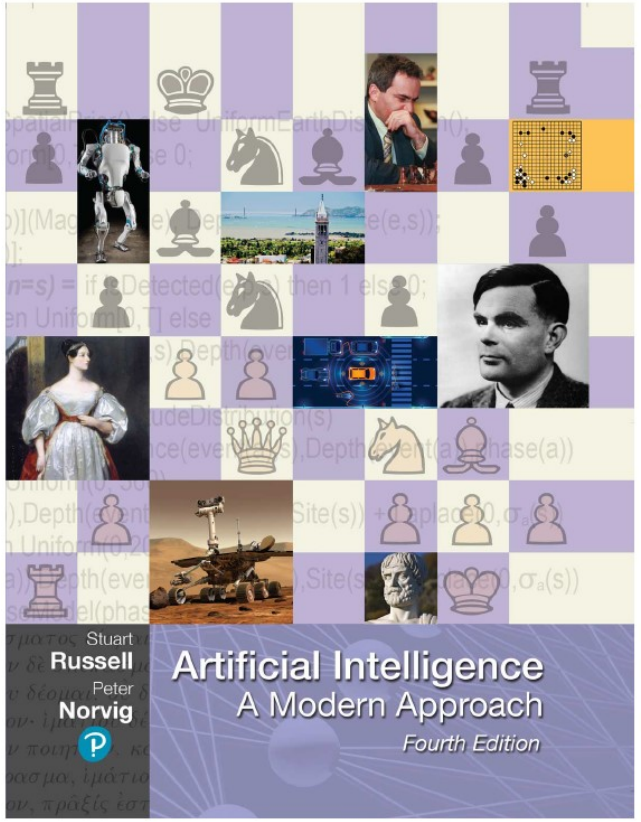
Методы искусственного интеллекта

Лекция 1. Введение в курс. Поиск

О курсе

- Тематически очень разнообразный материал (от поиска до нейронных сетей). Из-за этого где-то поверхностный.
- Критерий включения: имитация деятельности, требующей интеллекта (не следует путать с имитацией самого интеллекта)
- Минимум «философии», максимум прагматики
  - Задача – познакомиться с палитрой методов, которые могут быть полезны при решении широкого круга задач, и дать некоторые ориентиры относительно того, как следует выбирать между ними
- Во многом следуем AIMA (С. Рассел, П. Норвиг Искусственный интеллект: современный подход)
  - 4 издания
  - 1500 учебных заведений по всему миру
  - П. Норвиг – помимо прочего, директор по исследованиям Google



О курсе. Тематический план

- Краткая история ИИ
- Поиск решений. Алгоритмы поиска
- Символьный искусственный интеллект
  - Логические агенты
  - Представление знаний с помощью онтологий
- Решение задач с неопределенностью
  - Нечеткие вычисления
  - Байесовские сети
- Субсимвольные методы искусственного интеллекта. Глубокие нейронные сети
  - Компьютерное зрение
  - Обработка естественного языка
- Обучение с подкреплением

О курсе. Отчетность

- Экзамен (оценка в зачетку)
- Фактически, балльно-рейтинговая система:
  - 50% - практические задания
    - 8 - Поиск
    - 7 - Логические агенты и онтологии
    - 8 - Нечеткое управление
    - 9 - Обработка естественного языка
    - 9 - Компьютерное зрение
    - 9 - Обучение с подкреплением
  - 20 % - семинары (15 доклад и 5 обсуждение)
  - 30 % - экзамен
- Оценка:
  - [85; 100] – отлично
  - [60; 85) – хорошо
  - [40; 60) – удовлетворительно

Тема 1. Краткая история ИИ

Основные вехи ИИ

Годы	Характеристика этапа
1943-1955 гг.	Первая работа (1943 г., МакКаллок и Питтс) – модель искусственного нейрона. Показали, что все логические связки могут быть представлены с помощью таких структур, а также показана способность к обучению. 1950 г. – статья Computing Machinery and Intelligence Алана Тьюринга, в которой описан тест Тьюринга, принципы машинного обучения, генетические алгоритмы и обучение с подкреплением. 1951 г. – Марвин Минский и Дин Эдмондс создают первый сетевой компьютер (40 нейронов).
1956 г.	Дартмутский семинар, Джон Маккарти. Появление самого научного направления.
1952-1969 гг.	**Период больших ожиданий.** Аллен Нюэлл и Герберт Саймон – программа логик-теоретик (Logic Theorist), общий решатель задач (General Problem Solver). Гипотеза ***физической символической системы*** – существует физическая символическая система, которая имеет необходимые и достаточные средства для интеллектуальных действий общего вида. Манипуляции структурами данных, состоящими из символов. Развиваются также и субсимвольные методы – перцептроны, доказательство их сходимости.

Основные вехи ИИ

Годы	Характеристика этапа
1966-1973 гг.	**Столкновение с реальностью.** 1957 г. Герберт Саймон: «через 10 лет компьютер станет чемпионом мира по шахматам и машиной будут доказаны все важные теоремы». Первые попытки более сложных задач (машинный перевод). Результат: 1966 г. «машинный перевод научного текста общего характера не осуществлен и не будет осуществлен в ближайшей перспективе». Сворачивание финансирования многих инициатив. Неразрешимость многих проблем. Сложность алгоритмов (+ неразвитая теория оценки сложности). Принципиальная ограниченность некоторых структур (XOR с помощью перцептрона).
1969-1979 гг.	**Системы, основанные на знаниях.** Экспертные системы – Dendral, Mycin.
С 1980-х гг.	**Превращение в индустрию.**
С 1986 г.	**Возвращение к нейронным сетям.** Переоткрытие обратного распространения ошибки. Джеффри Хинтон

Основные вехи ИИ

Годы	Характеристика этапа
С 1987 г.	**Вероятностные рассуждения и машинное обучение.** Движение от булевой логики в сторону вероятностных моделей, к обучению от «жестко закодированных» правил, валидация на сложных проблемах, а не игрушечных примерах. Широкое распространение бенчмарков (UCI Repository, MNIST, ImageNet и другие). 1988 г. – байесовские сети – эффективный формализм для рассуждений в условиях неопределенности. 1988 г. – Ричард Саттон – формальный базис под обучение с подкреплением.
С 2001 г.	**Эпоха больших данных.**
С 2011 г.	**Эпоха глубокого обучения.** Триумф группы Джеффри Хинтона на ImageNet (2012), AlphaGo. Повышение сложности сетей, требующее специального оборудования.

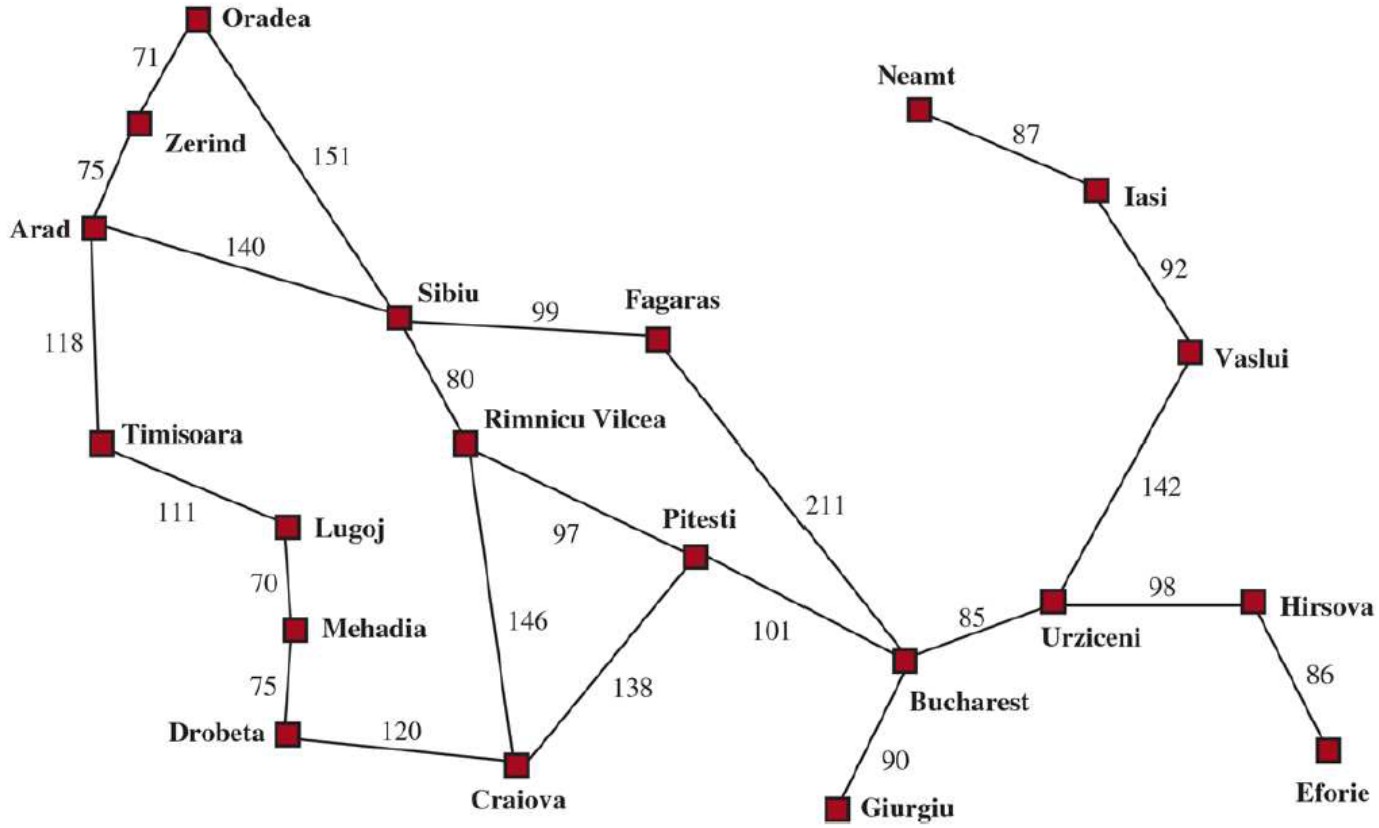
Тема 2. Поиск решений

Решение проблем (задач). Поиск решений

- **Агенты, решающие задачи (problem-solving)** – определяют, что делать, находя последовательность действий, которые приведут к желаемым состояниям (**цели**).
- Компоненты задачи:
  - Множество состояний, в которых может находиться среда (пространство состояний)
  - Начальное состояние
  - Одно или больше целевых состояний (цель)
  - Действия, доступные агенту
  - Модель переходов (изменение состояния в ответ на действие)
  - Функция стоимости действия
- Решение:
  - Путь из начального состояния в одно из целевых
    - Могут отличаться аддитивной стоимостью, один из них оптимальный
- Ограничения и допущения:
  - Среда статическая
  - Среда наблюдаемая
  - Среда детерминированная

