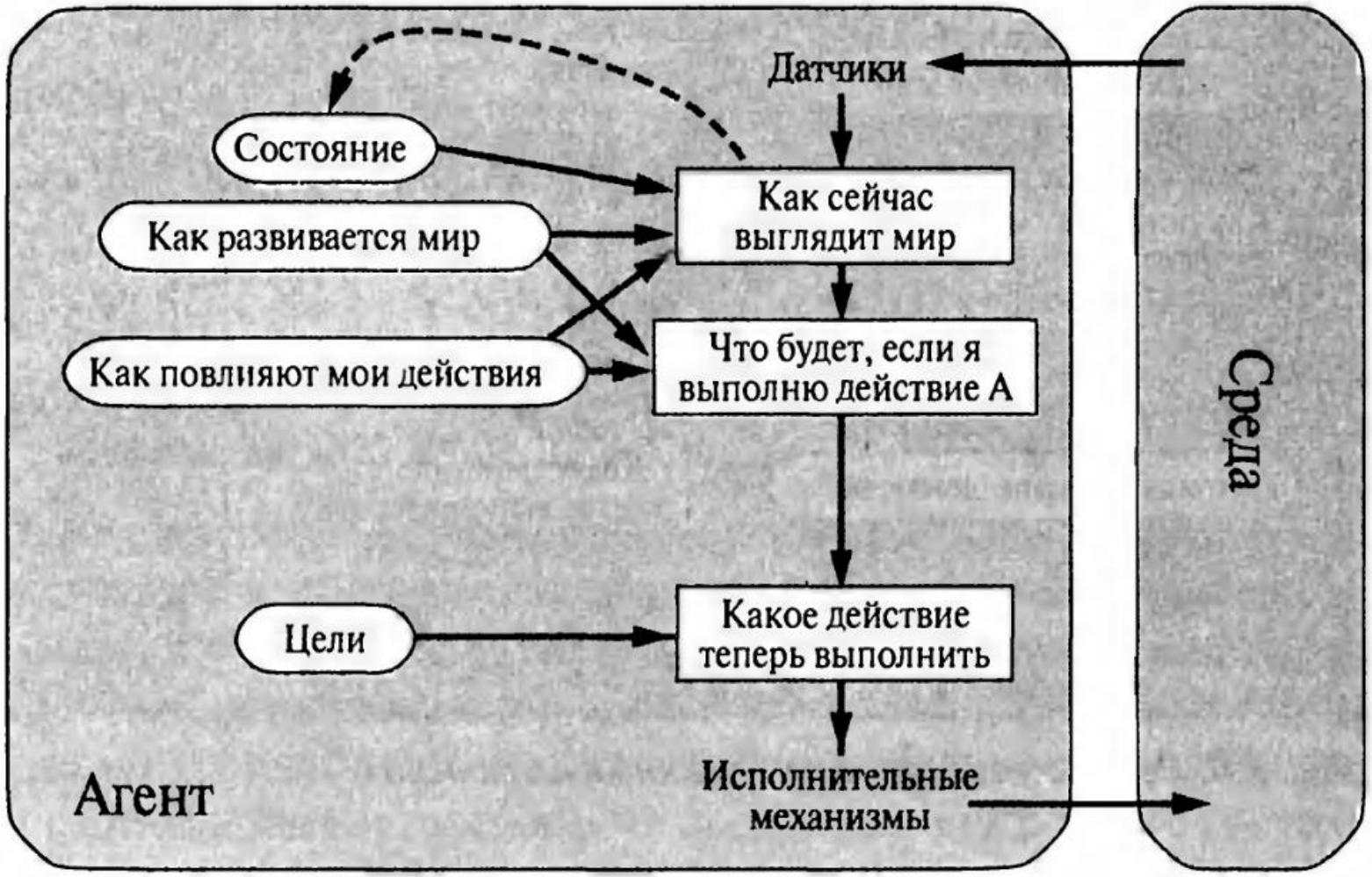
Машинное обучение и интеллектуальные системы

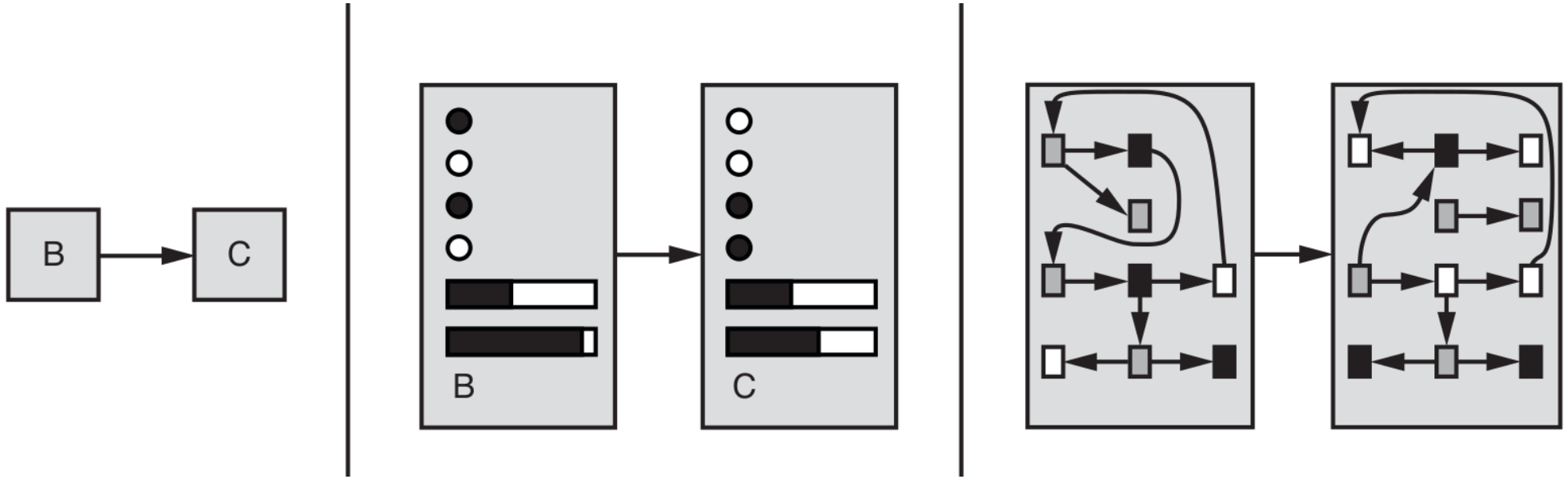
Лекция 2. Представления среды. Решение проблем при помощи поиска. Пространство состояний. Эвристики.

Агент с целью и моделью



Представление среды

Каким образом агент представляет окружающий мир?

