmanów nie atakujących się.
- Stan początkowy: Pusta szachownica.
- Akcja: Dodawaj hetmana w kolejnych kolumnach zaczynając od lewej dbając, by ustawiaby hetman nie atakował innych.
- Jak sprawdzić cel: Czy ustawione na szachownicy hetmany nie atakują się?
- Koszt ścieżki: zero - nie interesuje nas.

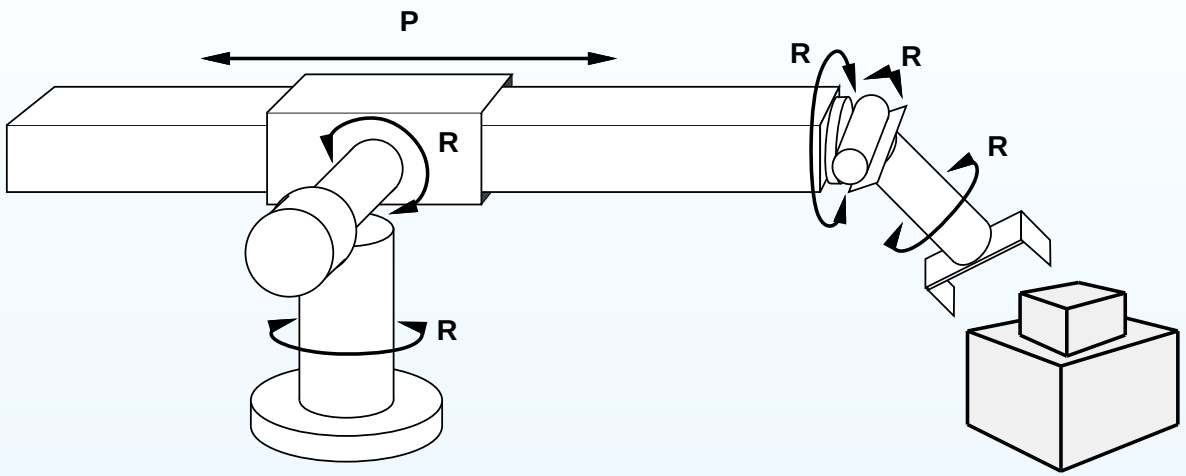
Istnieje 2057 możliwych sekwencji do przebadania. Zatem właściwe sformułowanie problemu może mieć duże znaczenie dla wyników.

Przykład: robot montażowy



- Stany:
- Stan początkowy:
- Akcja:
- Jak sprawdzić cel:
- Koszt ścieżki:

Przykład: robot montażowy



- Stany: real values: koordynaty położenia ramienia robota; części elementów do montażu.
- Stan początkowy: Dowolne położenie ramienia i dowolna konfiguracja elementów.
- Akcja: Ciągłe obroty ramienia w przegubach.
- Jak sprawdzić cel: Pełen montaż.
- Koszt ścieżki: Czas wykonania.

Podstawowy algorytm przeszukiwania

Jak znaleźć rozwiązanie dla problem-solving?

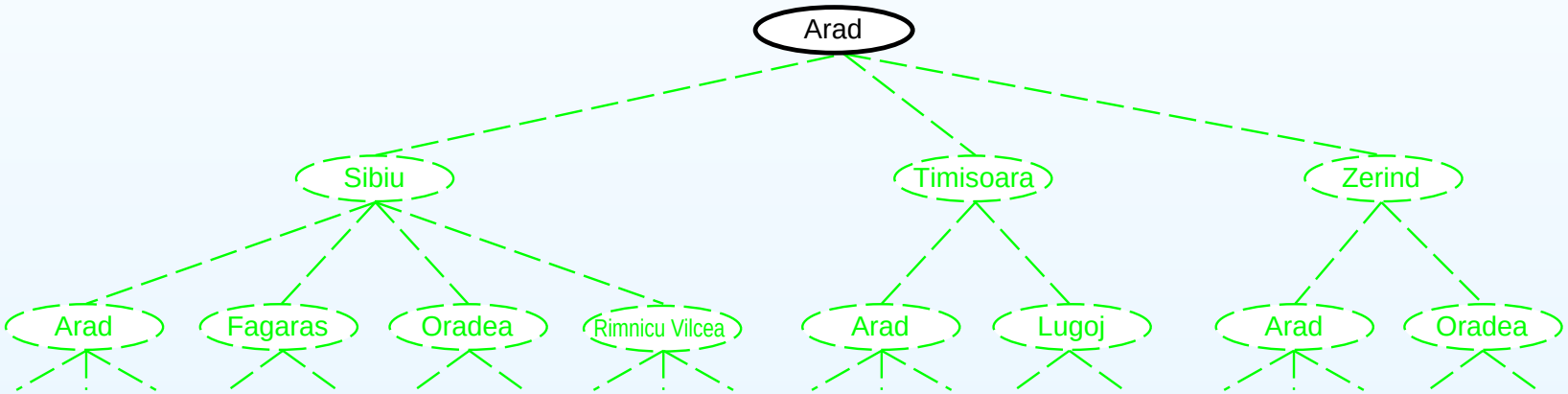
- Przeszukać przestrzeń stanów - należy pamiętać, że przestrzeń stanów zależy silnie od reprezentacji stanu.
- Przeszukiwanie przez jawne rozwijanie drzewa: korzeń drzewa = stan początkowy; węzły i liście generowane poprzez zastosowanie funkcji tworzącej potomków.
- Uogólniając, przeszukiwanie generuje graf, w którym ten sam węzeł może być wygenerowany na różnych ścieżkach.