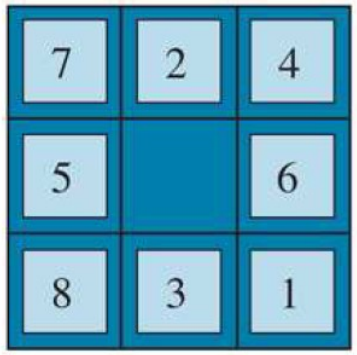
Поиск. Пример 1 – Путь из Арада в Бухарест

- Состояния:
  - Положение агента (город)
- Начальное состояние:
  - Arad
- Цель:
  - Bucharest
- Действия:
  - Перемещение в соседний город по одной из дорог
- Модель перехода:
  - Изменение текущего положения агента
- Стоимость действия:
  - Длина дороги, время и т.п.

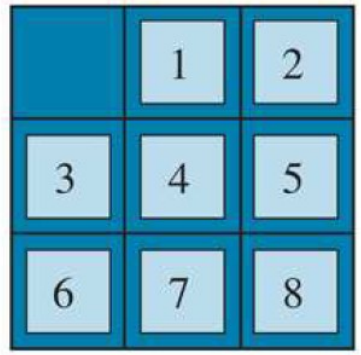


Поиск. Пример 2 – Игра «8»

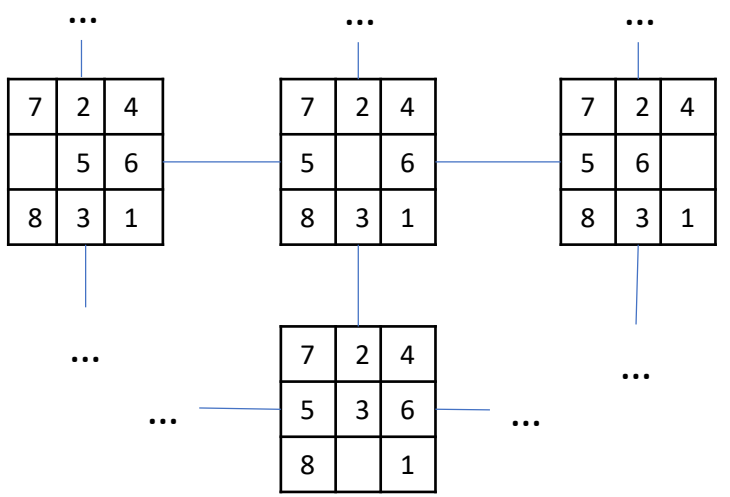
- Состояния:
  - Размещение фишек с номерами 1-8 на поле 3x3
- Начальное состояние
- Цель:
  - Размещение фишек по порядку
- Действия:
  - Перемещение одной фишки на пустую клетку
- Модель перехода:
  - Отображение состояния и действия в новое состояние
- Стоимость действия:
  - Всегда 1



Начальное состояние



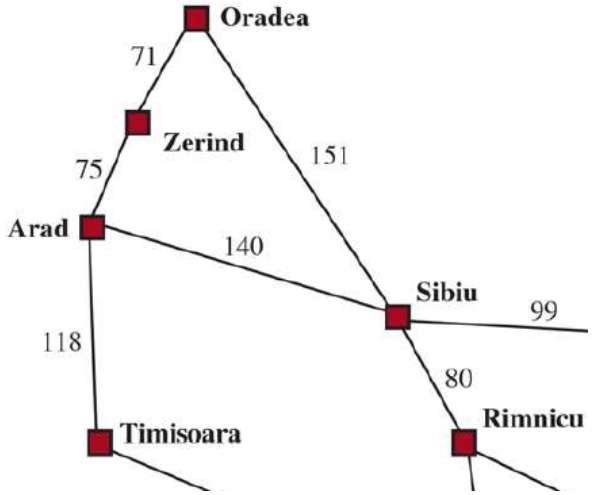
Цель

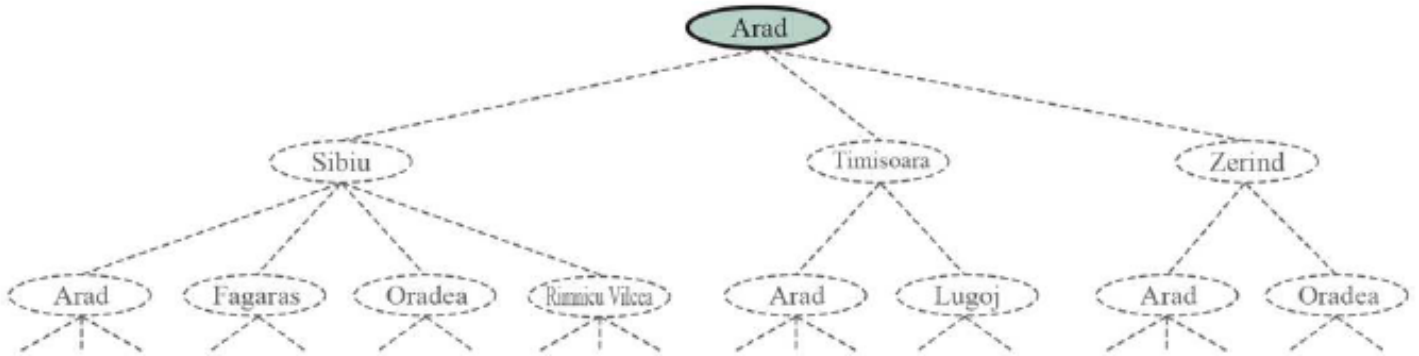


Но не всегда возможно в явном виде!

Алгоритмы поиска решений

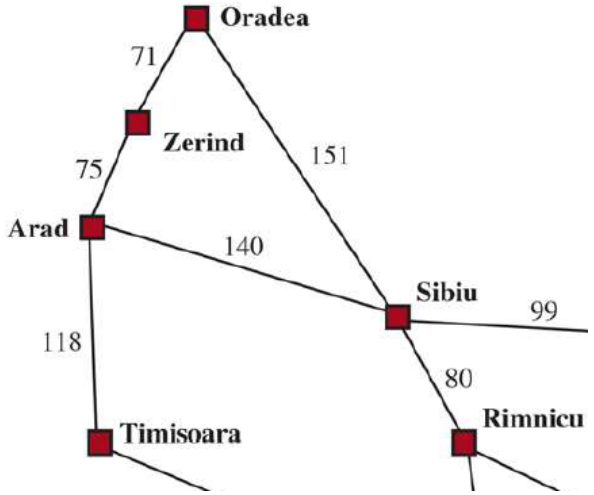
- Вход: задача поиска
- Выход:
  - Решение: последовательность действий
  - Сигнал о том, что решения нет
- Дерево поиска
  - Узлы – состояния
  - Переходы – действия
  - Но не тождественно графу состояний!
    - Может быть много узлов для состояния
    - У узла один родитель

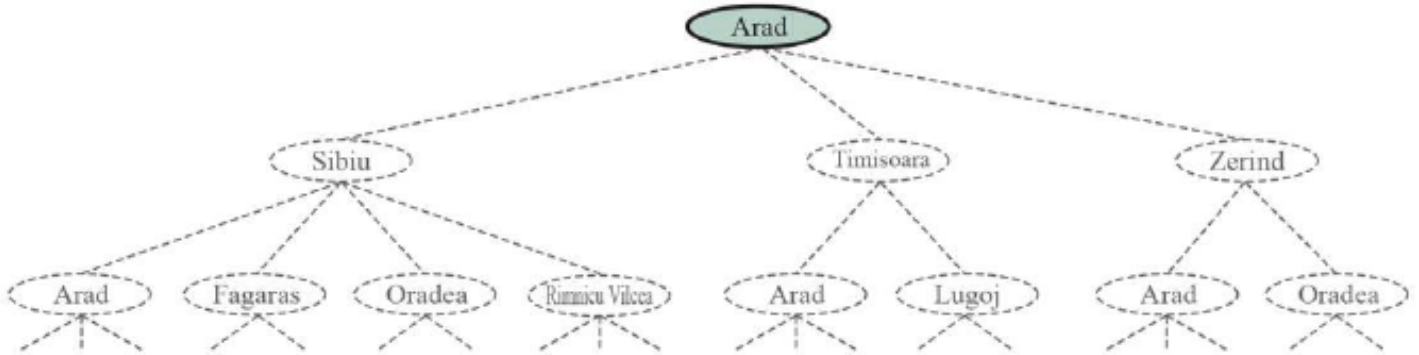




Алгоритмы поиска решений

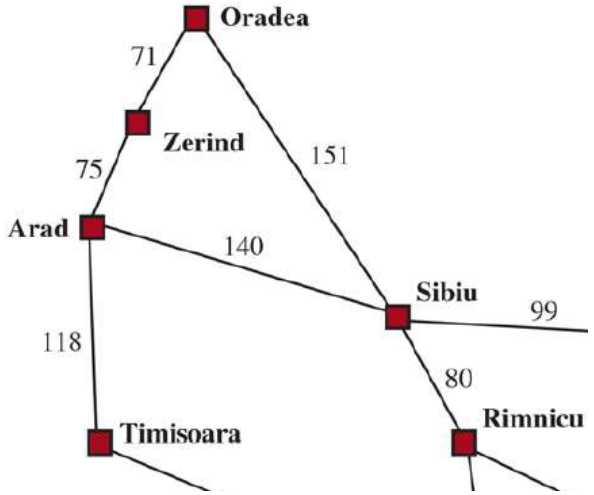
- Развертывание (expand) узла:
  - Получение новых *узлов* применением *действий*, доступных в *состоянии*, которому соответствует *узел*
- Родительский узел
- Дочерние узлы

