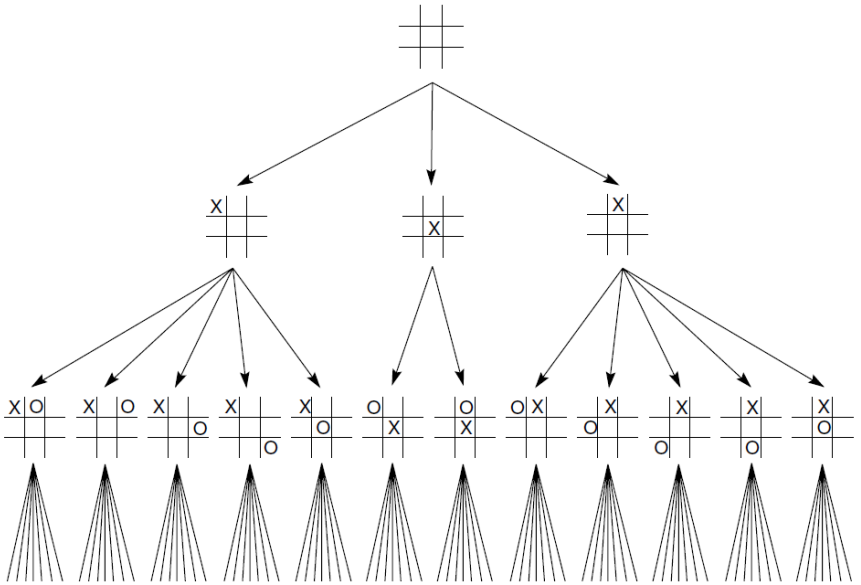
Информирано търсене в пространство на състояния

Евристично търсене

Евристично търсене

- Приложими са при наличие на специфична информация за предметната област, позволяваща да се конструира оценяваща функция (евристика), която връща числова оценка.
- Тази оценка може да служи например за мярка на близостта на оценяваното състояние до целта или на необходимия ресурс за достигане от оценяваното състояние до целта.



Евристично търсене
Heuristic Search

- Евристичната функция оценява броя ходове от дадено състояние до целта или стойността на пътя до нея. Изисквания: да е равна на 0 в целевите състояния и изчисляването и да не е свързано с голям разход на време и памет.
- Методите за евристично търсене реализират пълно изчерпване по гъвкава стратегия или търсене с отсичане на част от графа на състоянията.

Методи на най-доброто спускане best-first search

- ✓ *„Лакомо“ търсене*
  - ✓ *(greedy best first search).*

  Развива се състоянието, в което стойността на евристичната функция е минимална. Стратегията е ефективна, но не е пълна, нито оптимална.

  - ✓ *Beam Search*

  Търсене с ограничена широчина (търсене в лъч, beam search) - ограничаване на списъка

- ✓ *Търсене с минимизиране на общата цена на пътя A*.* Развива се състоянието, в което сумата от цената на изминатия път и стойността на евристичната функция е минимална.

Методи на най-доброто спускане best-first search

- Избира се възел за разширяване според оценяващата функция. Нарича се най-добро спускане, ако оценяващата функция намалява към целевото състояние.
- Използва списъци *frontier* за фронта на търсенето и closed за проучените състояния.
- Добавената стъпка в алгоритъма сортира състоянията с използване на евристичната- приблизителна оценка на тяхната "близост" до целта.

```
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)
```

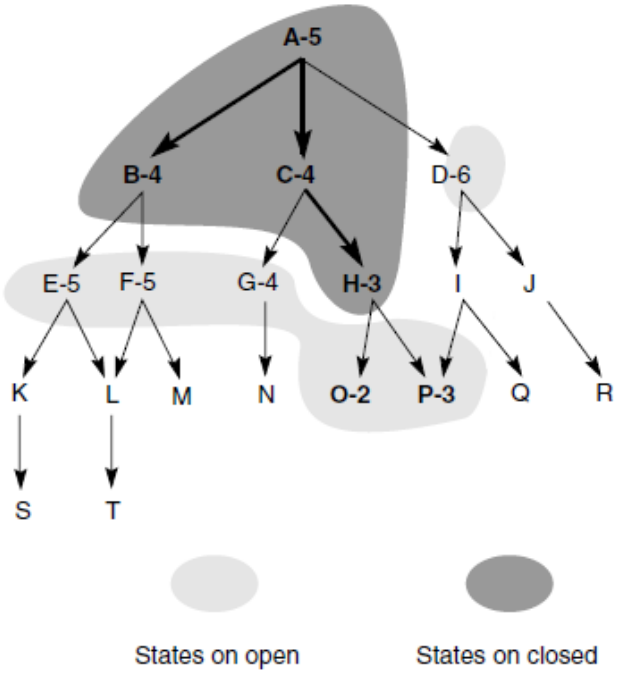
Greedy best-first search

- Алгоритъм за най-добро спускане, избира възел за разширяване според оценяващата функция. Използва списъци frontier за текущият фронт на търсенето и explored за проучените състояния.
- Добавената стъпка в алгоритъма сортира състоянията в frontier според евристичната-приблизителна оценка на тяхната "близост" до целта.

```
function GREEDY-BEST-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :  /* Cost f(n) = h(n) */

    frontier = Heap.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE
```

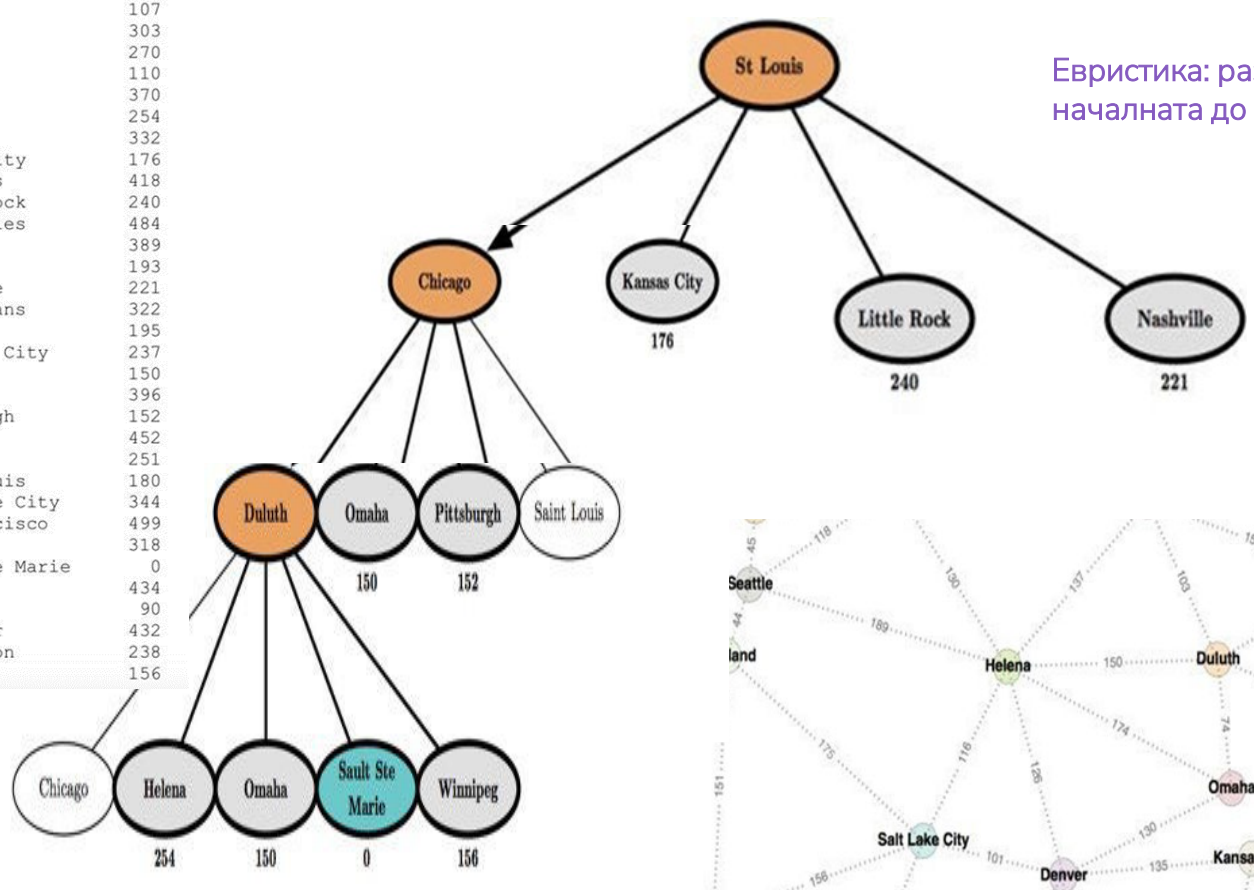
Пример Best-first search

1. open = [A5]; closed = [ ]
2. evaluate A5; open = [B4,C4,D6];
closed = [A5]
3. evaluate B4; open = [C4,E5,F5,D6];
closed = [B4,A5]
4. evaluate C4; open = [H3,G4,E5,F5,D6];
closed = [C4,B4,A5]
5. evaluate H3; open = [O2,P3,G4,E5,F5,D6];
closed = [H3,C4,B4,A5]
6. evaluate O2; open = [P3,G4,E5,F5,D6];
closed = [O2,H3,C4,B4,A5]
7. evaluate P3; the solution is found!