Prosty algorytm przeszukiwania drzewa

```
1  function TREE-SEARCH(problem, strategy) return a solution or fail
2     Initialize search tree to the initial state of the problem
3
4     do
5     if no candidates for expansion then return failure
6     choose leaf node for expansion according to strategy
7     if node contains goal state then return solution
8     else expand the node and add resulting nodes to the search tree
9     enddo
```

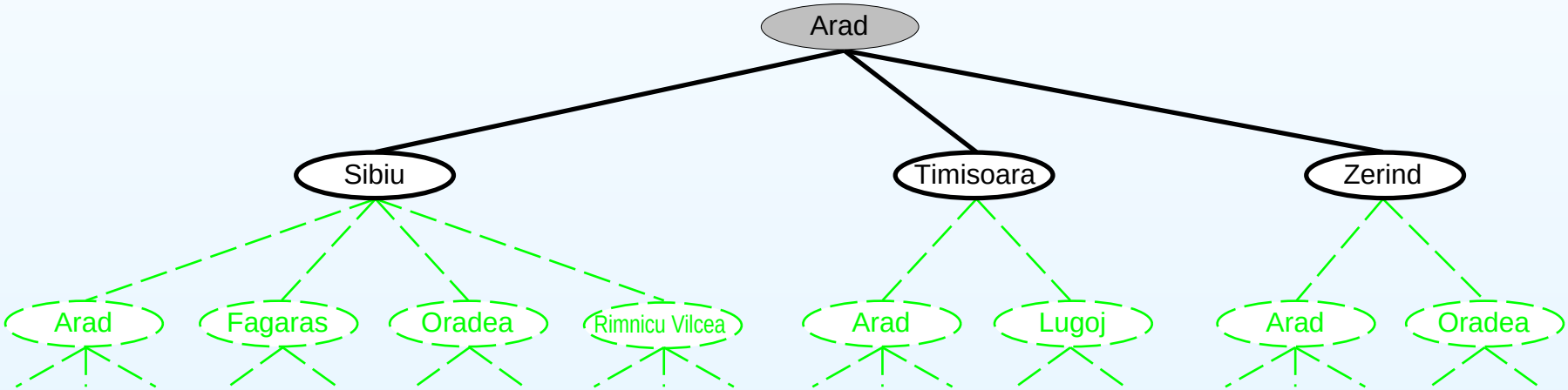
Przykład przeszukiwania drzewa

Stan początkowy



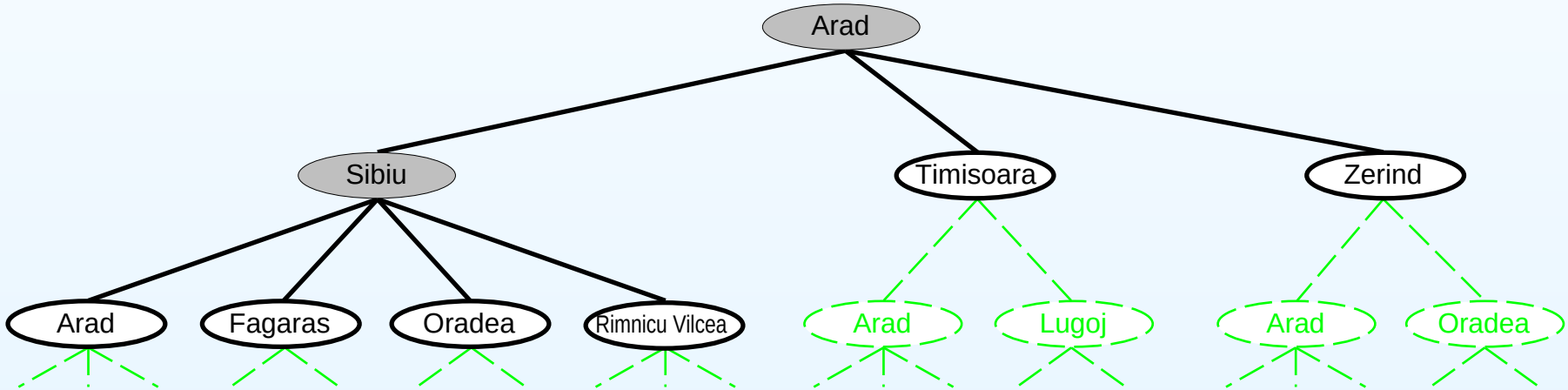
Przykład przeszukiwania drzewa

Po rozwinięciu węzła Arad

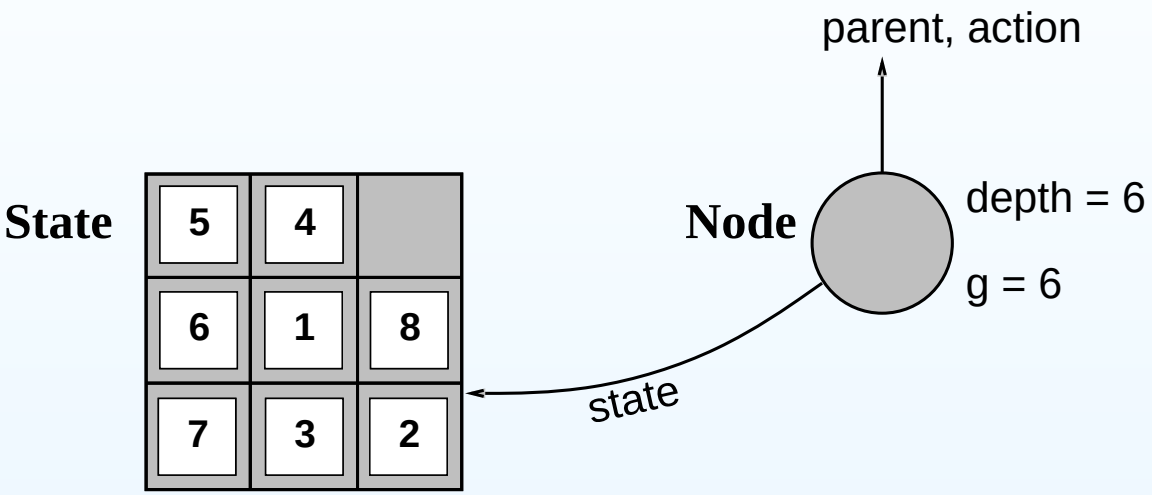


Przykład przeszukiwania drzewa

Po rozwinięciu węzła Sibiu



Implementacja stanów - węzły



Stan jest reprezentacją problem.

Węzeł jest strukturą danych stanowiącą część drzewa przeszukiwań.

Węzeł ma rodzica, dzieci, głębokość, koszt ścieżki $g(X)$.
$node =< state, parent - node, action, path - cost, depth >$

Węzły skrajne czekają na rozwinięcie z użyciem funkcji generującej potomków węzła rozwijanego.

Uwaga: Stany nie posiadają rodziców, potomków, głębokości itp.

Implementacja: przeszukiwanie drzewa

```
1  function TREE-SEARCH(problem,fringe) return a solution or failure
2     fringe = INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
3
4     do
5     if fringe is EMPTY then return failure
6     node = REMOVE-FIRST(fringe)
7     if GOAL-TEST(problem, STATE[node]) then return node
8     fringe = INSERT-ALL(EXPAND(node, problem), fringe)
```