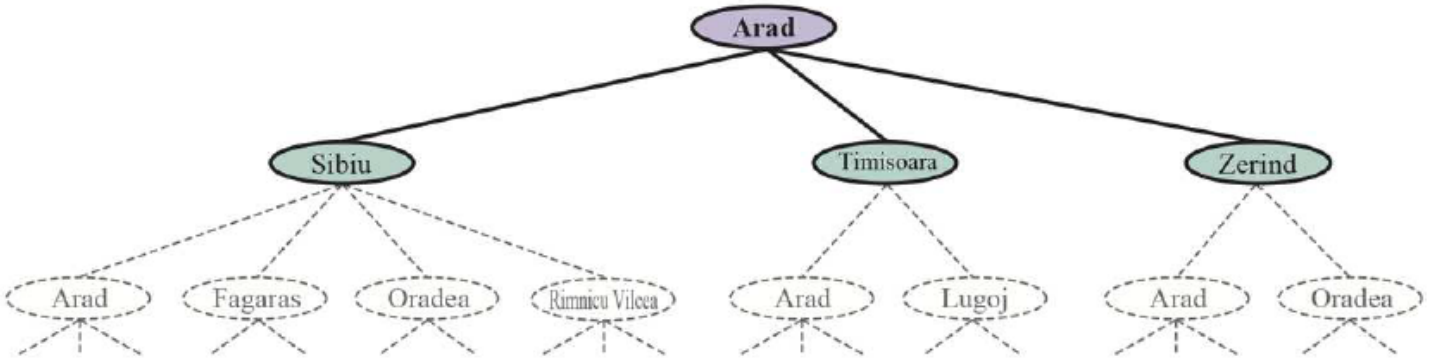




Алгоритмы поиска решений

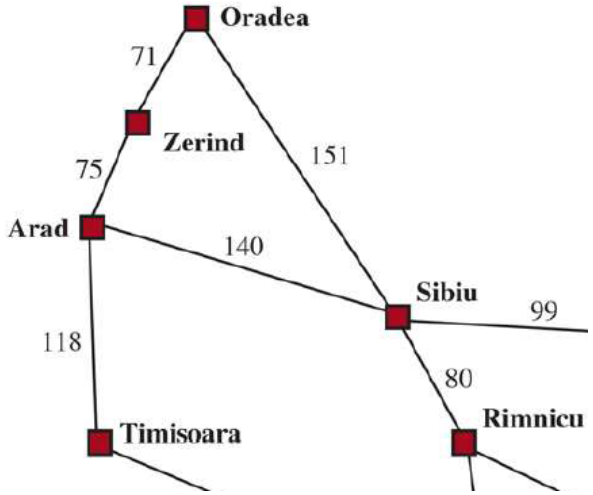
- Развертывание (expand) узла:
  - Получение новых *узлов* применением *действий*, доступных в *состоянии*, которому соответствует *узел*
- Родительский узел
- Дочерние узлы

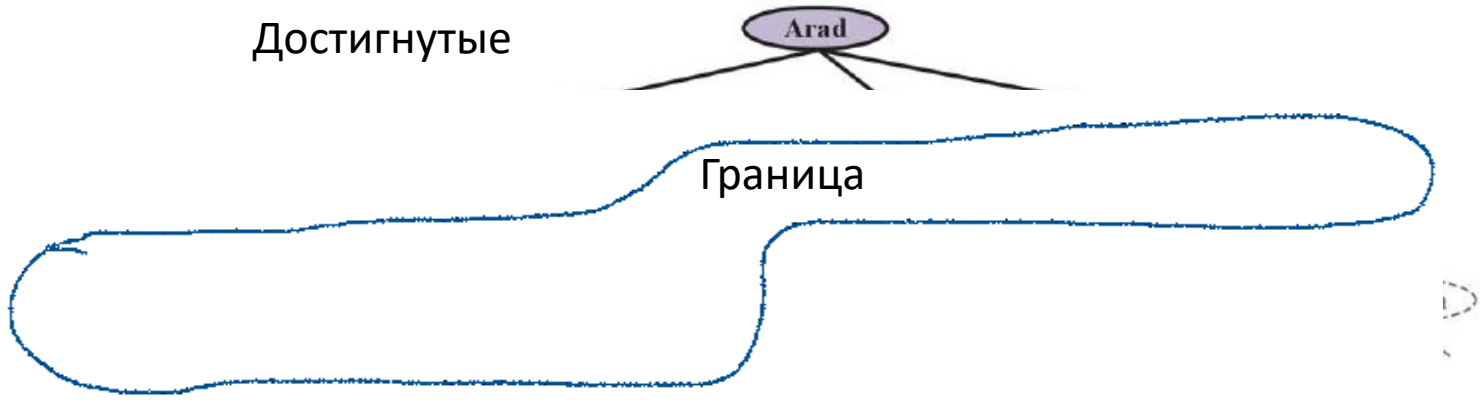




Алгоритмы поиска решений

- Развертывание (expand) узла:
  - Получение новых *узлов* применением *действий*, доступных в *состоянии*, которому соответствует *узел*
- Родительский узел
- Дочерние узлы
- Выбор одного из узлов
- Граница дерева поиска
  - Периферия, фронт
- Достигнутые состояния





Поиск по лучшему совпадению (best first search)