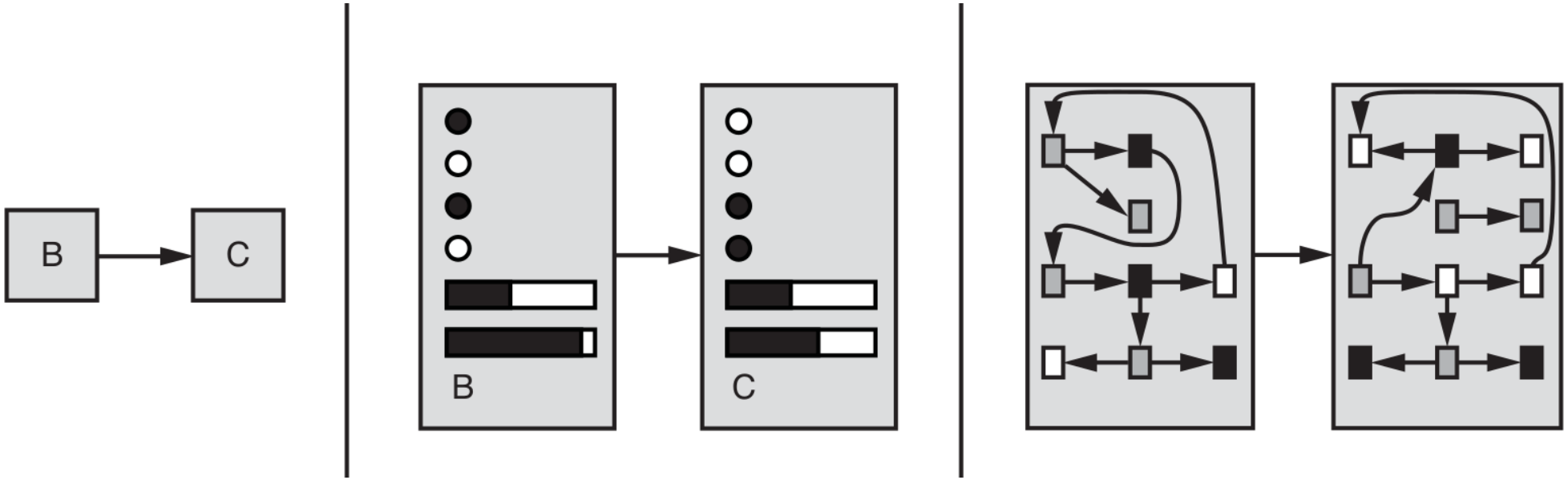


Атомарное Разложенное Структурированное

Представление среды

- Атомарное представление - состояния мира не имеют внутренней структуры. Примеры - планирование маршрута, где состояния - города. Система распознавания речи, где скрытые состояния - буквы. Применения: решение проблем (Problem solving), игры, марковские модели.
- Разложенное состояние описывается набором переменных. Можно говорить о сходстве и различии состояний. Пример - состояние автомобиля (координаты, скорость, уровень топлива, ...). Применения - задачи удовлетворения ограничений, пропозициональная логика, планирование, машинное обучение (большинство моделей).
- Структурированные представления - явное описание объектов, их составляющих и связей между ними. Применения: логика первого порядка, реляционные БД, онтологические модели, семантический анализ.

Представление среды



Атомарное Разложенное Структурированное

Выразительность
Сложность обработки

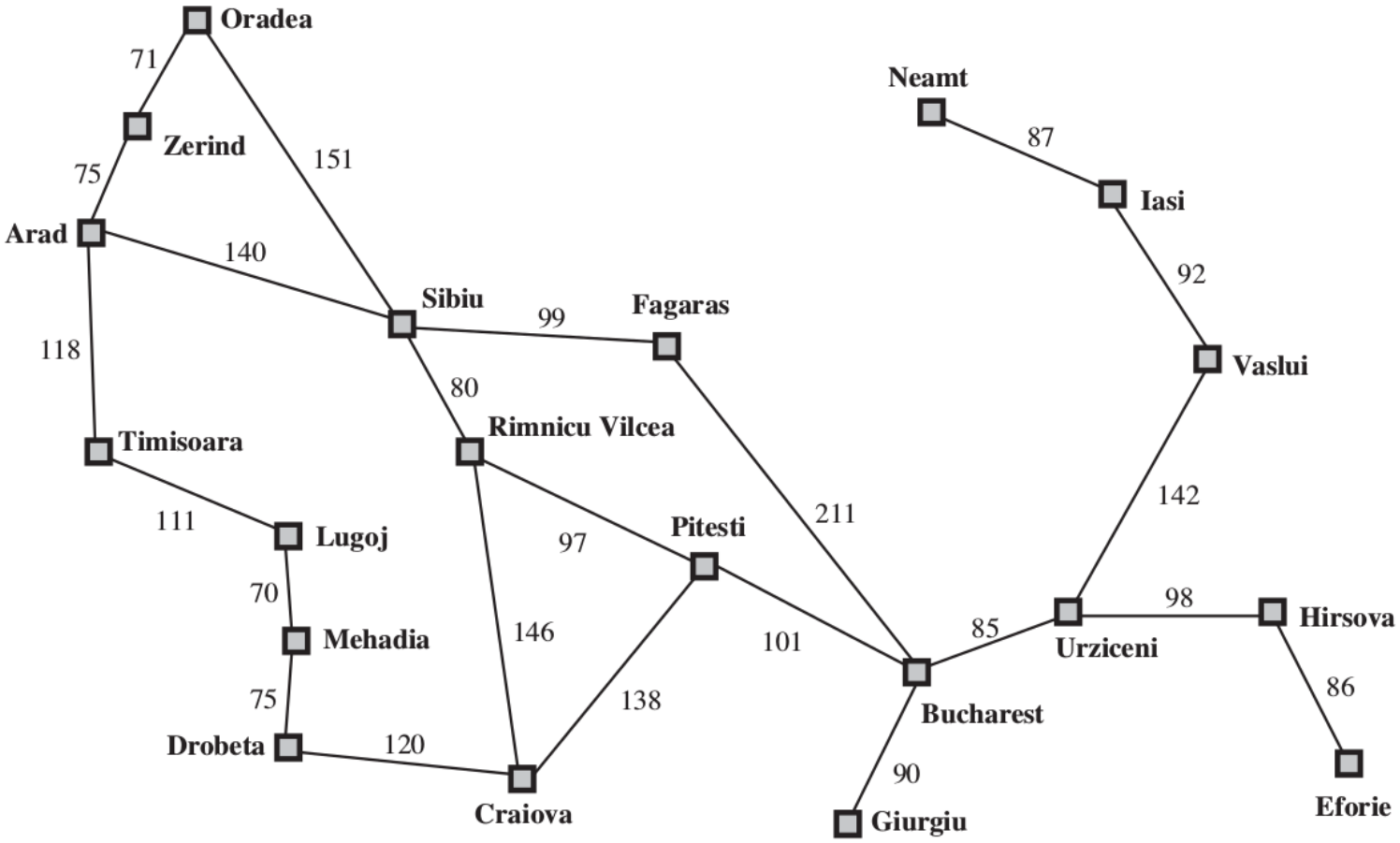
Problem Solving (классический поиск)