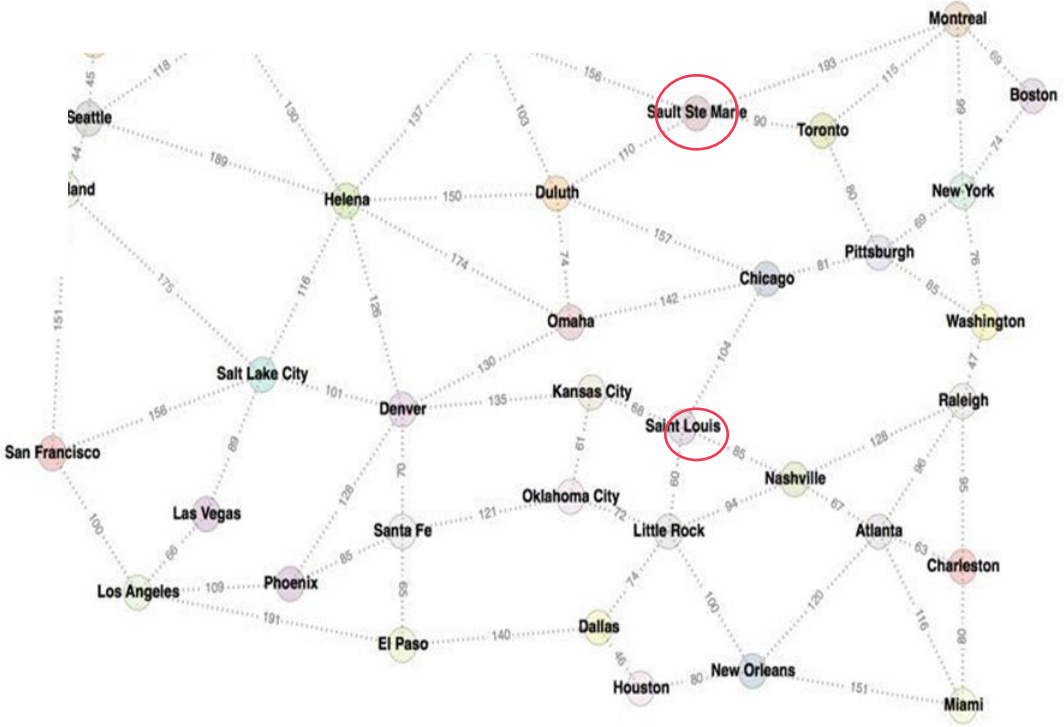


Greedy Best-first search St.Luis->Sault St. Marie

Atlanta	272
Boston	240
Calgary	334
Charleston	322
Chicago	107
Dallas	303
Denver	270
Duluth	110
El Paso	370
Helena	254
Houston	332
Kansas City	176
Las Vegas	418
Little Rock	240
Los Angeles	484
Miami	389
Montreal	193
Nashville	221
New Orleans	322
New York	195
Oklahoma City	237
Omaha	150
Phoenix	396
Pittsburgh	152
Portland	452
Raleigh	251
Saint Louis	180
Salt Lake City	344
San Francisco	499
Santa Fe	318
Sault Ste Marie	0
Seattle	434
Toronto	90
Vancouver	432
Washington	238
Winnipeg	156

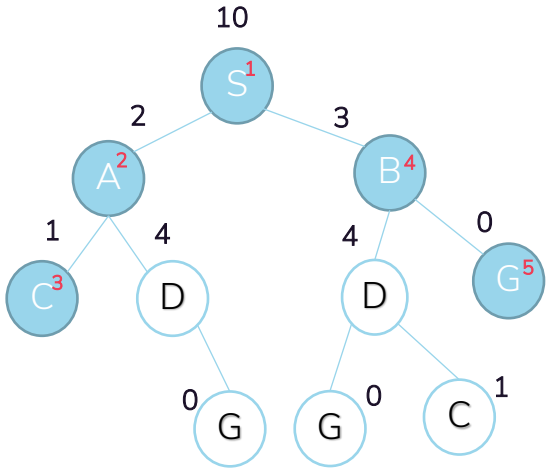


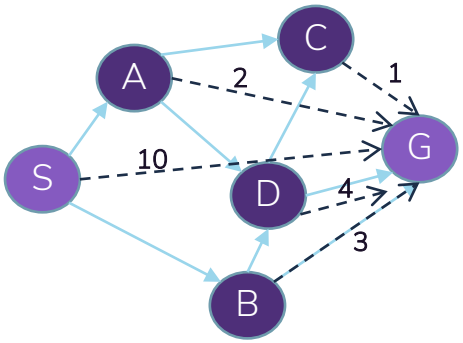
Евристика: разстоянието по права линия от началната до крайната точка (SLD)



Best first Search -пример

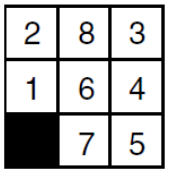
h(A)=2 h(B)=3 h(C)=1 h(D)=4 h(S)=10 h(G)=0

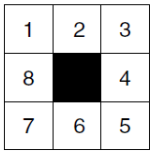




Open	Visited node
(S,10)	()
((S,A,2),(S,B,3))	(S,10)
((A,C,1),(S,B,3),(A,D,4))	(S,A,2)
((S,B,3),(A,D,4))	(A,C,1)
((A,D,4)	(S,B,3)
((B,G,0), (A,D,4),(B,D,1))	**(B,G,0)**
S->-B->G	

Плъзгащ пъзел - преразгледан



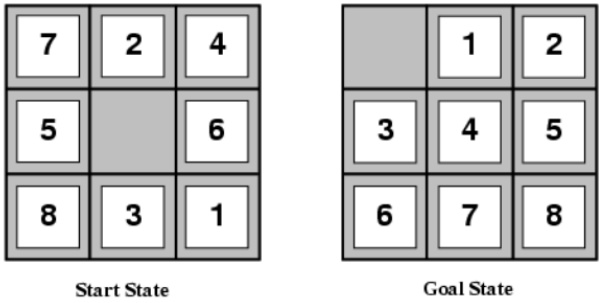


8-puzzle

- $MisplacedTiles(n) = 5$
- *Manhattan Distance* $(n) = 1 + 1 + 0 + 0 + 0 + 1 + 1 + 2$

2 8 3 1 6 4 7 5	**5**	**6**	**0**
2 8 3 1 4 7 6 5	**3**	**4**	**0**
2 8 3 1 6 4 7 5	**5**	**6**	**0**
	Tiles out of place	Sum of distances out of place	2 x the number of direct tile reversals