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure

function EXPAND(problem, node) yields nodes
  s ← node.STATE
  for each action in problem.ACTIONS(s) do
    s' ← problem.RESULT(s, action)
    cost ← node.PATH-COST + problem.ACTION-COST(s, action, s')
    yield NODE(STATE=s', PARENT=node, ACTION=action, PATH-COST=cost)
```

Состояние –> Узел дерева

Node:

State – состояние

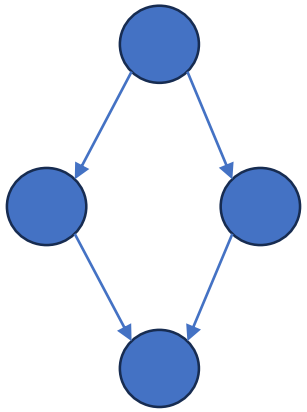
Parent – узел-родитель

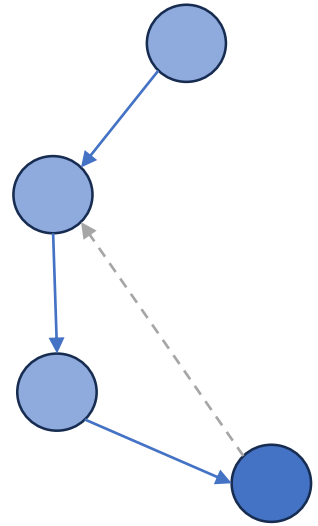
Action – действие, в результате которого узел создан

Path-Cost – общая стоимость пути от исходного состояния

Обработка посещенных состояний

- Запоминать:
  - НО: их может быть очень много
- Не запоминать:
  - НО: можно постоянно посещать одни и те же
- Находить циклы





Характеристики алгоритмов поиска

- Полнота. Гарантируется ли, что алгоритм найдет решение, если оно есть и сообщит в том случае, если его нет?
  - Чтобы быть полным, должен *систематически* исследовать пространство состояний, достижимых из начального
- Оптимальность (по стоимости). Находит ли алгоритм лучшее решение по критерию стоимости?
- Временн*а*я сложность.
  - Количество узлов, создаваемых в процессе поиска
- Пространственная сложность.
  - Максимальное количество узлов, хранимых в памяти

Относительно сложности задачи:
d – количество действий в оптимальном решении
m – максимальное количество действий в любом пути
b – степень ветвления (количество потомков узла)

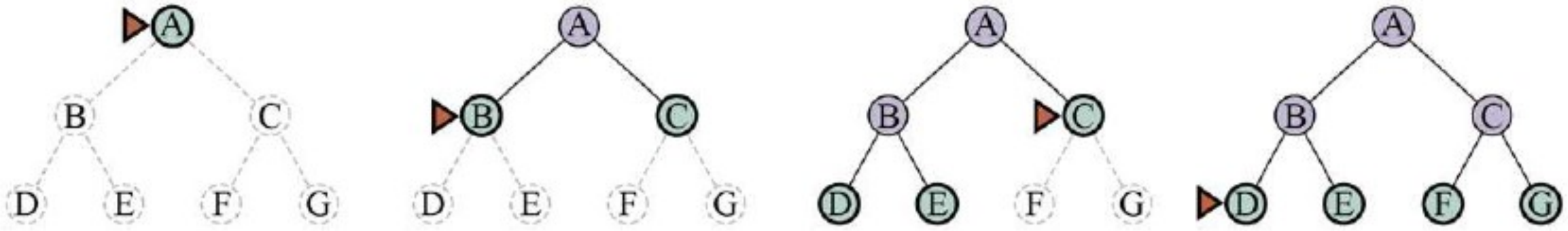
Стратегии неинформированного поиска

Нет информации о том, как близко *текущее состояние* к *целевому*

- Поиск в ширину
- Алгоритм Дейкстры (поиск по критерию стоимости)
- Поиск в глубину

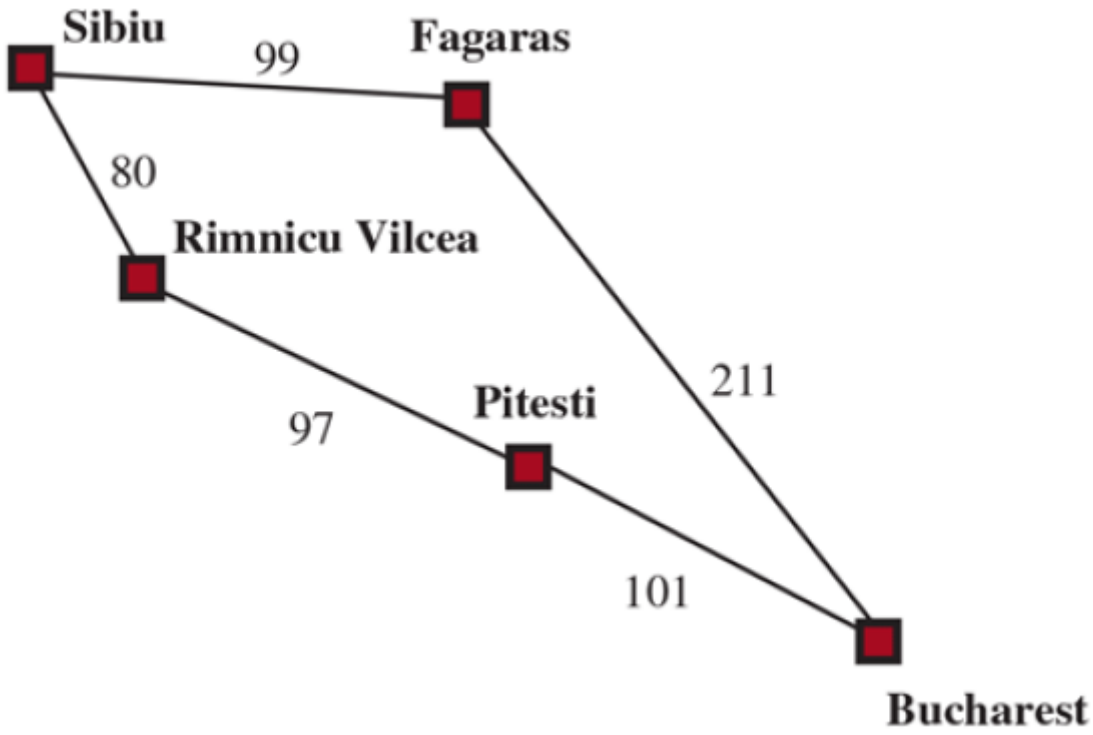
Поиск в ширину

- Best-First-Search, в котором $f(x)$ – глубина узла (количество действий, за которые узел достижим из начального состояния)
- Можно лучше:
  - Замена очереди с приоритетами на обычную
  - Замена поздней проверки цели на раннюю
  - reached: множество, а не словарь



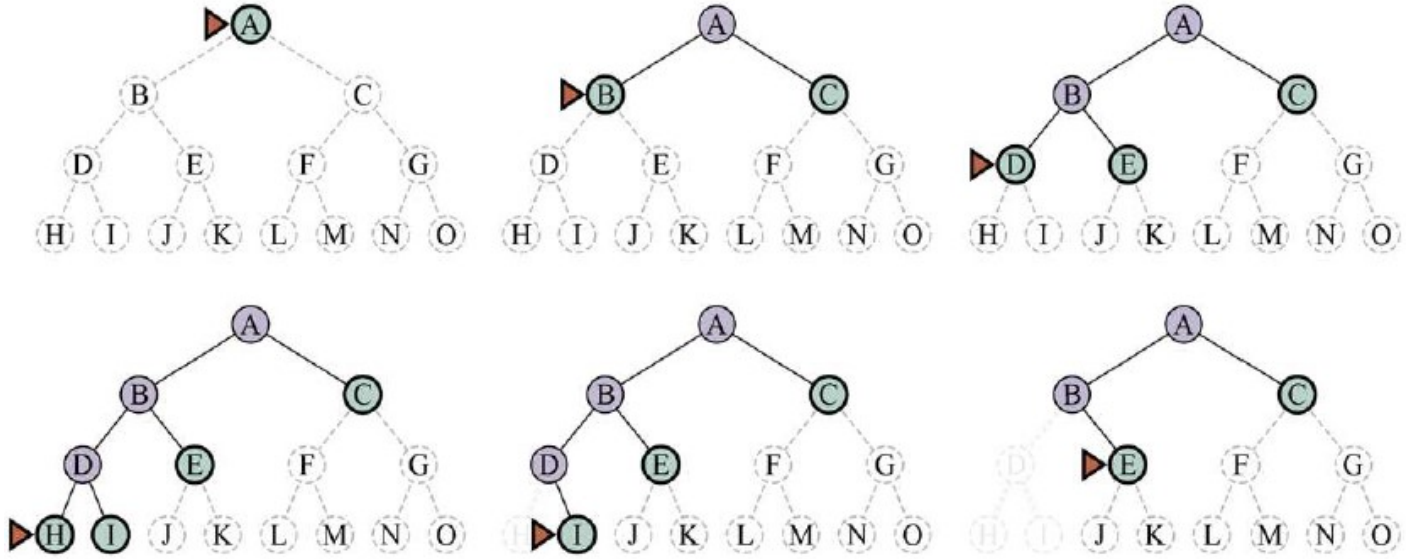
Алгоритм Дейкстры

- Best-First-Search, в котором $f(x)$ – суммарная стоимость (Path-Cost)



Поиск в глубину

- Best-First-Search, в котором $f(x)$ – глубина узла (количество действий, за которые узел достижим из начального состояния) с **обратным знаком**
- Можно лучше:
  - Как поиск в дереве (без отслеживания посещенных состояний)



Поиск в глубину

- **Не** оптимальный по стоимости.
- Полнота:
  - Конечные древовидные пространства состояний: полный
  - Конечные ациклические пространства состояний: полный, но может посещать одни и те же состояния много раз
  - Конечные циклические пространства состояний: требует проверки на циклы для обеспечения полноты
  - Бесконечные пространства: не полный
- Но:
  - Память! – не запоминает посещенные состояния, крошечный фронт

Поиск в глубину. Итеративное углубление

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution node or failure
  for depth = 0 to ∞ do
    result ← DEPTH-LIMITED-SEARCH(problem, depth)
    if result ≠ cutoff then return result

function DEPTH-LIMITED-SEARCH(problem, ℓ) returns a node or failure or cutoff
  frontier ← a LIFO queue (stack) with NODE(problem.INITIAL) as an element
  result ← failure
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    if DEPTH(node) > ℓ then
      result ← cutoff
    else if not IS-CYCLE(node) do
      for each child in EXPAND(problem, node) do
        add child to frontier
  return result
```

Неинформированный поиск. Сравнение

Критерий	Поиск в ширину	Алгоритм Дейкстры	Поиск в глубину	Поиск в глубину (с ограничением глубины)	Поиск в глубину с итеративным углублением
Полнота	Да	Да	Нет	Да	Да
Оптимальность по стоимости	Да	Да	Нет	Нет	Да
Время	$O(b^d)$	$O(b^{1+\lfloor C*/\epsilon \rfloor})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Пространство	$O(b^d)$	$O(b^{1+\lfloor C*/\epsilon \rfloor})$	$O(bm)$	$O(bl)$	$O(bd)$

d – количество действий в оптимальном решении
m – максимальное количество действий в любом пути
b – степень ветвления (количество потомков узла)
*C** – стоимость оптимального решения
ε – минимальная стоимость действия
l – глубина поиска (для алгоритма с ограничением глубины)

Стратегии информированного (эвристического) поиска

Кроме определения самой задачи используются знания (относящиеся к данной предметной области), позволяющие повысить эффективность нахождения решения

- Эвристика задается в виде функции $h(n)$ – оценка стоимости кратчайшего пути из состояния, соответствующего узлу дерева n, до целевого состояния

Алгоритм A*

A* = Best-First-Search, в котором функция оценки f(n):