- Направление AI, отличающееся следующими чертами:
  - Атомарные представления
  - Хорошо поставленные задачи, детерминированная, дискретная, статическая, наблюдаемая и известная среда.
  - Наличие цели. Цель задаётся как множество состояний.
  - Формулирование задачи - определение, какие состояния и действия рассматривать в контексте цели.
  - Решение задач через поиск в пространстве состояний. Решение - последовательность действий.
  - Использование эвристик для снижения числа рассматриваемых состояний

Постановка задачи

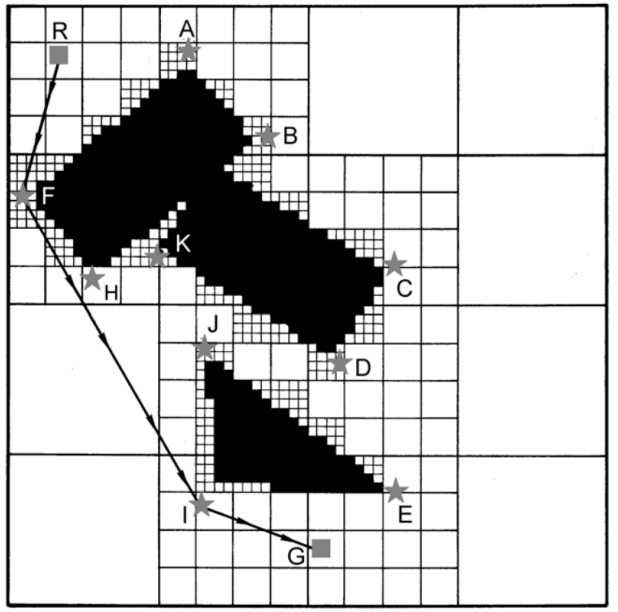
- Задача задаётся через:
  - Начальное состояние
  - Набор действий, доступных агенту
  - Модель переходов. Result(s,a) возвращает состояние, полученное через применение действия a в состоянии s. Описывает суть действий.
  - Критерий достижения целей. Определяется через набор состояний или свойств состояний.
  - Модель стоимостей действий. C(s,a,s') - стоимость перехода из состояния s в состояние s' через действие a. Стоимость целого пути полагается за сумму этих стоимостей.

Примеры задач



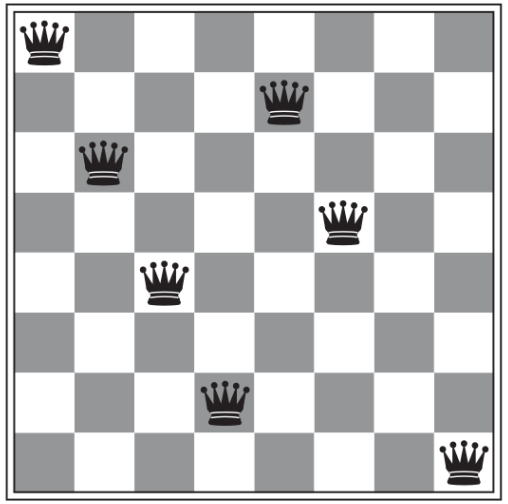
Примеры задач

- Поиск маршрутов
- Проектирование интегральных схем
- Навигация роботов
- Упорядочивание сборочных операций
- Паззлы



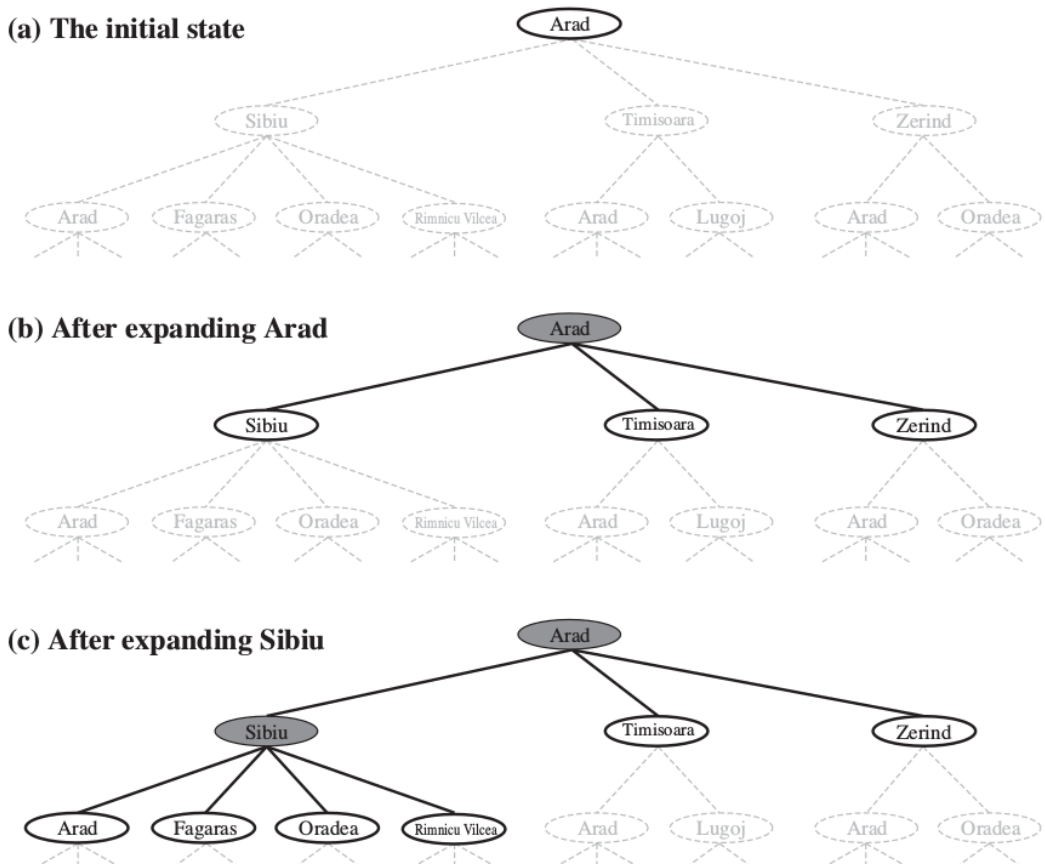
7	2	4
5		6
8	3	1

	1	2
3	4	5
6	7	8



Поиск решений

Последовательности действий, начиная с начального состояния, формируют *дерево поиска*, где узлы соответствуют состояниям, а ветви - действиям.