Плъзгащ пъзел



Пример

h1(Start) = 8

h2(Start) = 3+1+2+2+2+3+3+2 = 18

Heuristics
Sliding puzzle- revisited

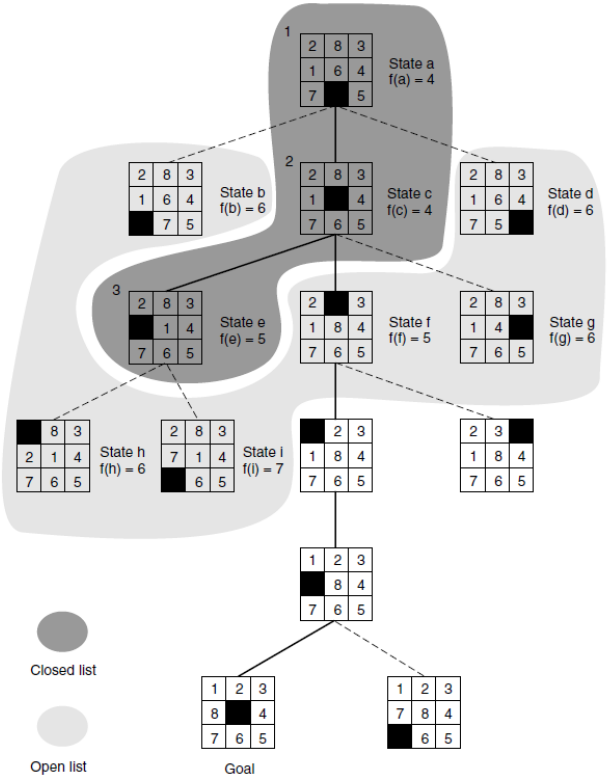
2	8	3
1	6	4
	7	5

1	2	3
8		4
7	6	5

8-puzzle

$h_1(n)$ *heuristic* *Manhattan Distance*

$$h_2\,(n) = 1 + 1 + 0 + 0 + 0 + 1 + 1 + 2$$



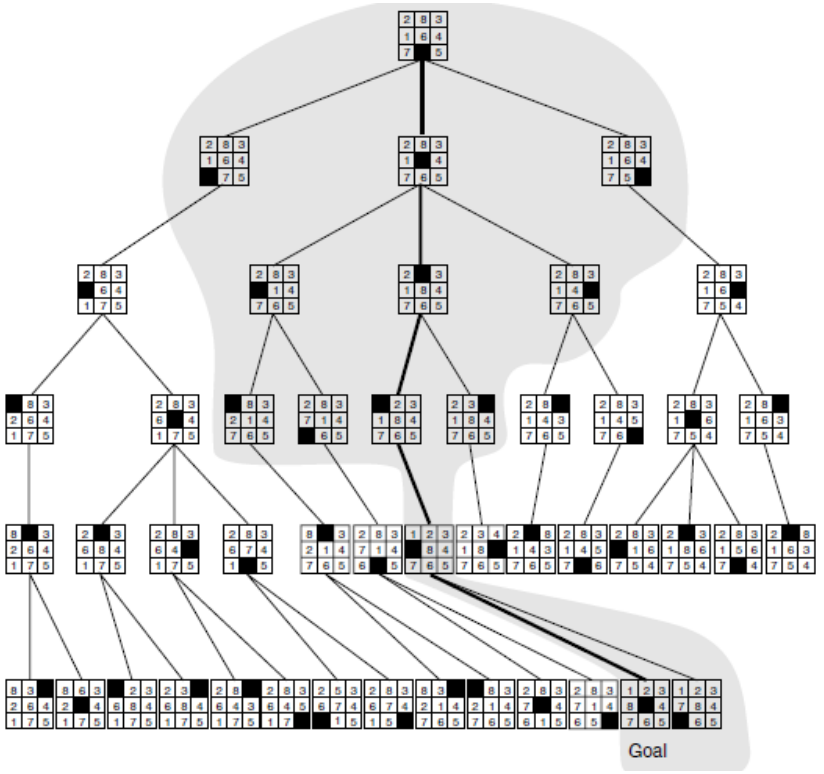
Търсене с минимизиране цената на пътя A*

Вариант на основния алгоритъм за най-доброто спускане (best first search). Най-широко използван вариант на best-first search

Пример:

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

(h(n) броят плочки не на място.)



Сравнение с обхождането в широчина (сиво обхождане със стратегията A*)

A star

```
function A-STAR-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :  /* Cost f(n) = g(n) + h(n) */

    frontier = Heap.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE
```

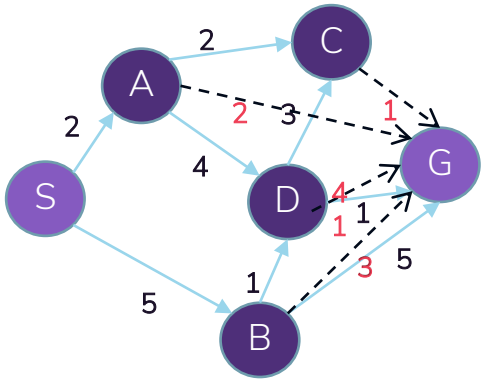
Greedy best-first search

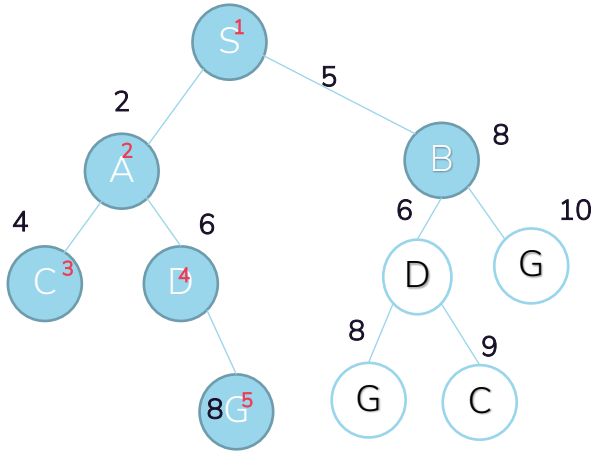
```
function GREEDY-BEST-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :  /* Cost f(n) = h(n) */

    frontier = Heap.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE
```

A*

h(A)=2 h(B)=3 h(C)=1 h(D)=4 h(S)=10
h(G)=0

Приемлива евристика



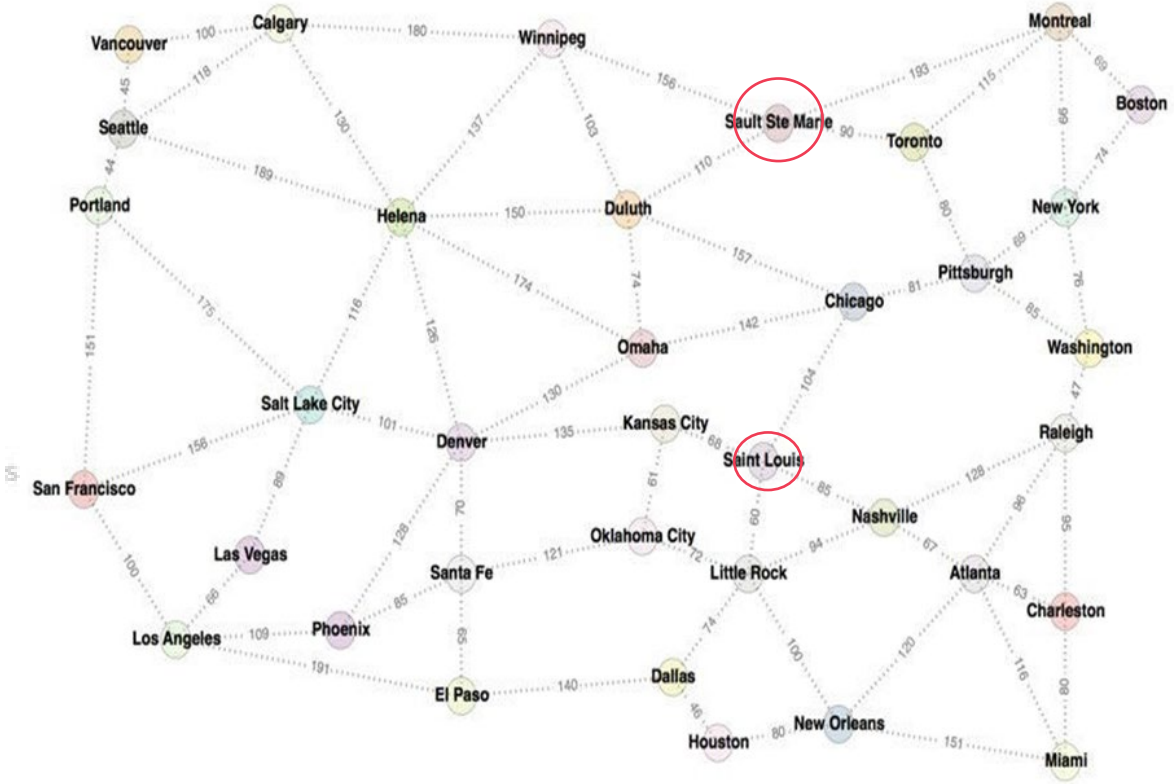


Open	Visited node
(S,0)	()
((S,A,4),(S,B,8))	(S,0)
((S,A,C,5),(S,A,D,7), (S,B,8))	(S,A,4)
((S,A,D,7), (S,B,8))	(A,C,5)
((S,A,D,G,8), (S,B,8),(S,A,D,C,9))	(A,D,7)
((S,B,8),(S,A,D,C,9))	**(D,G,8)**
S->A->D->G	