$$f(n) = g(n) + h(n)$$

$g(n)$ – стоимость пути от начального состояния до n

$h(n)$ – эвристическая оценка пути наименьшей стоимости от n до цели

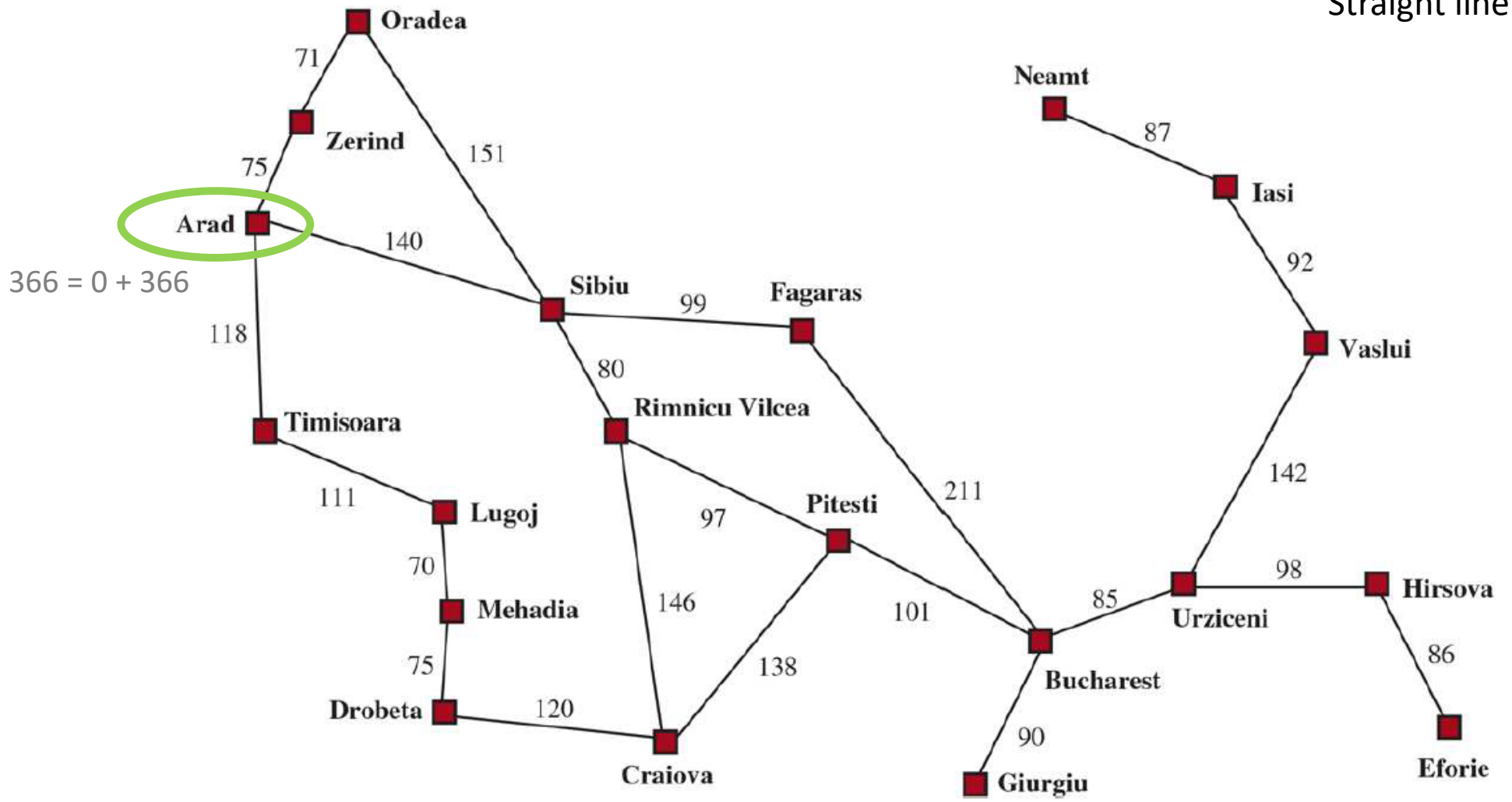
Следовательно:

$f(n)$ – оценка стоимости наилучшего пути от начального состояния до целевого через n

Алгоритм A*

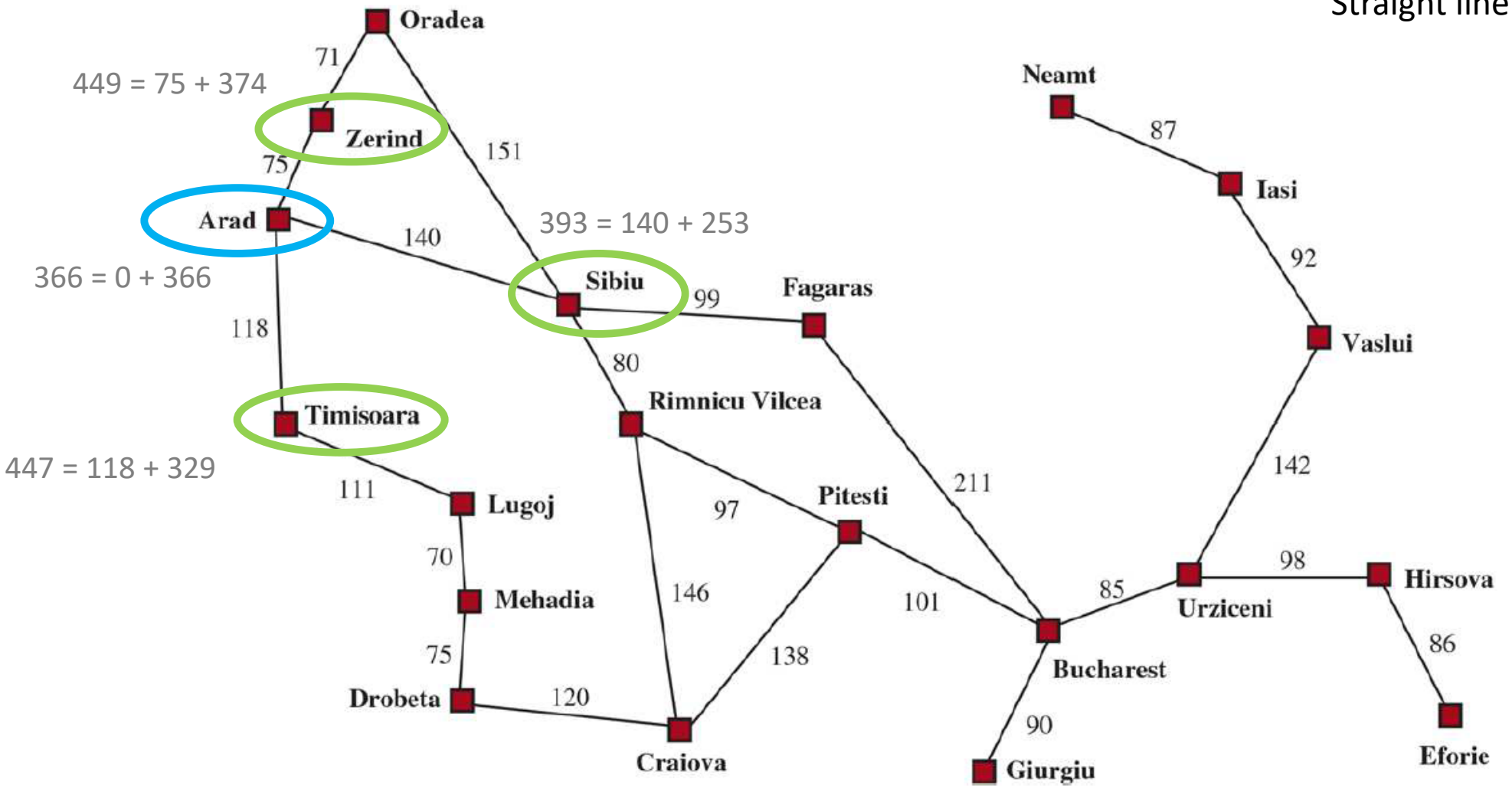
$h(n) = h_{SLD}(n)$
Straight line distance