Поиск решений

function TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

---

function GRAPH-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  initialize the explored set to be empty
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
      only if not in the frontier or explored set

Поиск решений

Стратегии поиска оцениваются с точки зрения 4 критериев:

- Полнота - гарантия найти решение, если оно существует.
- Оптимальность - будет ли полученное решение оптимальным.
- Временная сложность - как долго ищется решение
- Сложность по памяти - как много памяти требуется

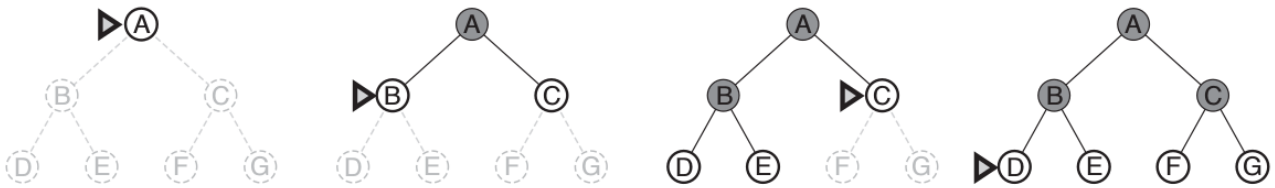
Неинформированный поиск

Стратегии "слепого поиска", не имеющие никакой информации о задаче, кроме её формулировки. Основные стратегии:

- Поиск в глубину (Depth-first search)
- Поиск в ширину (Breadth-first search)
- Итеративное углубление (Iterative deepening)
- Поиск по критерию стоимости (Uniform-cost search)

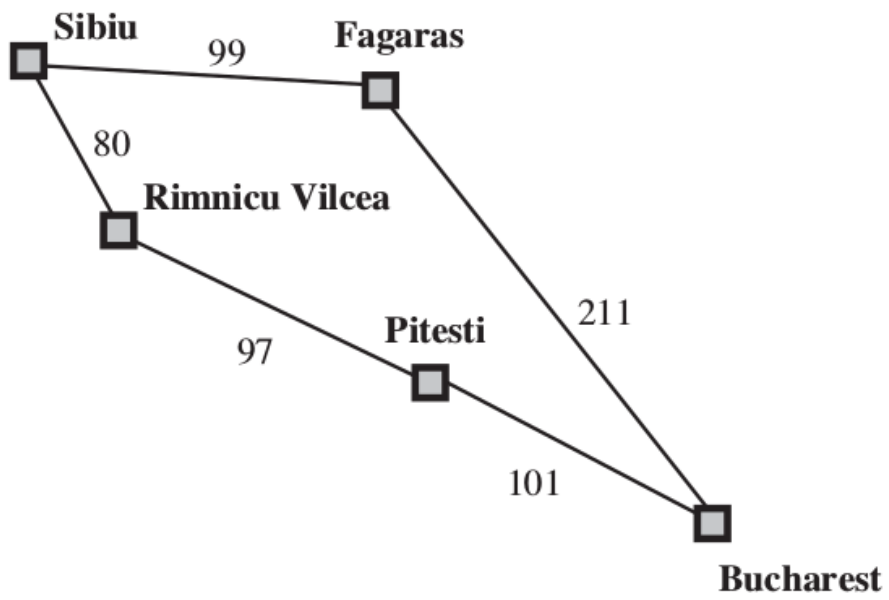
Поиск в ширину

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
    node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    frontier ← a FIFO queue with node as the only element
    explored ← an empty set
    loop do
        if EMPTY?(frontier) then return failure
        node ← POP(frontier)   /* chooses the shallowest node in frontier */
        add node.STATE to explored
        for each action in problem.ACTIONS(node.STATE) do
            child ← CHILD-NODE(problem, node, action)
            if child.STATE is not in explored or frontier then
                if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
                frontier ← INSERT(child, frontier)
```

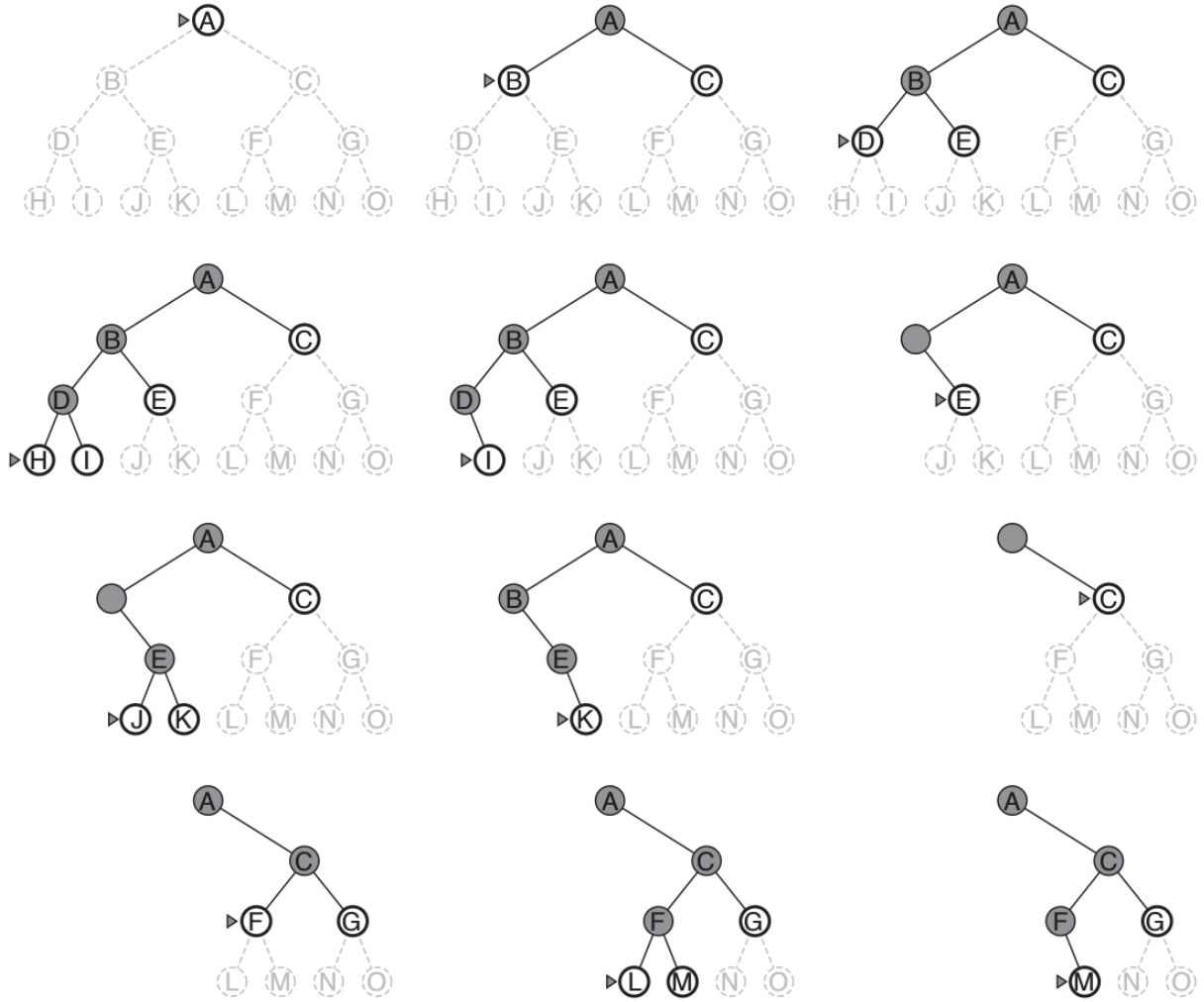


Поиск по критерию стоимости

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)   /* chooses the lowest-cost node in frontier */
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        frontier ← INSERT(child, frontier)
      else if child.STATE is in frontier with higher PATH-COST then
        replace that frontier node with child
```