A* Search пример карта

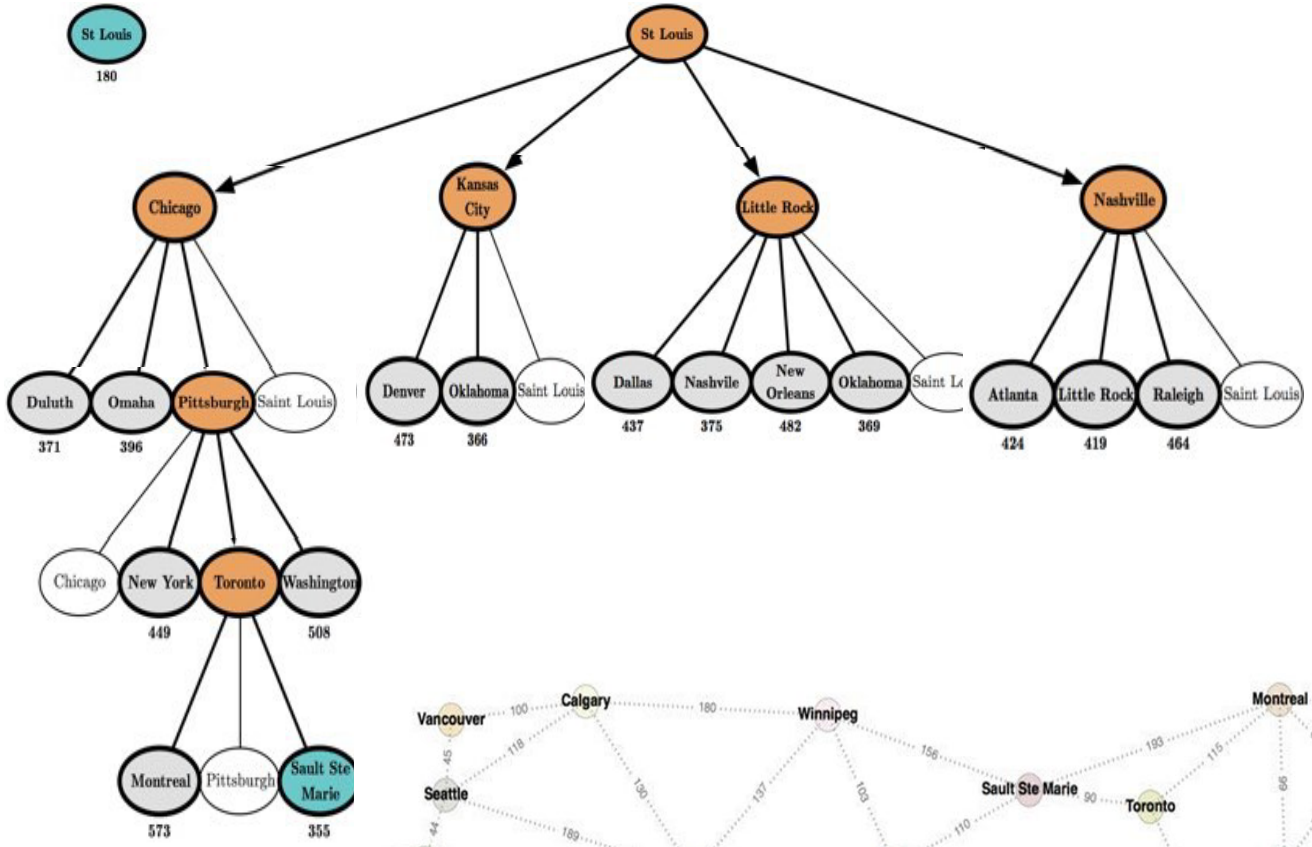
Straight Line Distance

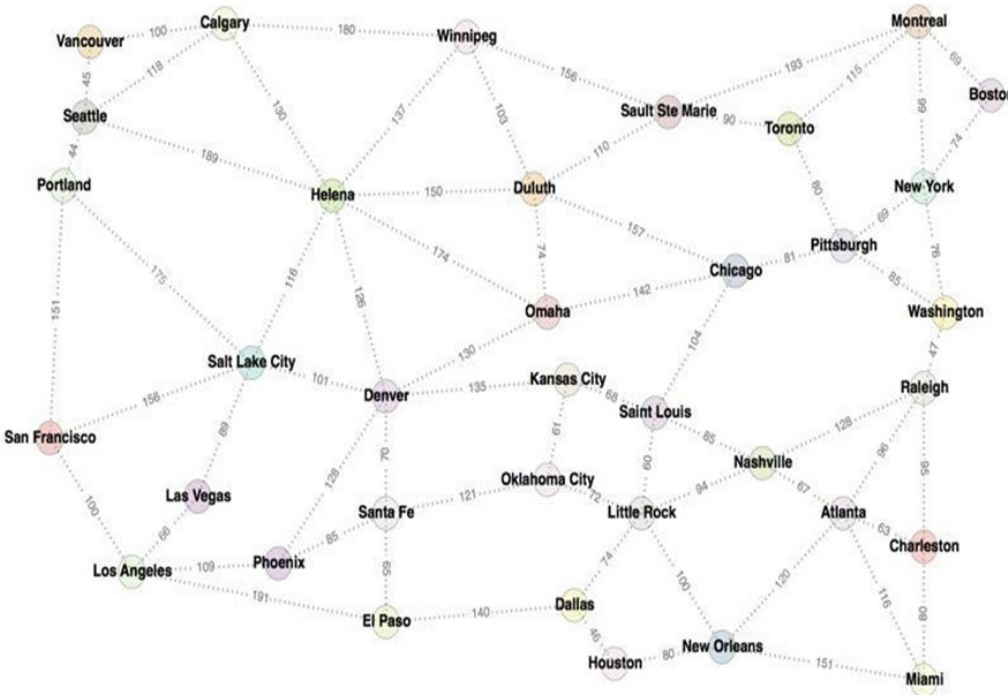
Atlanta	272
Boston	240
Calgary	334
Charleston	322
Chicago	107
Dallas	303
Denver	270
Duluth	110
El Paso	370
Helena	254
Houston	332
Kansas City	176
Las Vegas	418
Little Rock	240
Los Angeles	484
Miami	389
Montreal	193
Nashville	221
New Orleans	322
New York	195
Oklahoma City	237
Omaha	150
Phoenix	396
Pittsburgh	152
Portland	452
Raleigh	251
Saint Louis	180
Salt Lake City	344
San Francisco	499
Santa Fe	318
Sault Ste Marie	0
Seattle	434
Toronto	90
Vancouver	432
Washington	238
Winnipeg	156



A* Search

Atlanta	272
Boston	240
Calgary	334
Charleston	322
Chicago	107
Dallas	303
Denver	270
Duluth	110
El Paso	370
Helena	254
Houston	332
Kansas City	176
Las Vegas	418
Little Rock	240
Los Angeles	484
Miami	389
Montreal	193
Nashville	221
New Orleans	322
New York	195
Oklahoma City	237
Omaha	150
Phoenix	396
Pittsburgh	152
Portland	452
Raleigh	251
Saint Louis	180
Salt Lake City	344
San Francisco	499
Santa Fe	318
Sault Ste Marie	0
Seattle	434
Toronto	90
Vancouver	432
Washington	238
Winnipeg	156



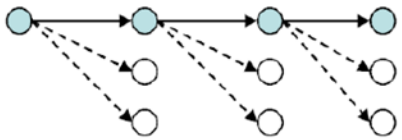


Beam Search

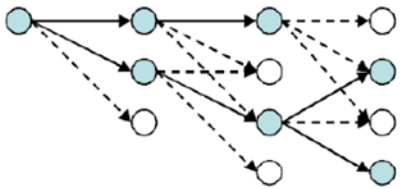
- Търсене с ограничена широчина (търсене в лъч, beam search) - ограничаване на списъка Open/fringe до първите n най-добри възела (в съответствие с евристиката). При n=1 се доближава до метода на най-бързото изкачване.