366 = 0 + 366

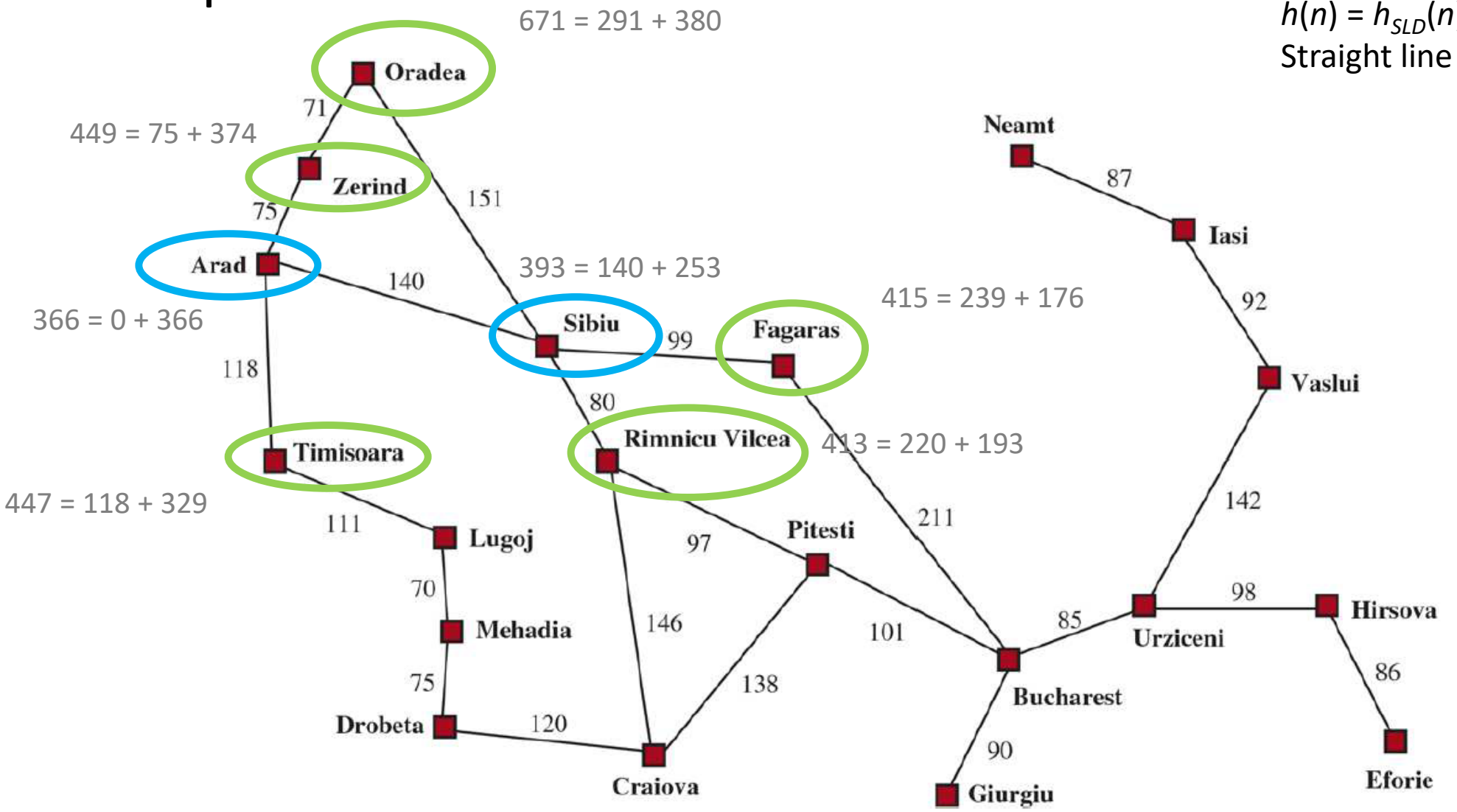


Алгоритм A*

$h(n) = h_{SLD}(n)$
Straight line distance



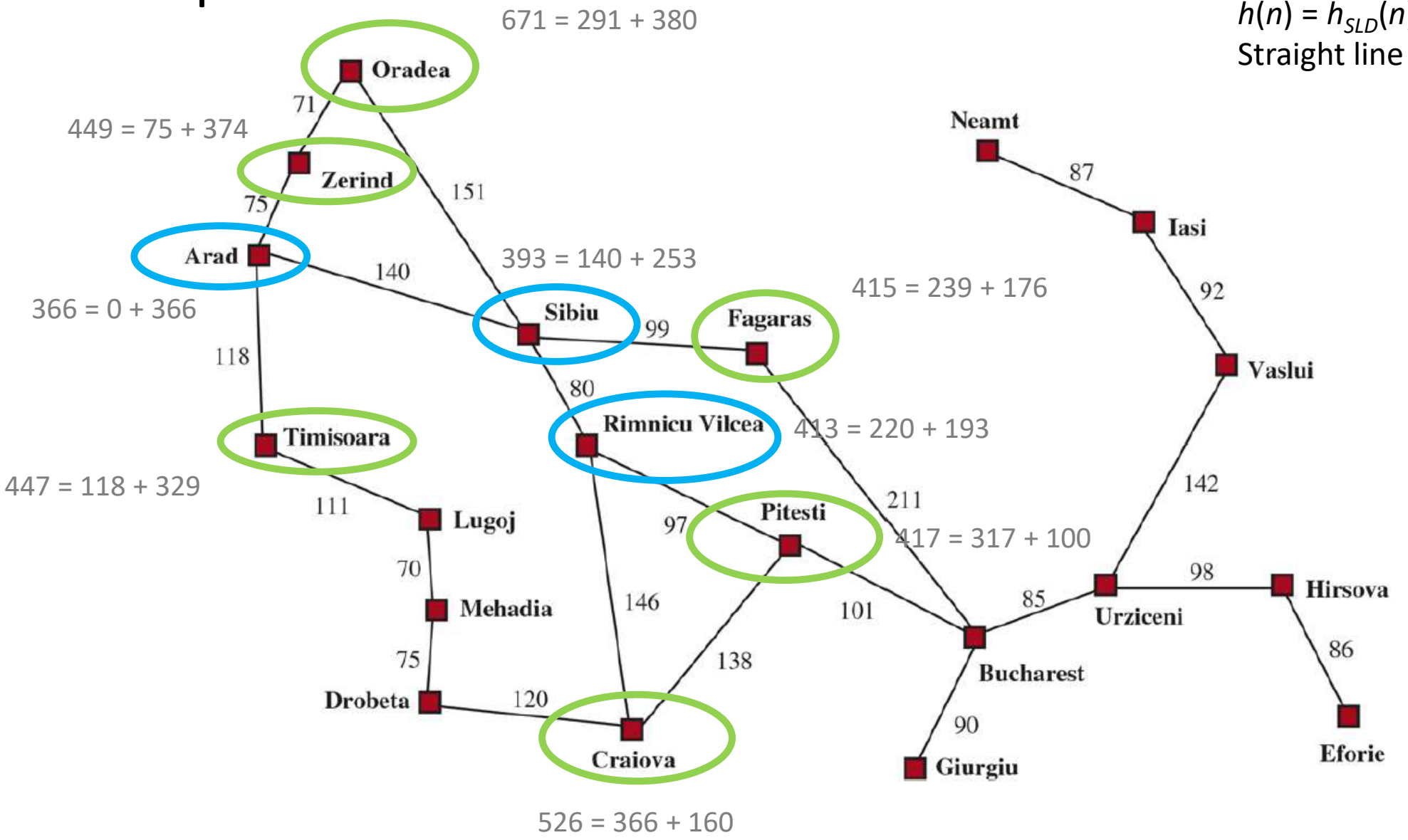
Алгоритм A*



$h(n) = h_{SLD}(n)$
Straight line distance

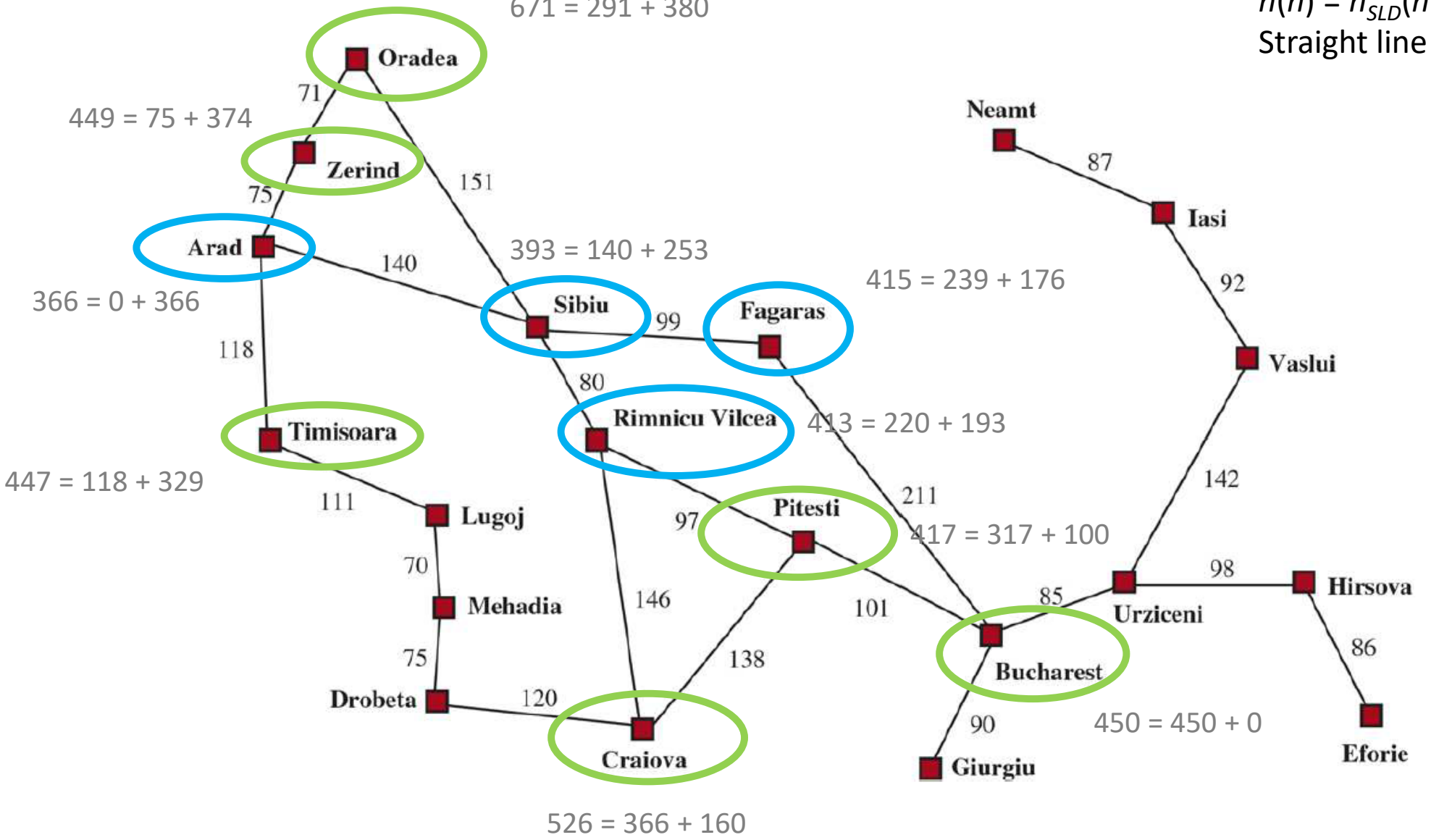
Алгоритм A*

$h(n) = h_{SLD}(n)$
Straight line distance



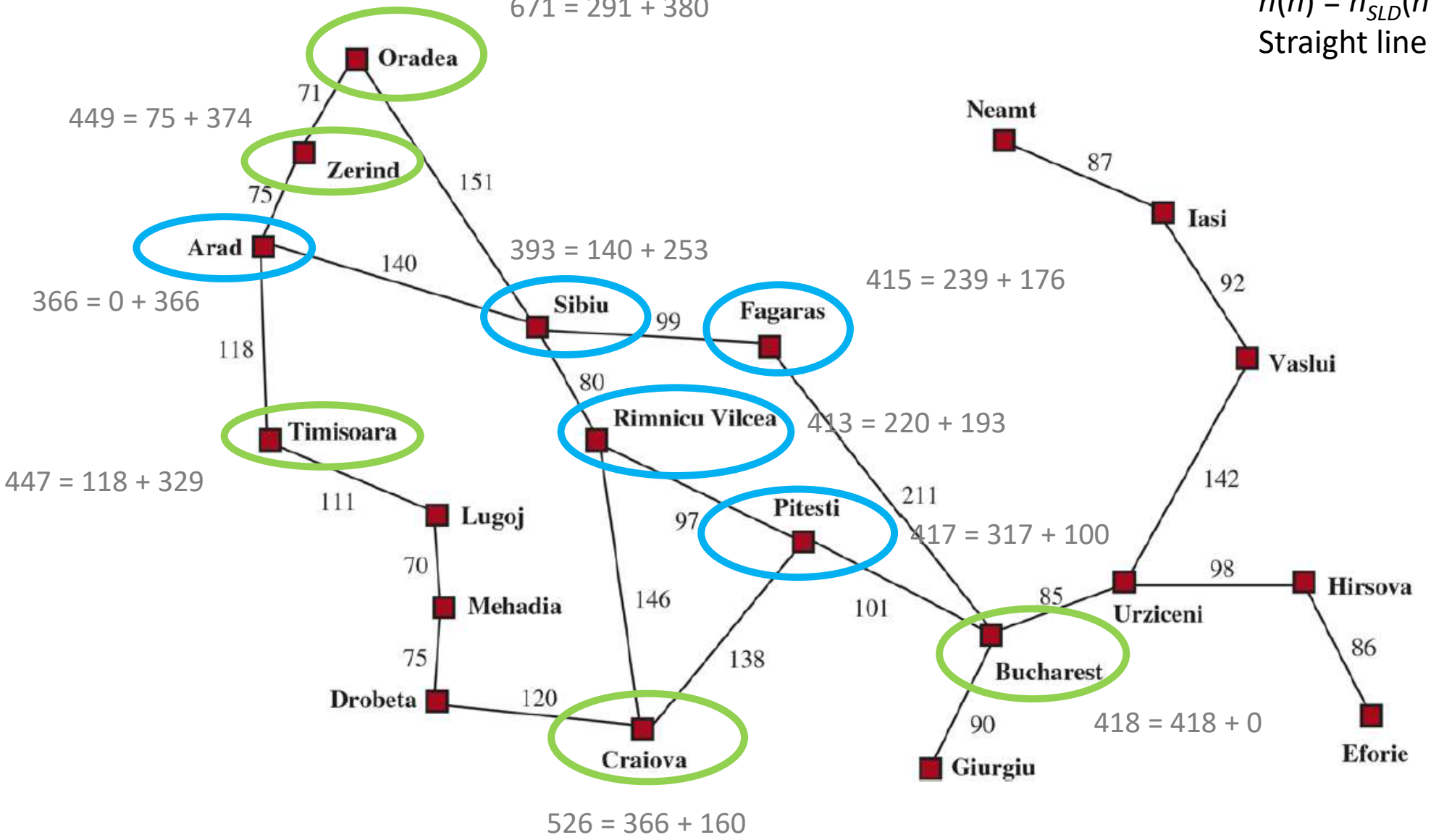
Алгоритм A*

$h(n) = h_{SLD}(n)$
Straight line distance



Алгоритм A*

$h(n) = h_{SLD}(n)$
Straight line distance



Алгоритм A*

Свойства A*:

- Полный
- Оптимальный, при использовании *допустимых* эвристик

Свойства эвристик:

- **Допустимость** (**admissibility**). Допустимые эвристики *оптимистичны* – они никогда не переоценивают стоимость достижения цели.
- **Монотонность, преемственность** (**consistency**). Для любого узла *n* и для любого преемника *n'* узла *n*, сформированного в результате действия *a*, оценка стоимости достижения цели из узла *n* не больше, чем стоимость достижения узла *n'* плюс оценка стоимости достижения цели из *n'*:

$$h(n) <= c(n, a, n') + h(n')$$

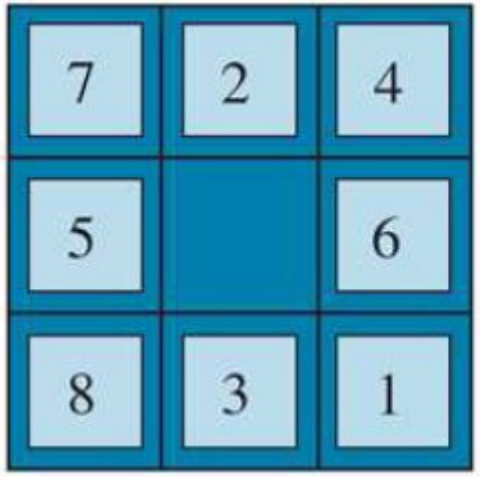
Алгоритм A*

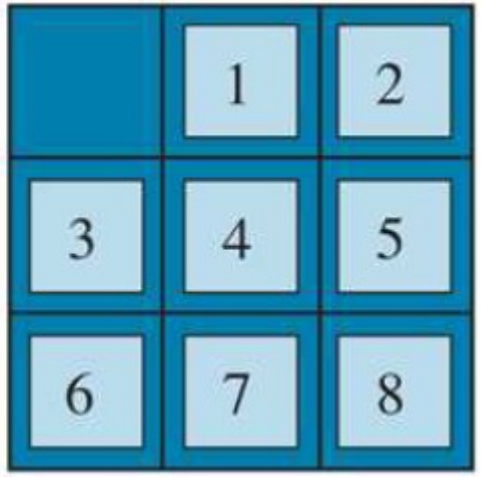
Основная проблема: все еще развертывает очень много вершин (хотя меньше, чем алгоритм Дейкстры, например!).

Методы борьбы:

- Использование недопустимых эвристик (нарушающих свойство допустимости) - меньше вершин развертывается, но может быть найдено не оптимальное решение:
  - Взвешенный A*:
    - $f(n) = g(n) + W * h(n), W > 1$
- A* с ограничением по памяти, MA*, SMA*.
- A* с итеративным углублением

Эвристический поиск. Все ли эвристики одинаково хороши?





Решение содержит 26 действий.

Возможные эвристики:

- h_1 – количество фишек, находящихся не на своем месте ($h_1 = 8$)
- h_2 – сумма расстояний между текущими и целевыми позициями фишек ($h_1 = 18$)

Эвристический поиск. Эффективный коэффициент ветвления