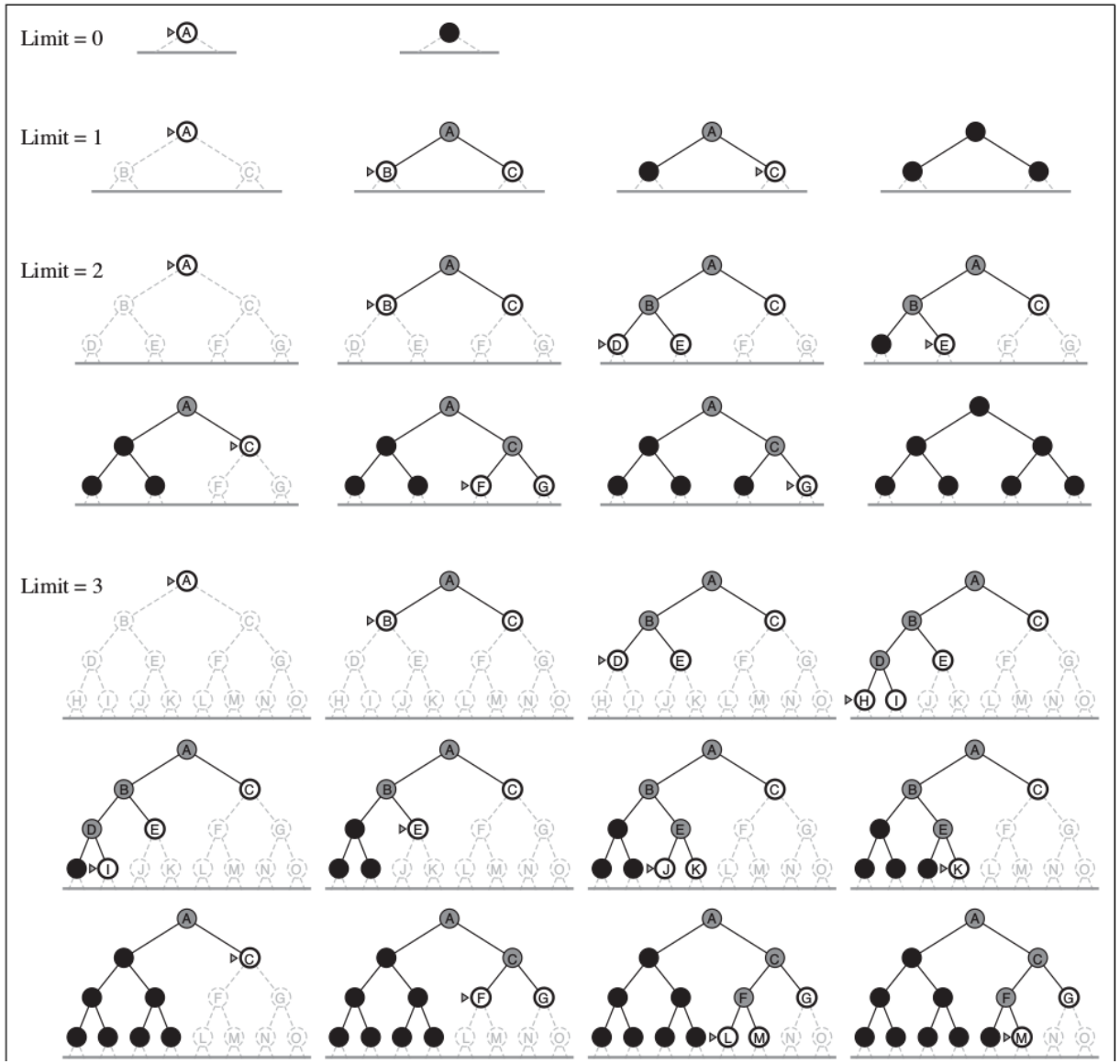


Поиск в глубину

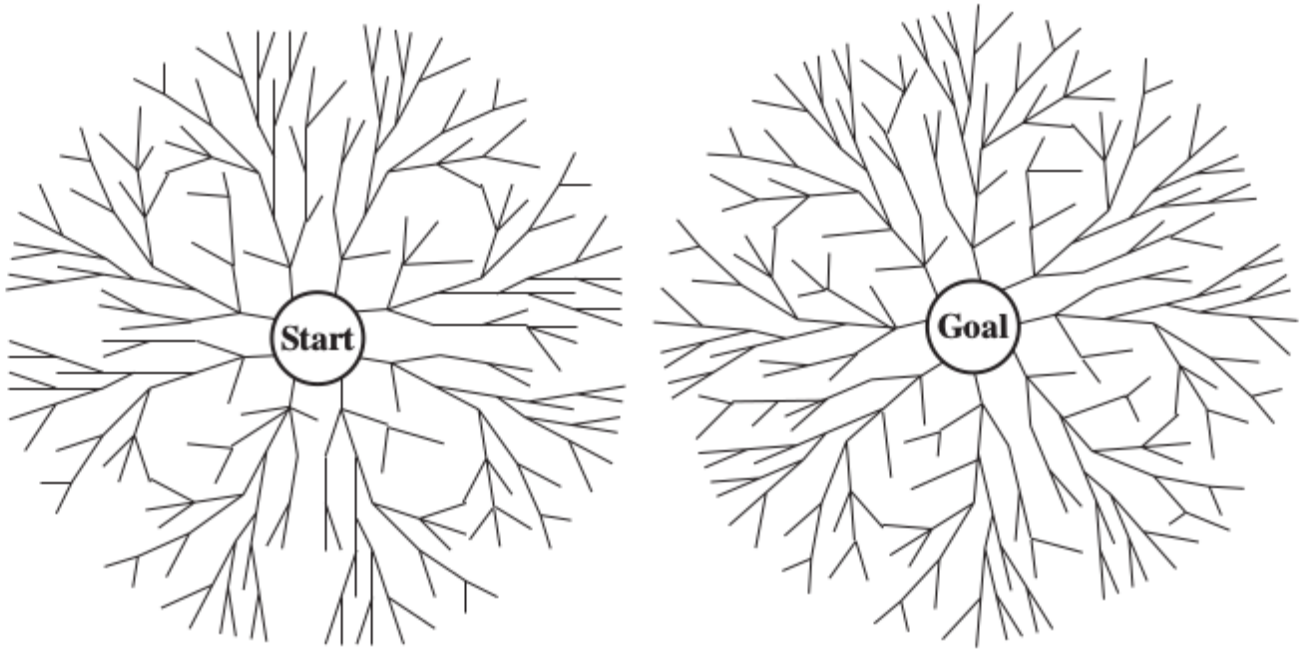


Итеративное углубление

function ITERATIVE-DEEPENING-SEARCH(*problem*) **returns** a solution, or failure
  for *depth* = 0 **to** ∞ **do**
    result $\leftarrow$ DEPTH-LIMITED-SEARCH(*problem*, *depth*)
    if *result* $\neq$ cutoff **then return** *result*



Двусторонний поиск



Информированный поиск

- Выбор следующего узла для раскрытия осуществляется через функцию оценки.
- Основные стратегии: жадный поиск, A* и его модификации.
- Функция оценки включает в себя эвристику - оценку близости к цели.