OPEN = {initial state}
while OPEN is not empty do
1. Remove the best node from OPEN, call it n.
2. If n is the goal state, backtrace path to n (through recorded parents) and return path.
3. Create n's successors.
4. Evaluate each successor, add it to OPEN, and record its parent.
5. If |OPEN| > k , take the best k nodes (according to heuristic) and remove the others from the OPEN.

done

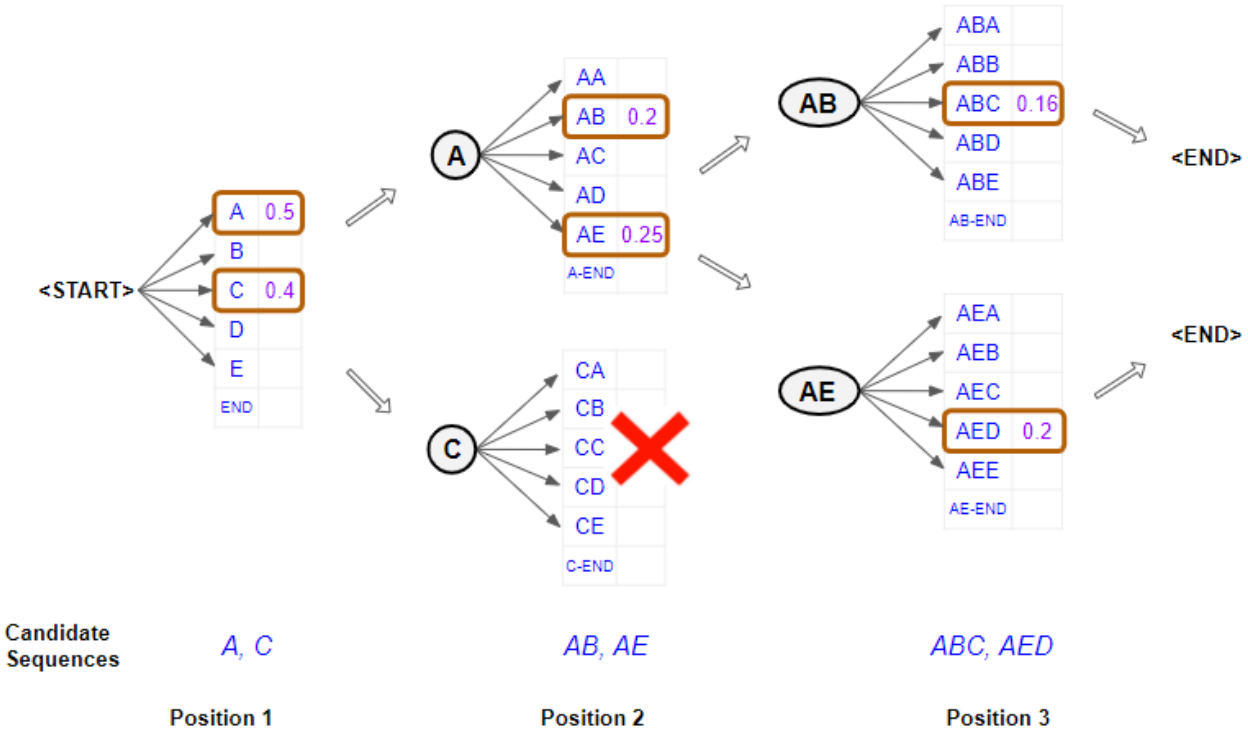


Greedy search



Beam search

Beam Search example



Greedy Best-first search

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Values of h_{SLD}—straight-line distances to Bucharest.

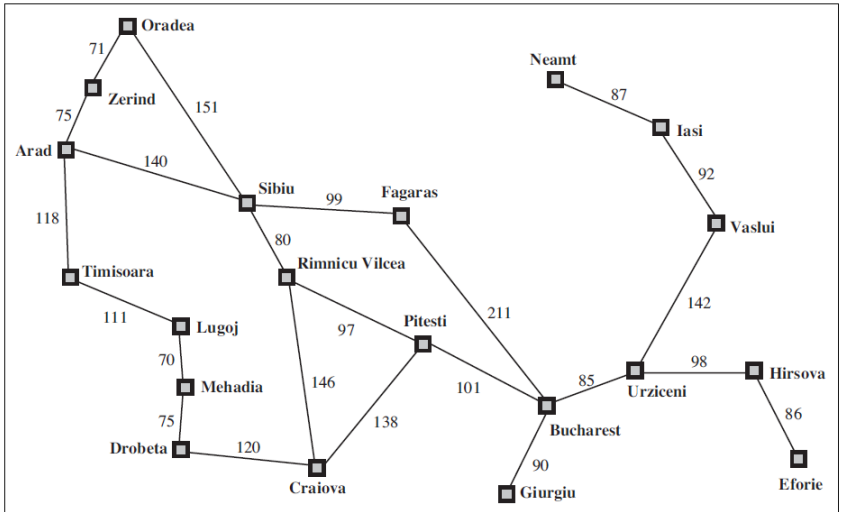


Figure 3.2 A simplified road map of part of Romania.

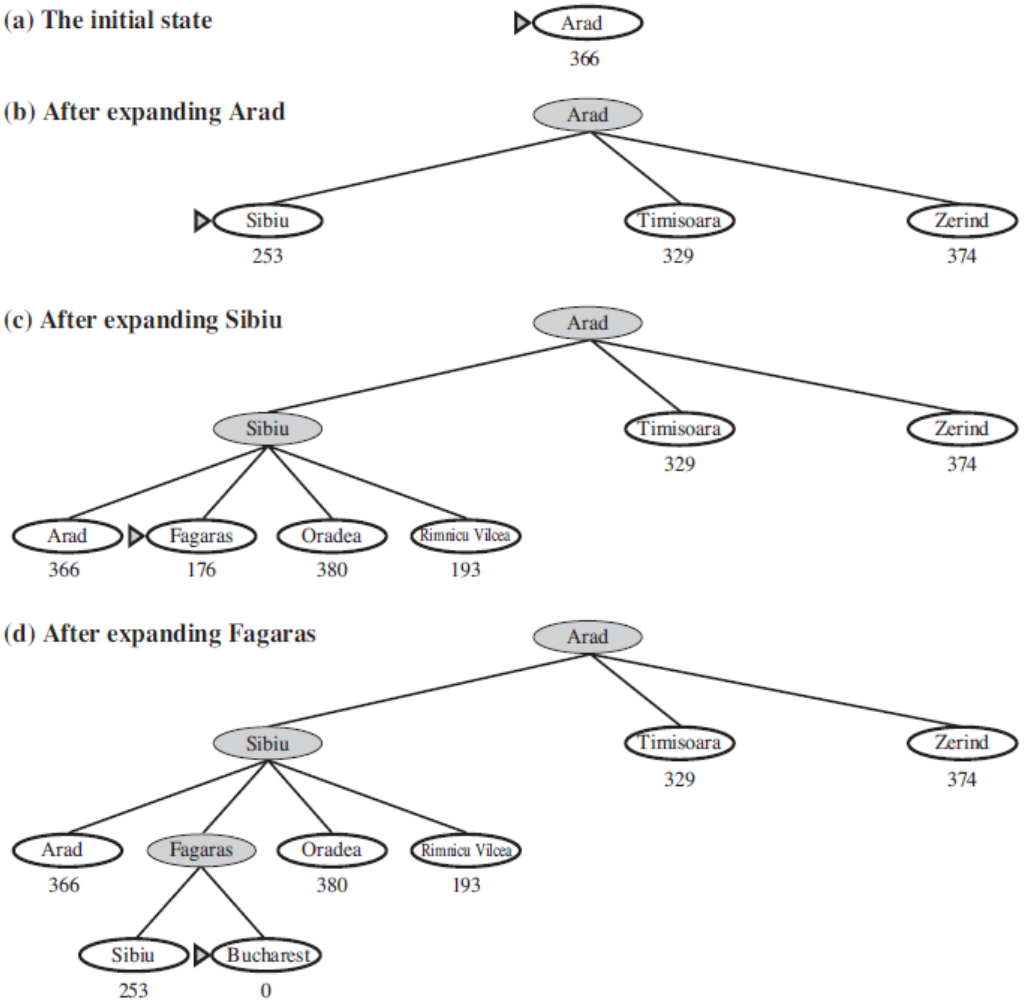


Figure 3.23 Stages in a greedy best-first tree search for Bucharest with the straight-line distance heuristic h_{SLD}. Nodes are labeled with their h-values.

A star

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Values of h_{SLD}—straight-line distances to Bucharest.