Если общее количество узлов, вырабатываемых в процессе поиска *N*, а глубина решения *d*, то b^* представляет собой коэффициент ветвления, который должно иметь однородное дерево с глубиной *d*, чтобы в нем содержалось *N*+1 узлов. Поэтому:

$$N + 1 = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d$$

	Search Cost (nodes generated)			Effective Branching Factor		
d	BFS	A*(h_1)	A*(h_2)	BFS	A*(h_1)	A*(h_2)
6	128	24	19	2.01	1.42	1.34
8	368	48	31	1.91	1.40	1.30
10	1033	116	48	1.85	1.43	1.27
12	2672	279	84	1.80	1.45	1.28
14	6783	678	174	1.77	1.47	1.31
16	17270	1683	364	1.74	1.48	1.32
18	41558	4102	751	1.72	1.49	1.34
20	91493	9905	1318	1.69	1.50	1.34
22	175921	22955	2548	1.66	1.50	1.34
24	290082	53039	5733	1.62	1.50	1.36
26	395355	110372	10080	1.58	1.50	1.35
28	463234	202565	22055	1.53	1.49	1.36

Эвристический поиск. Доминирование

Эвристика h_2 **доминирует** над h_1, т.к. для любого n $h_2(n) >= h_1(n)$.

Следовательно, при использовании в A* эвристики h_2 не будет развернуто больше узлов, чем при использовании h_1 (за исключением, возможно, нескольких узлов с оптимальной оценкой стоимости).

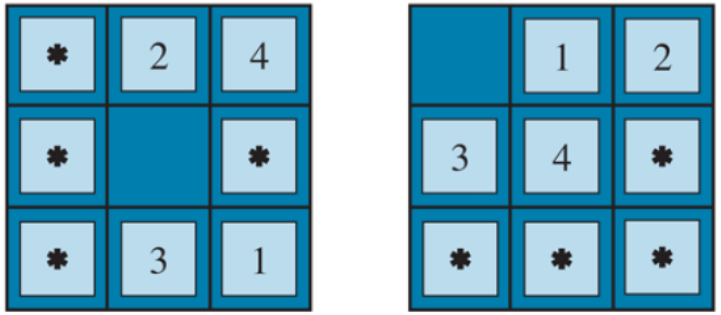
Неформальное доказательство:

1. В A* развертываются все узлы с $f(n) < C^*$
2. Значит, должен быть развернут каждый узел с $h(n) < C^* - g(n)$
3. Но правая часть константа, и если неравенство выполняется для узла с h_2, то оно выполнится для этого узла и с h_1 (но может выполниться также для других узлов)

Чем больше значения, тем лучше эвристика (при условии сохранения допустимости*)

Эвристический поиск. Источники эвристик

- Ослабление ограничений задачи
  - В игре 8: возможность пересечения фишек
  - Суперграф по отношению к оригинальной задаче (больше ребер), значит оптимальное решение основной – это решение ослабленной, а решение ослабленной – допустимая эвристика
- База паттернов (решений подзадачи)
  - В игре 8: решение части задачи – например, фишек 0-4
- Опорные точки
- Обучение (эвристика как функция от нескольких признаков)