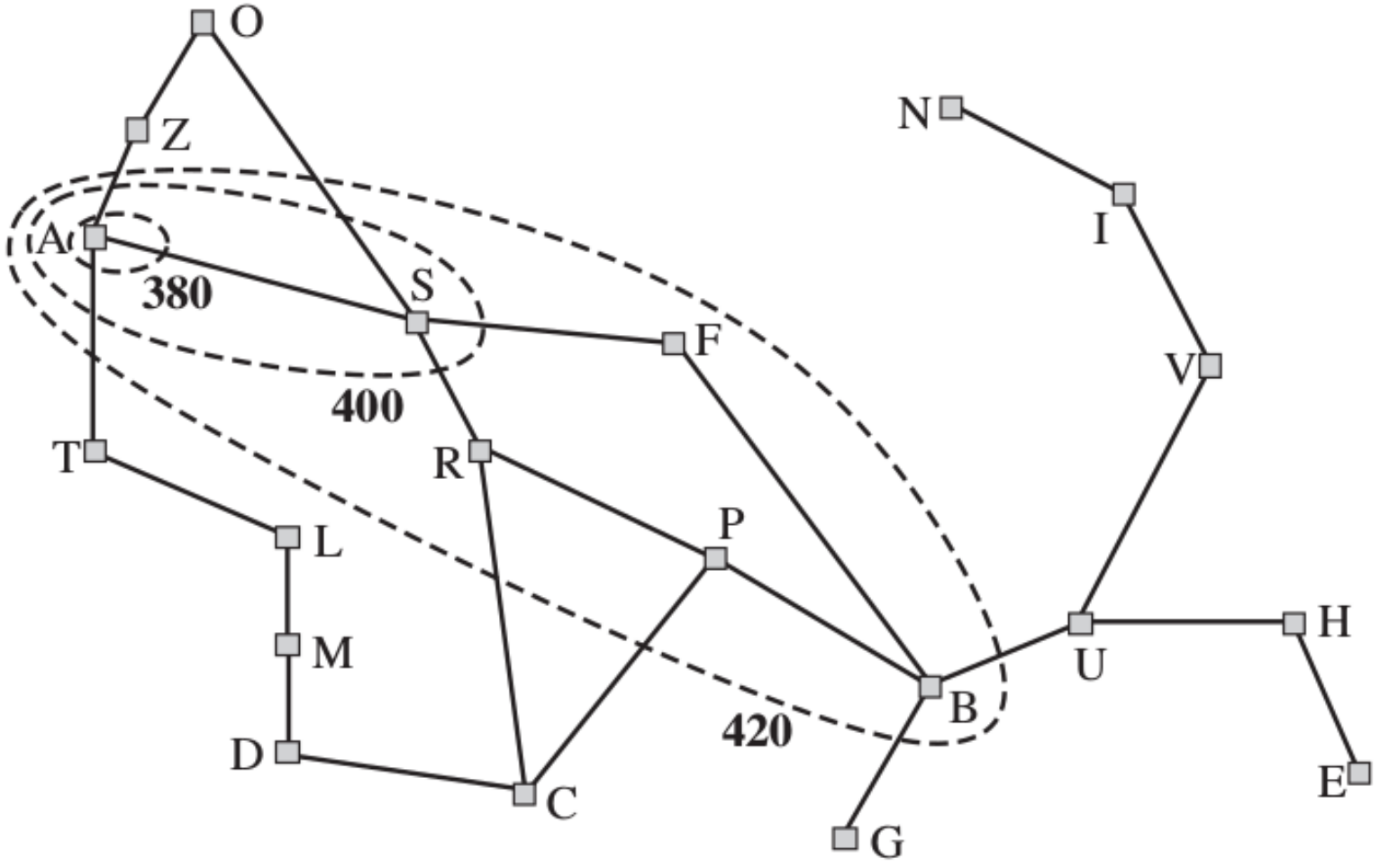
Информированный поиск

- Жадный поиск использует эвристику h(n) для упорядочивания узлов
- A* поиск использует g(n) + h(n), где g(n) - стоимость пути до текущего узла. A* оптимален для допустимых и непротиворечивых эвристик.
- Допустимые эвристики никогда не переоценивает стоимость пути до цели.
- Непротиворечивая эвристика удовлетворяет формуле (неравенство треугольника):