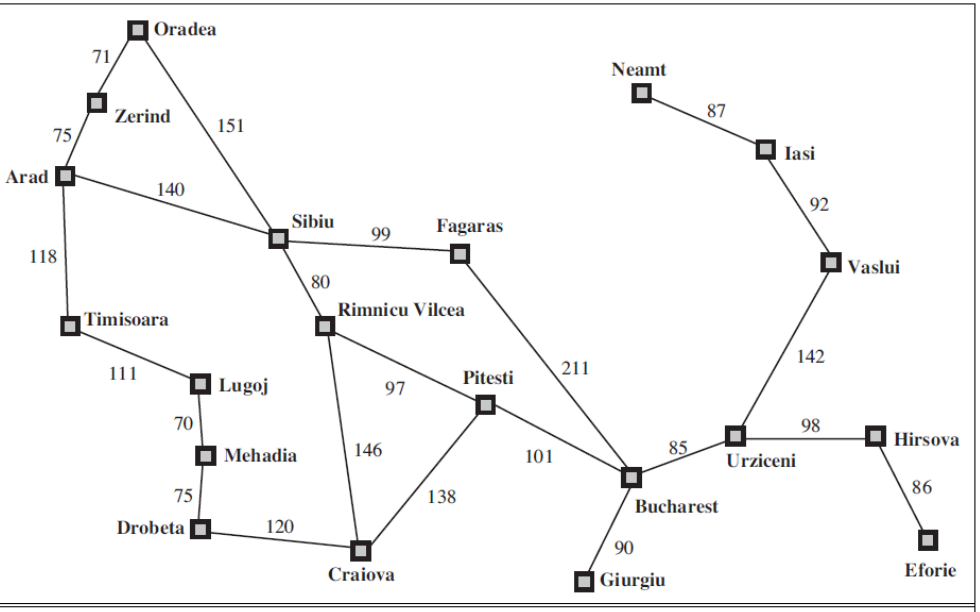


Figure 3.2 A simplified road map of part of Romania.

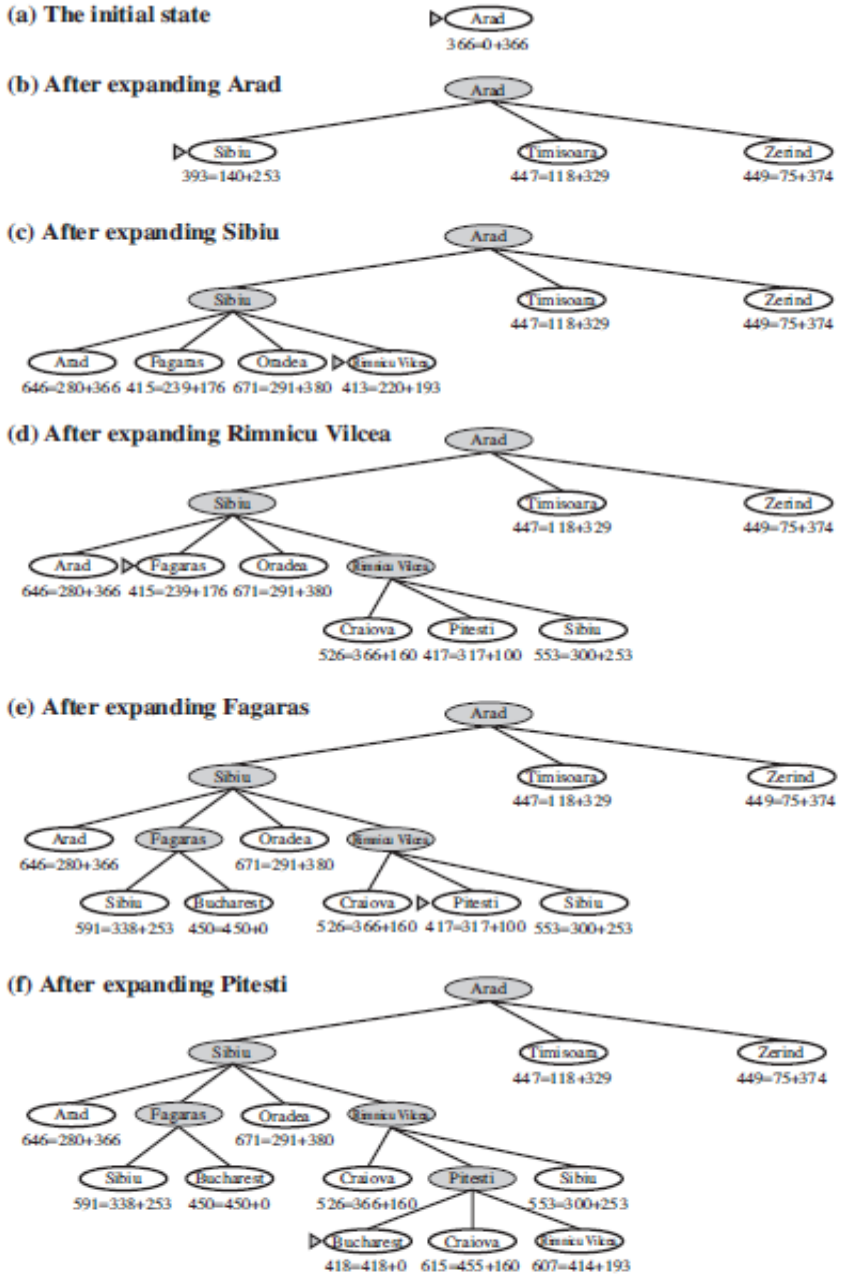


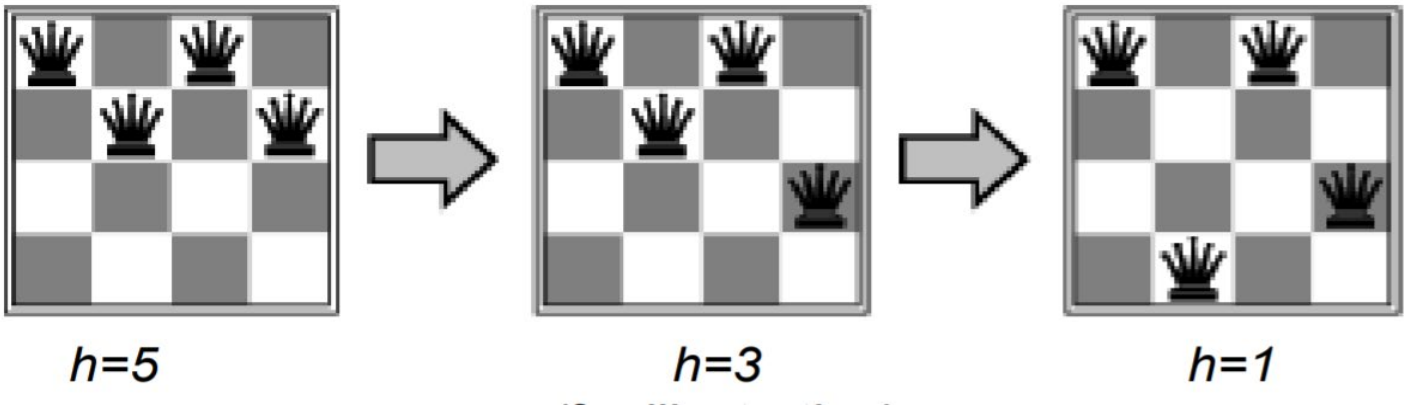
Figure 3.24 Stages in an A* search for Bucharest. Nodes are labeled with $f = g + h$. The h values are the straight-line distances to Bucharest taken from Figure 3.22.