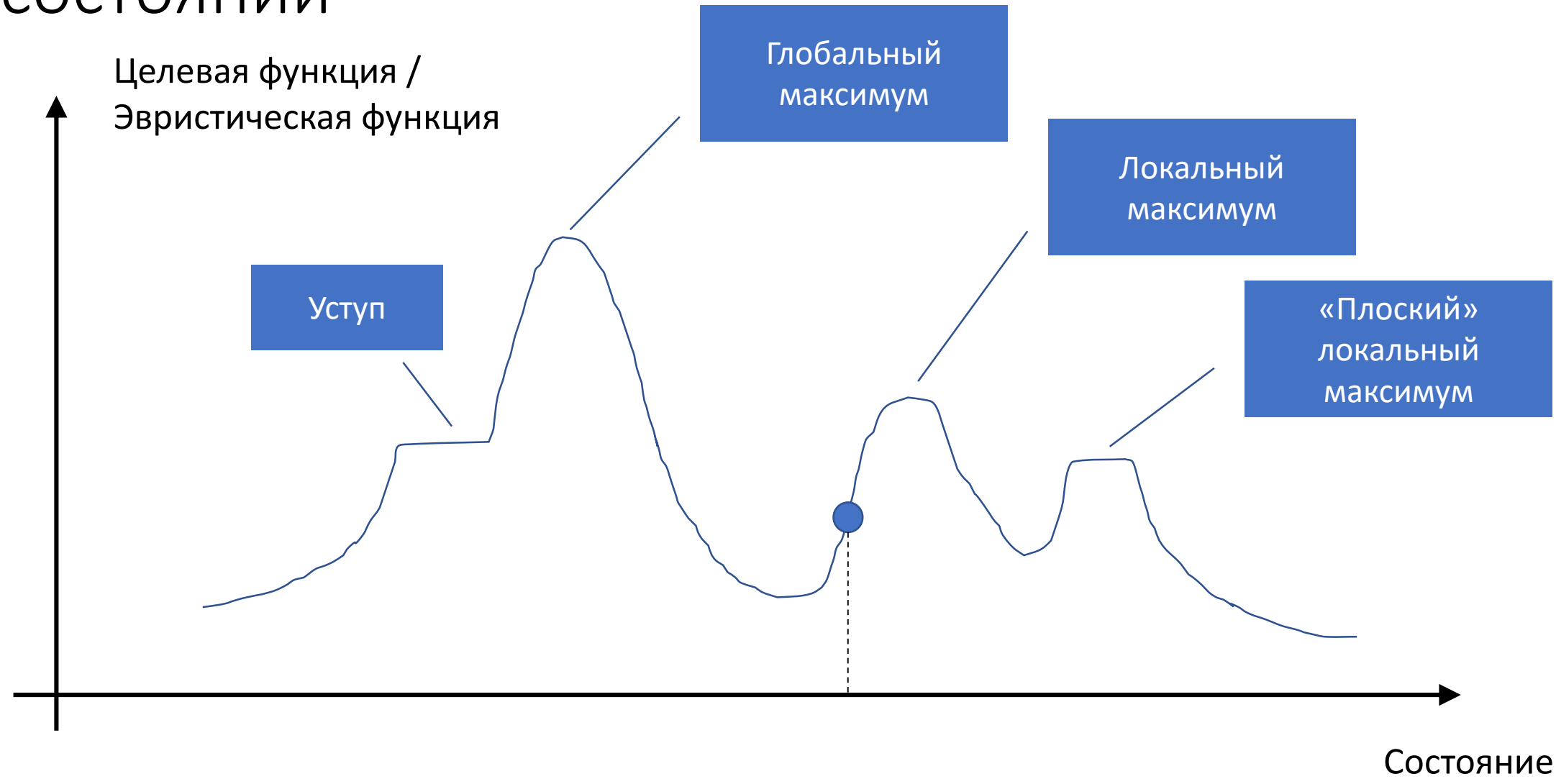


Тема 3. Локальный поиск

Локальный поиск и оптимизация

- Что если путь не нужен? (может быть легко реконструирован или вообще не имеет смысла)
  - Составление плана
  - Удовлетворение ограничений
- Ответом могут быть **алгоритмы локального поиска**:
  - Действуют на основе единственного *текущего состояния*
  - Рассматривают только переход к некоторое *соседнее состояние*
  - Не предусматривают систематическое исследование пространства состояний:
    - (+) Небольшой объем памяти
    - (+) Работа с очень большими пространствами состояний
    - (-) Не всегда находится действительно наилучшее решение

Ландшафт одномерного пространства состояний



Поиск с восхождением к вершине (Hill-climbing)

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    current ← problem.INITIAL
    while true do
        neighbor ← a highest-valued successor state of current
        if VALUE(neighbor) ≤ VALUE(current) then return current
        current ← neighbor
```

Причины проблем поиска восхождением

- Локальные максимумы
- Хребты
- Плато

Решение проблем поиска восхождением

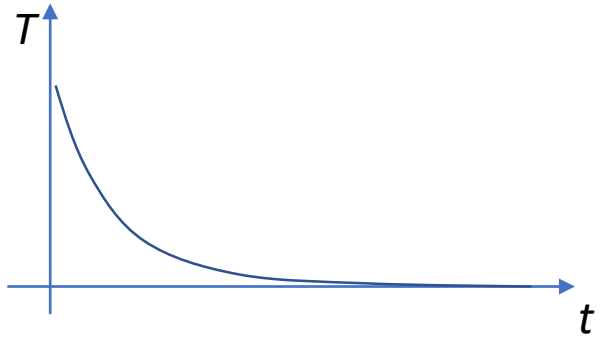
- Поиск с восхождением с выбором первого варианта
  - Выбирается не лучший из «соседей», а первый такой, у которого значение целевой функции лучше, чем у данного
- Поиск с восхождением и перезапуском случайным образом
  - Несколько поисков из случайных состояний

Поиск с эмуляцией отжига (Simulated annealing)

Также: ***имитация*** отжига

Внимание! Тут минимизация!

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  current ← *problem*.INITIAL
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE(*current*) – VALUE(*next*)
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{\Delta E/T}$



Если лучше, то принимаем

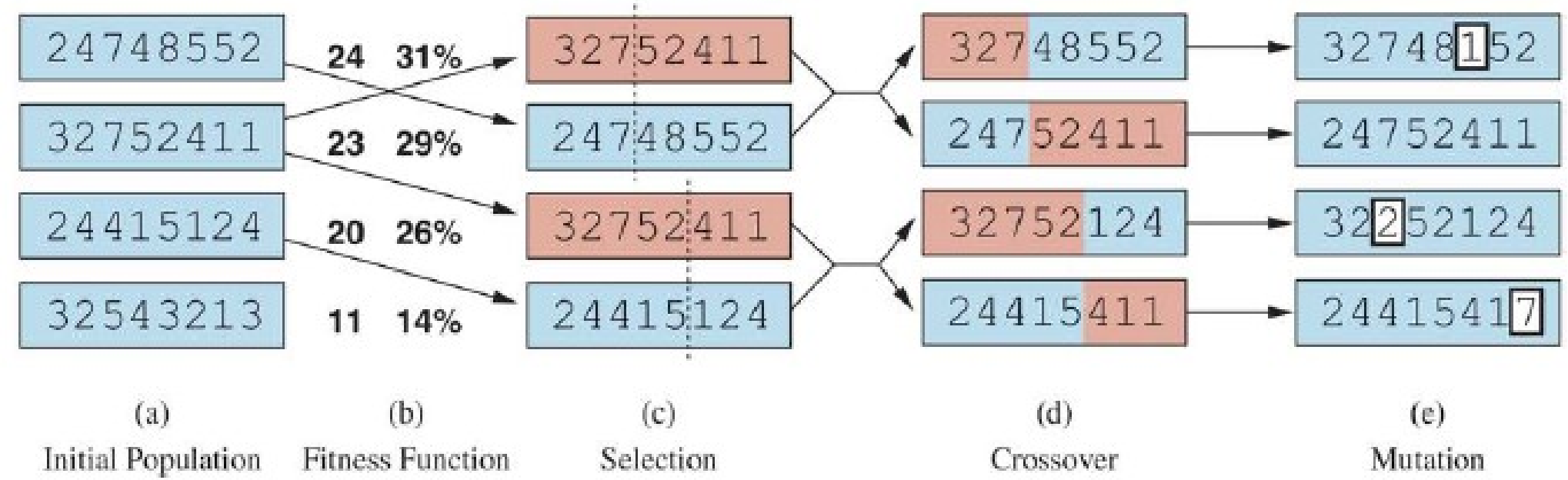
Локальный лучевой поиск (Local beam search)

- Общая идея: хранить не одно состояние, *k* (своеобразный «луч»)
- Общая схема:
  - Начать с *k* случайным образом выбранных
  - Сформировать всех соседей для каждого из *k* состояний
  - Оставить для следующей итерации только *k* из них
    - Лучшие
    - Стохастически – с вероятностью, пропорциональной качеству

Эволюционные алгоритмы

- Общая идея: моделировать естественный отбор среди решений
- Особенности:
  - Рассматривается целая *популяция* (изначально формируется случайным образом), состоящая из *индивидов* (состояний)
  - Индивиды представлены особым образом, допускающим осмысленные:
    - Рекомбинацию
    - Мутацию
  - Задана функция оценки (функция пригодности, приспособленности, fitness function)

Эволюционные алгоритмы



Эволюционные алгоритмы. Параметры и разновидности

- Размер популяции
- Представление индивида (аналог ДНК)
  - Строка в каком-то алфавите (бинарная, цифровая, буквенная, …)
- Количество индивидов, участвующих в формировании нового
  - Обычно 2, но не обязательно
- Процесс отбора – какие индивиды будут участвовать в формировании новых
- Процедура рекомбинации
  - Обычно, кроссинговер, но не обязательно
- Скорость и характер мутации
- Формирование следующего поколения
  - Элитизм (переход «хороших» индивидов в новое поколение)?

Резюме

- Решение задач с помощью поиска
  - Состояние, действие, восстановление последовательности действий
- Алгоритмы поиска
  - Неинформированные
    - Best-first-search, поиск в ширину, поиск в глубину
  - Информированные
    - A* и его вариации
  - Эвристики и их свойства
- Локальный поиск
  - Поиск с восхождением к вершине
  - Отжиг, генетические алгоритмы
- Рекомендуемая литература:
  - С. Рассел, П. Норвиг Искусственный интеллект: современный подход, 4-е изд.
  - https://github.com/aimacode/aima-python