$$h(n) \leq c(n, a, n') + h(n')$$

A*: расширение пространства поиска

A - начальное состояние
B - цель
Эвристика - расстояние по прямой

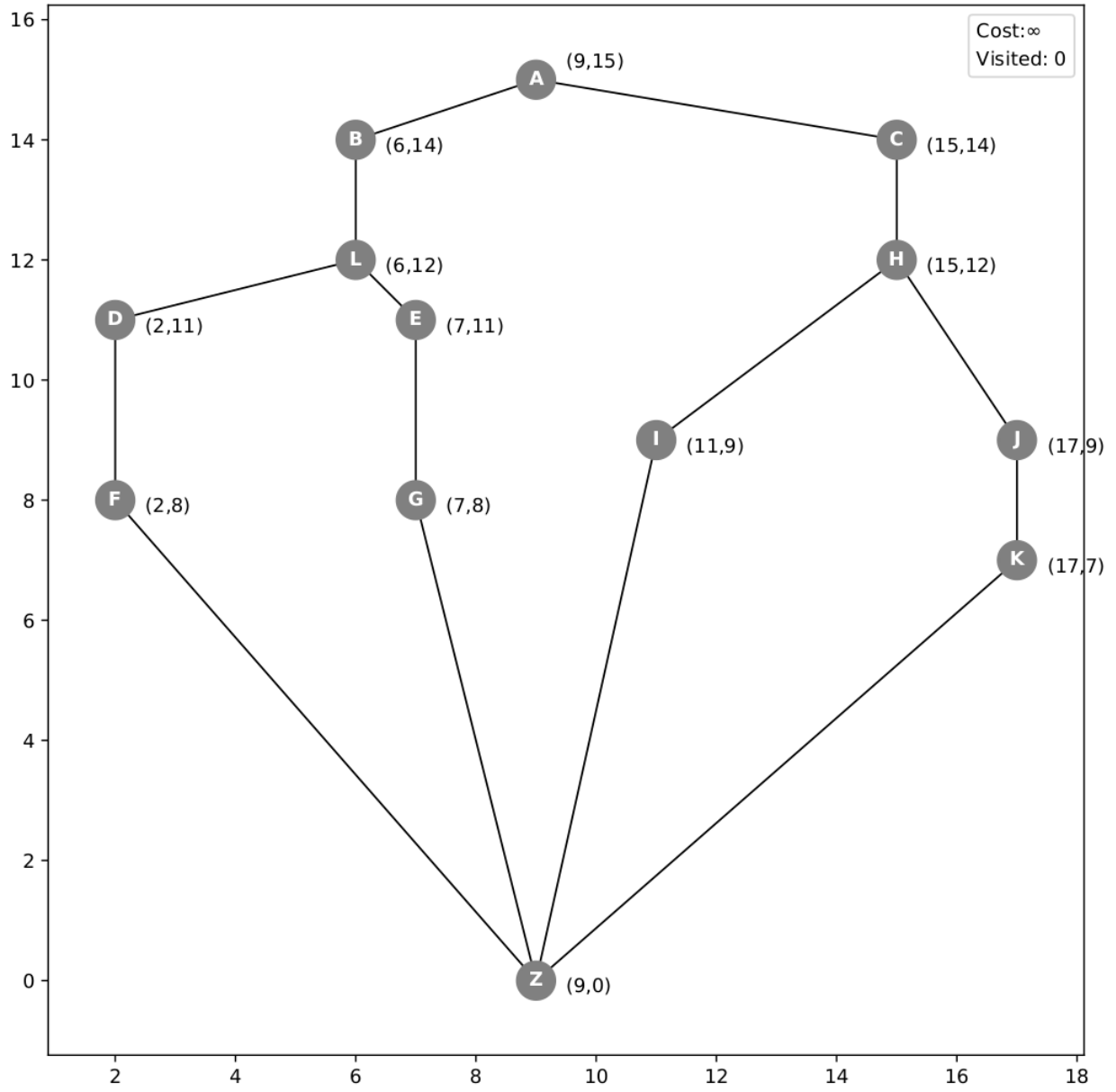


Примеры получения эвристик

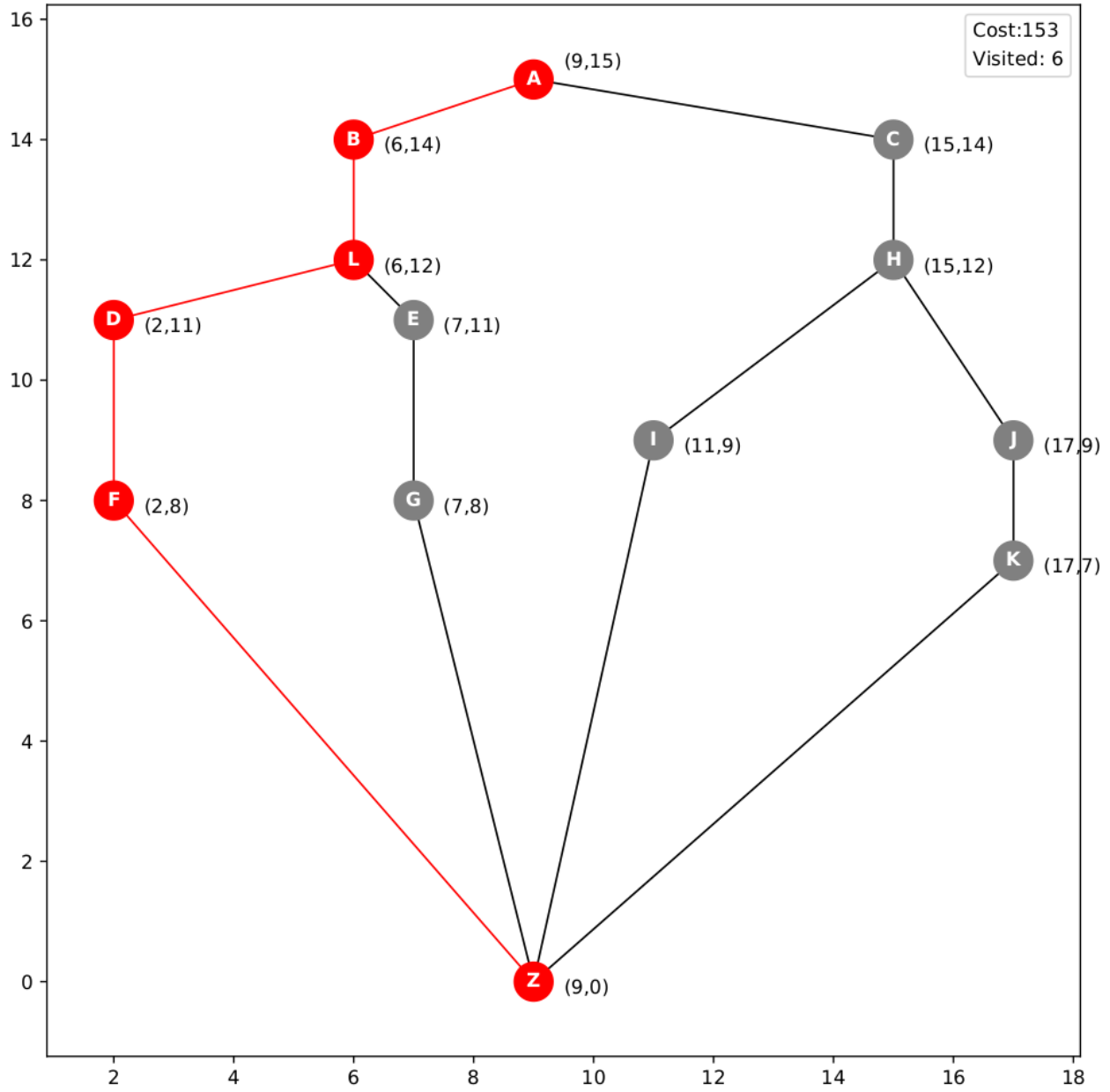
- Упрощённые проблемы
- Базы шаблонов
- Экспертные оценки, выученные эвристики.