4 Queen

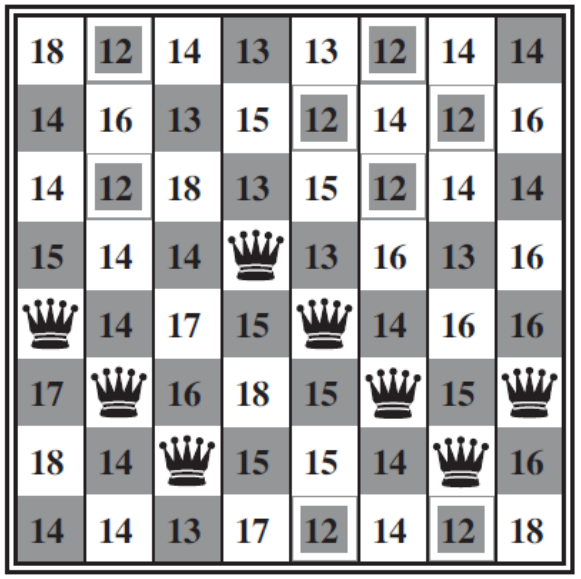
n-queens задача: Необходимо е да се поставят осем царици на шахматната дъска по такъв начин, че нито една от тях да не застрашава друга

евристика:

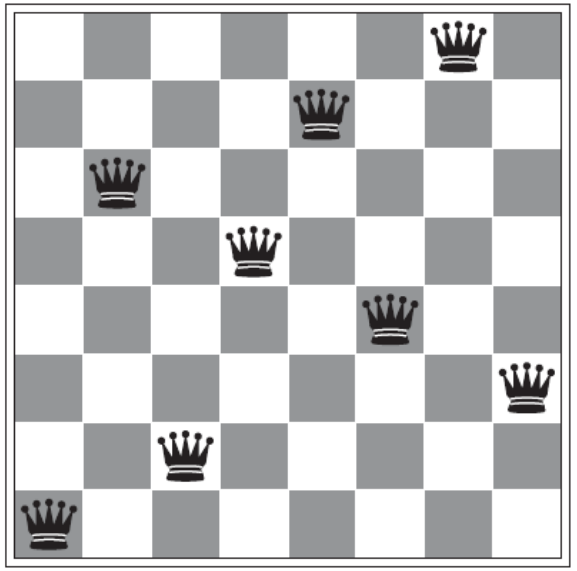
h = двойки нападащи се царици



8 Queen problem



(a)



(b)

Figure 4.3 (a) An 8-queens state with heuristic cost estimate $h = 17$, showing the value of h for each possible successor obtained by moving a queen within its column. The best moves are marked. (b) A local minimum in the 8-queens state space; the state has $h = 1$ but every successor has a higher cost.

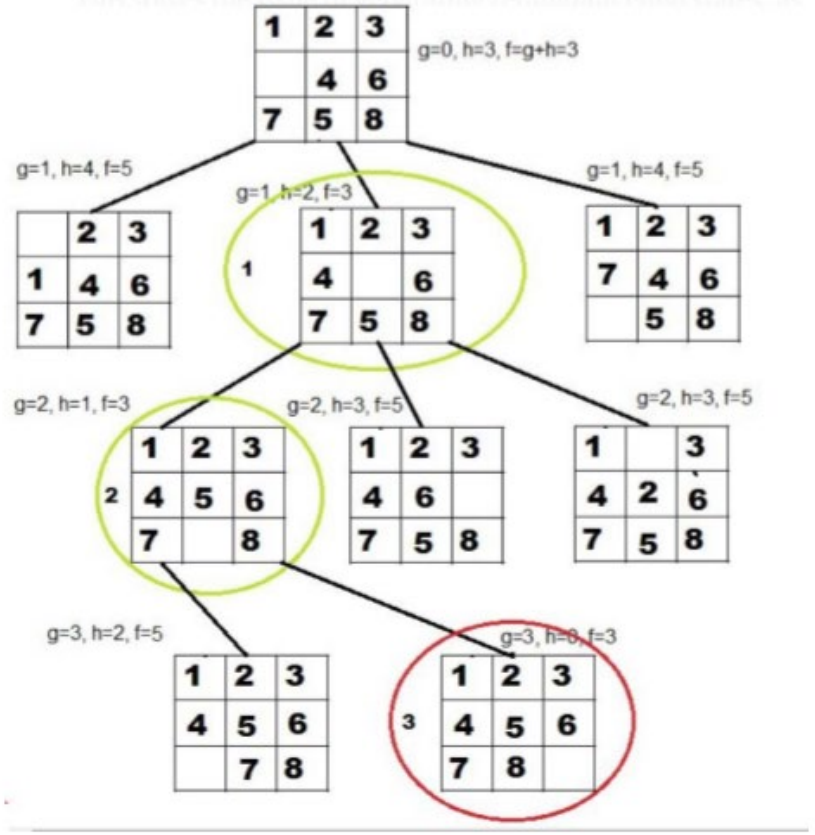
Задачи A*

1	2	3
4	5	6
7		8

Depth (g): 2

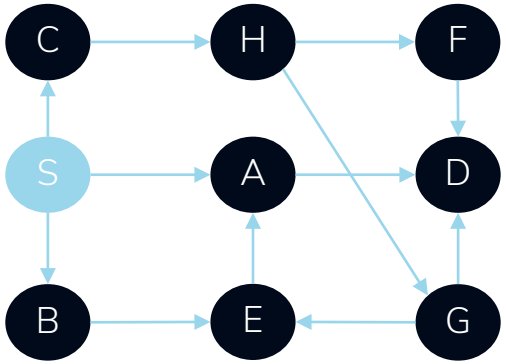
Misplaced Tiles (h): 2

f: **4**



Greedy Best-first search, Beam Search

A=5 B=1 C=3 D=9 E=4 F=0 G=0 H=7



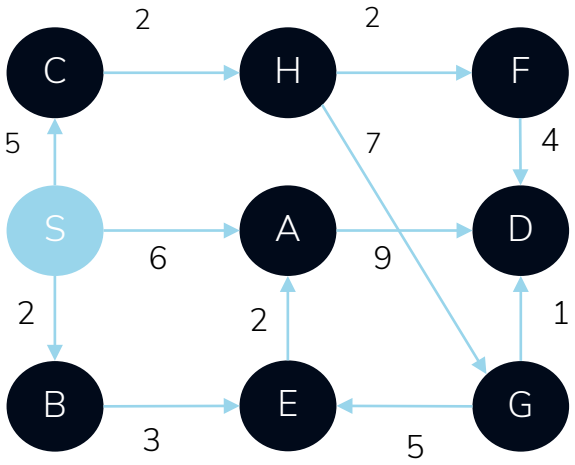
Open	Visited
S	()
(1 B S) (3 C S) (5 A S)	S
(3 C S) (4 E B S) (5 A S)	S,B
(4 E B S) (5 A S) (7 H C S)	S,B,C
(5 A E B S) (7 H C S)	S,B,C,E
(7 H C S) (9 D A E B S)	S,B,C,E,A
(0 G H C S) (9 D A E B S)	S,B,C,E,A,D,H,G
	(0 G H C S)

Beam 2	
S	()
(1 B S) (3 C S) (5 A S)	S
(3 C S) (4 E B S)	S,B
(4 E B S) (7 H C S)	S,B,C
(5 A E B S) (7 H C S)	S,B,C,E
(7 H C S) (9 D A E B S)	S,B,C,E,A
(0 F H C S) (0 G H C S)	S,B,C,E,H
(0 G H C S)	S,B,C,E,A,FG
	(0 G H C S)

A* search

Heuristic Values

A=5 B=1 C=3 D=9 E=4 F=0 G=0 H=7



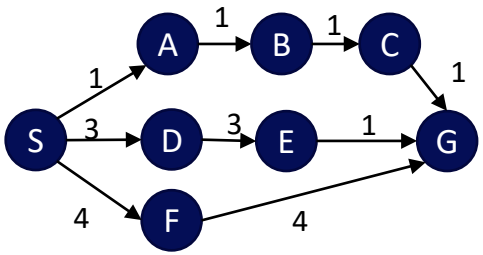
A* Open	V
S	S
(2+1 B S) (5+3 C S) (6+5 A S)	S,B
(5+3 C S) (5+4 E B S) (6+5 A S)	S,B,C
(5 +4 E B S) (6+5 A S) (8+7 H C S)	S,B,C,E
(7+5 A E B S) (6+5 A S) (8+7 H C S)	S,B,C,E,A
(15+9 D A S) (7+5 A E B S) (8+7 H C S)	S,B,C,E,A,H,
(14+0 G H C S) (15+9 D A E B S) (7+5 A S)	S,B,C,E,A,H,G

A * , Best First search

Heuristic Values

A=3,B=2,C=1,D=2,E=1,F=3,G=0,S=4

Start S goal G



BestFS	Visited
4 S	S
(3AS) (2DS) (3FS)	S,D
(3AS)(1EDS) (3FS)	S,A,D,E
(0 GEDS) (3AS)((3FS)	S,A,D,E

A*	Visited
4 S	S
(1+3AS) (3+2DS) (4+3FS)	S,A
(2+2BAS)(3+2DS) (4+3FS)	S,A,B
(3+1CBAS)(3+2DS) (4+3FS)	S,A,B,C
(4+0GCBAS)(3+2DS) (4+3FS)	S,A,B,C,G