```
1  function Expand( node, problem) returns a set of nodes
2     successors = the empty set
3     do for each action, result in Successor-Fn(problem, State[node])
4        s = a new Node
5        Parent-Node[s] = node; Action[s] = action; State[s] = result
6        Path-Cost[s] = Path-Cost[node] + Step-Cost(node, action, s)
7        Depth[s] = Depth[node] + 1
8        add s to successors
9     return successors
```

Strategie przeszukiwania

Strategie są rozróżniane przez kolejność rozwijania węzłów.

Kryteria oceny strategii przeszukiwania:

- Zupełność (completeness): czy zawsze znajdzie rozwiązanie jeżeli ono istnieje?
- Złożoność czasowa (time complexity): liczba węzłów generowana/rozwijana.
- Obszar zajmowanej pamięci (space complexity): maksymalna liczba węzłów w pamięci.
- Optymalność (optimality): czy zawsze znajdzie najmniej kosztowne rozwiązanie?

Złożoność obliczeniowa mierzona jest z uwzględnieniem:

- b — maksymalnego współczynnika rozgałęzienia w drzewie,
- d — głębokości najmniej kosztownego rozwiązania,
- m — maksymalnej głębokości przestrzeni stanów (może być ∞).

Ślepe (uninformed) strategie przeszukiwania

Strategie ślepe używają tylko danych z określenia problem-solving. Jedyne co wiedzą o rozwiązaniu, to jak ono wygląda.

- Przeszukiwanie wszerz — BFS (Breadth-first search).
- Przeszukiwanie o stałym koszcie — UCS (Uniform-cost search).
- Przeszukiwanie w głąb — DFS (Depth-first search).
- Przeszukiwanie w głąb z ograniczeniem (Depth-limited search).
- Iteracyjne zagłębianie (Iterative deepening search).