Пример: поиск на карте

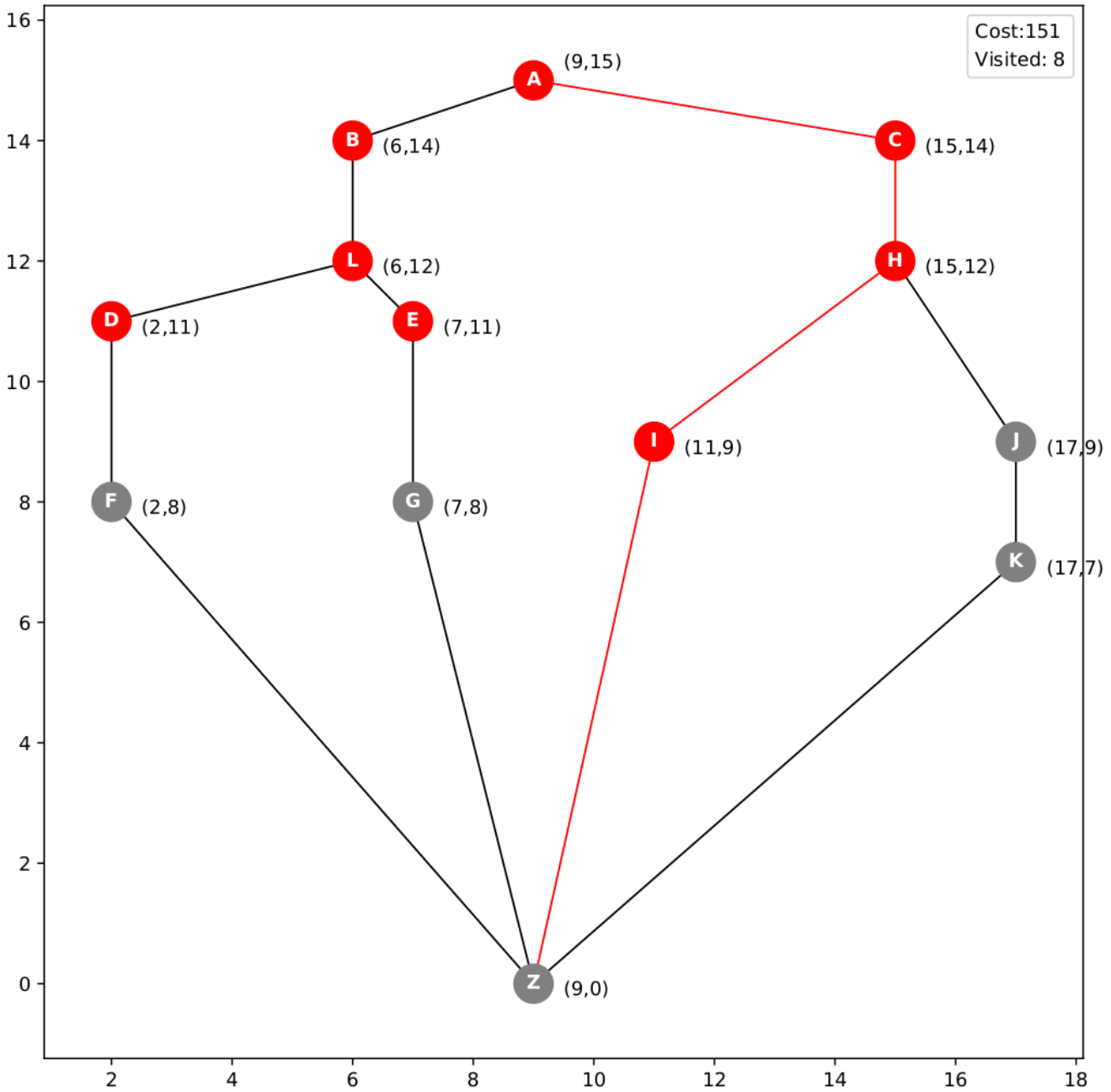
- Начальное состояние - A
- Действия - перемещения между пунктами
- Цель - Z
- Стоимость действий - Евклидово расстояние между точками
- Оптимальное решение: A-B-L-E-G-Z, стоимость 93



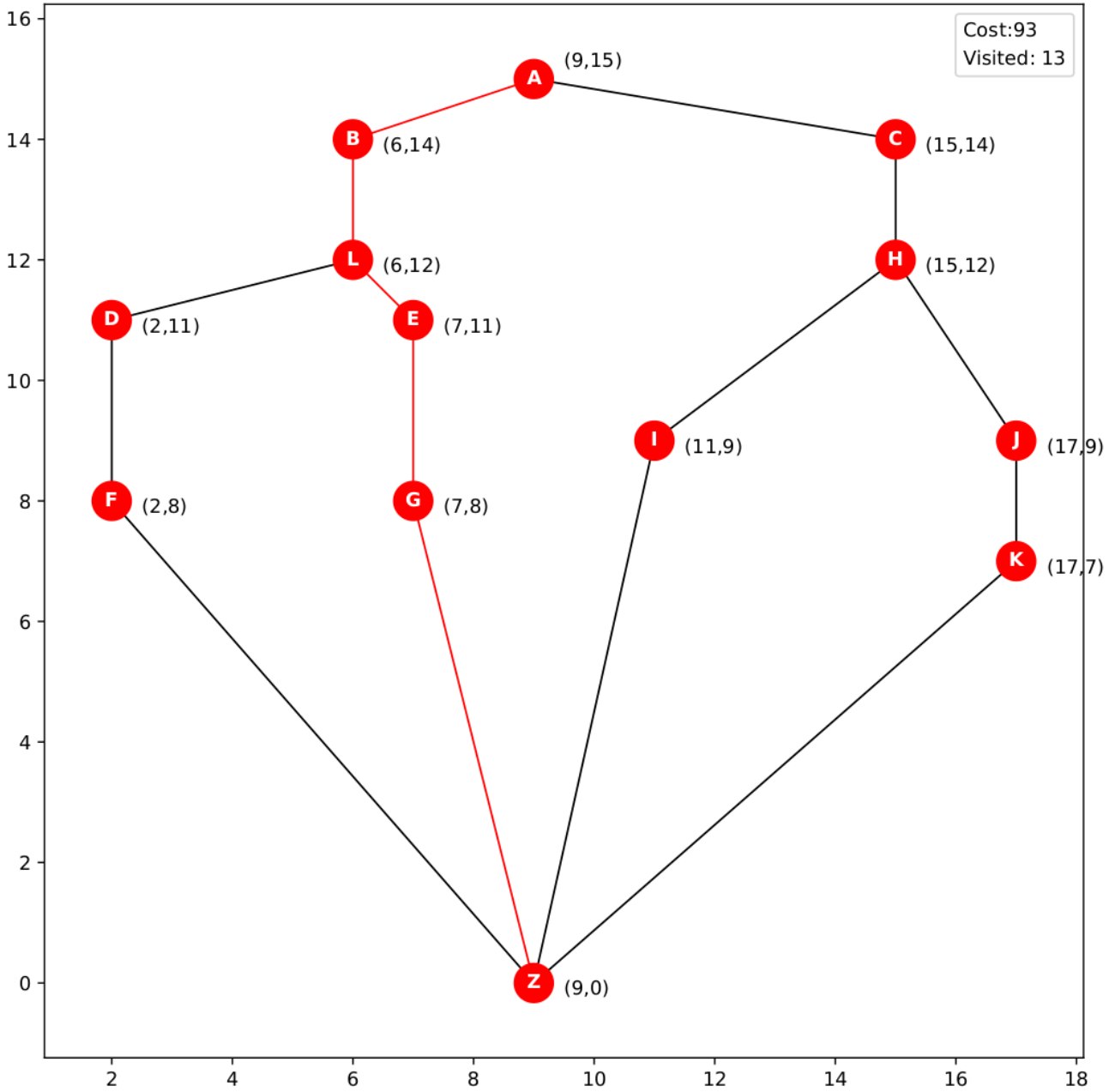
Поиск в глубину



Поиск в ширину



Поиск по критерию стоимости



A* (h(n) - прямое расстояние до цели)