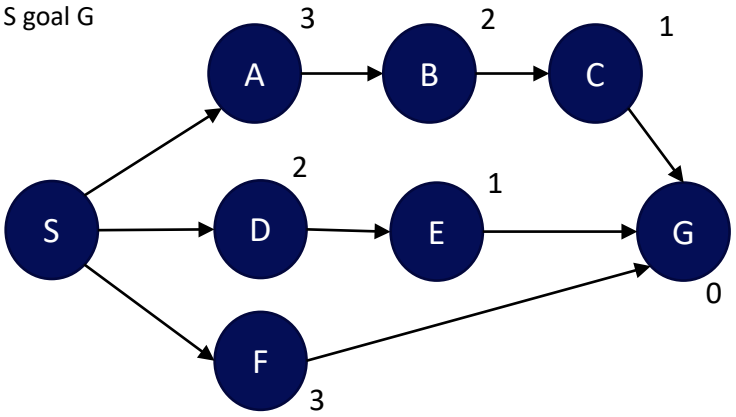
Beam search

Heuristic Values

A=3,B=2,C=1,D=2,E=1,F=3,G=0,S=4

Simple Hill Climbing
4 S
2DS
1EDS
0 GEDS

Start S goal G



Beam Search 2	
4 S	
(2DS) (3AS) (3 F S)	S
(1EDS) (2DS) (3AS)	D
(0GEDS) (2DS)	E

Добра евристика

- If $h_1(n) < h_2(n) < h^*(n)$ for all n, h_2 По-добра от (dominates) h_1
- Премахване на ограниченията, за да създаде (много) по-лесен проблем
- Комбиниране на евристика: вземете максимума от няколко допустими евристика
- Използване на статистически оценки за изчисляване на g
- Идентифицирайте добри функции, след това използвайте алгоритъм за обучение, за да намерите евристична функция

Оптимална стратегия и евристика

❑ Tree Search

A search uses an admissible heuristic

i.e., h(n) ≤ h*(n) where h*(n) is the true cost from n.

(Also require h(n) 0, so h(G) = 0 for any goal G.)

❑ Graph Search

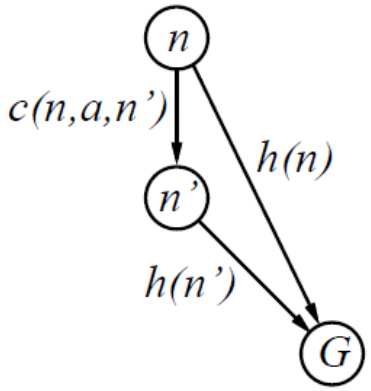
A heuristic is consistent if

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

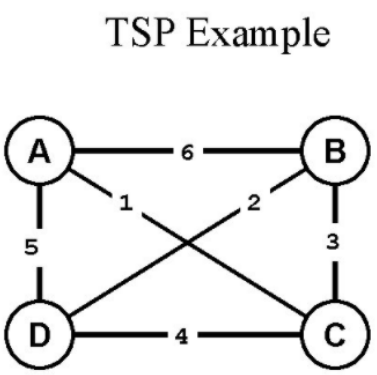
If h is consistent, we have

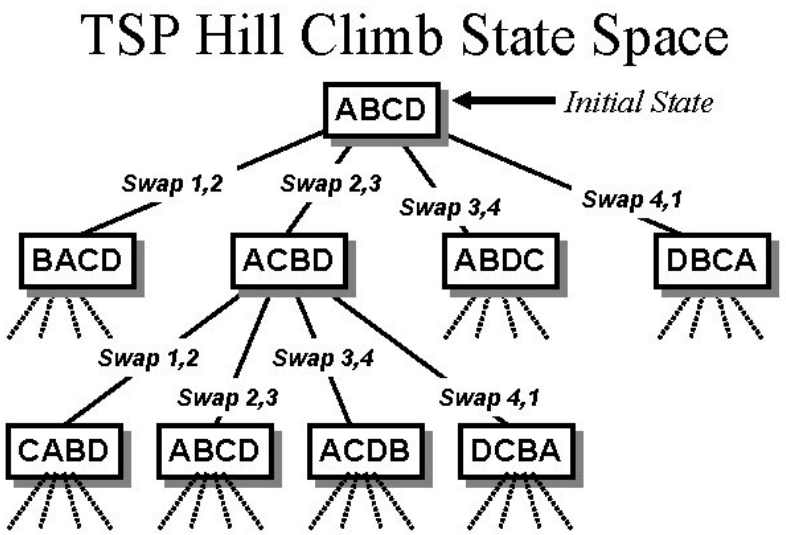
$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\geq g(n) + h(n) \\ &= f(n) \end{aligned}$$

I.e., $f(n)$ is nondecreasing along any path.



TSP


TSP Example
A
B
C
D
6
1
2
5
3
4


TSP Hill Climb State Space
ABCD
Initial State
Swap 1,2
Swap 2,3
Swap 3,4
Swap 4,1
BACD
ACBD
ABDC
DBCA
Swap 1,2
Swap 3,4
Swap 2,3
Swap 4,1
CABD
ABCD
ACDB
DCBA

22

Резюме

- Среди, които са детерминистични, наблюдаеми, статични и напълно известни. В такива случаи агентът може да конструира последователности от действия, които постигат целите му; Този процес се нарича **search**.
- Преди агентът да започне да търси решения, **goal** трябва да бъдат идентифицирани и добре дефинирани **problem** трябва да бъде формулиран.
- Проблемът се състои от пет части: **initial state**, набор от **actions**, **transition model** Описание на резултатите от тези действия, **goal test** функция, и **path cost** функция. Средата на проблема е представена от **state space**. **path** чрез пространството на състоянието от началното състояние до целевото състояние е **solution**.
- Алгоритмите за търсене третират състоянията и действията като **atomic**: Те не вземат предвид никаква вътрешна структура, която биха могли да притежават.
- Базово TREE-SEARCH алгоритъм разглежда всички възможни пътища за намиране на решение, докато а GRAPH-SEARCH алгоритъм избягва разглеждането на излишни пътища.
- Алгоритмите за търсене се оценяват въз основа на **completeness**, **optimality**, **time complexity**, и **space complexity**. Сложността зависи от b, факторът на разклоняване в пространството от състояния, и d, дълбочината на най-плиткото решение.

Резюме

Uninformed search методите имат достъп само до дефиницията на проблема. Основните алгоритми са следните:

- Breadth-first search първо разширява най-плитките възли; Той е пълен, оптимален за разходите за единични стъпки, но има експоненциална пространствена сложност.
- Uniform-cost search разширява възела с най-ниска цена на пътя, $g(n)$, и е оптимално за общите разходи за стъпки.
- Depth-first search разширява първо най-дълбокия неразширен възел. Той не е нито пълен, нито оптимален, но има линейна пространствена сложност. Depth-limited search добавя дълбочина, обвързана с лимита.
- Iterative deepening search извиква търсене в дълбочина с увеличаване на границите на дълбочината, докато не бъде намерена цел. Той е пълен, оптимален за разходите за единични стъпки, има времева сложност, сравнима с търсенето на ширина и има линейна пространствена сложност.
- Bidirectional search може значително да намали сложността на времето, но не винаги е приложима и може да изисква твърде много пространство.

Informed search методи могат да имат достъп до heuristic функция $h(n)$ която оценява цената на решение от n.

- Генеричният best-first search алгоритъм избира възел за разширение в съответствие с evaluation function.
- Greedy best-first search разширява възли с минимални $h(n)$. Тя не е оптимална, но често е ефективна.
- A* търсене разширява възли с минимални f(n)= g(n)+ h(n). A* е пълна и оптимална, при условие че h(n) е допустимо (for TREE-SEARCH) или последователни (for GRAPH-SEARCH). Пространствената сложност на A* все още е непосилна.
- RBFS (recursive best-first search) and SMA* (simplified memory-bounded A*) са стабилни, оптимални алгоритми за търсене, които използват ограничени количества памет; Ако им се даде достатъчно време, те могат да решат проблеми, които A* не може да реши, защото паметта му свършва.

Изпълнението на евристични алгоритми за търсене зависи от качеството на евристичната функция. Понякога може да се конструира добра евристика, като се намалят ограниченията дефинирането на проблема, като се съхраняват предварително изчислените разходи за решаване на подзадачи в база данни с шаблони или като се учат от опита с проблемния клас.