Информирано пребарување

Што е информираното пребарување?

- Голем број од алгоритмите за слепо пребарување се комплетни, но ...
- обемот на просторот на пребарување ги прави прескапи или неприменливи.
- Решенија:
  - Воведување редослед на развивање на јазлите со помош на функција за проценка (evaluation function) f(n) која зависи од јазелот n
  - Воведување на хеуристичка функција (heuristic function) h(n) која ја проценува најниската цена од јазелот n до крајниот јазел

Кои се најпознатите алгоритми за информирано пребарување (1)?

- Пребарување “првиот најдобар” (Best-first search) (или “прво по најдобриот”)
- Алчен “првиот најдобар” (Greedy Best-first search)
- A* пребарување (A-star search)

Кои се најпознатите алгоритми за информирано пребарување (2)?

- Хеуристички (heuristic search):
  - RBFS (Recursive Best-First Search)
  - SMA* (Simplified Memory Bounded A*)
  - Насочено како зрак (Beam search)
- Локално пребарување (local search):
  - Искачување по рид (Hill climbing)
  - Симулирано калење (Simulated annealing)

Најдобриот прв (Best-first search)

- Користејќи функција за проценка f(n) за секој јазел поодделно, развивај го најпожелниот (т.е. најдобриот) јазел во непосредна близина
- Во суштина, ова значи подредување на јазлите во опаѓачки редослед на нивната пожелност
- Варијации на оваа тема:
  - Алчен (лаком) алгоритам (greedy best-first search)
  - A ѕвезда (A* search)

Врз основа на што се одредува разложувањето?

- Greedy search (алчно пребарување) – пребарува врз основа на хеуристичката функција h(n)
- A* search, пребарува според f(n)=g(n)+h(n)

Значењето на g(n) и h(n)

- g(n) – фактичката цена на чинење од почетната состојба до состојбата **n**

  (ОВА СЕ ЗНАЕ, БИДЕЈЌИ ВЕЌЕ СМЕ СТИГНАЛЕ ДО n И ЗНАЕМЕ ШТО И КАКО СМЕ ПОМИНАЛЕ)

- h(n) – проценета цена на чинење на најевтиниот пат од состојбата **n** до целта

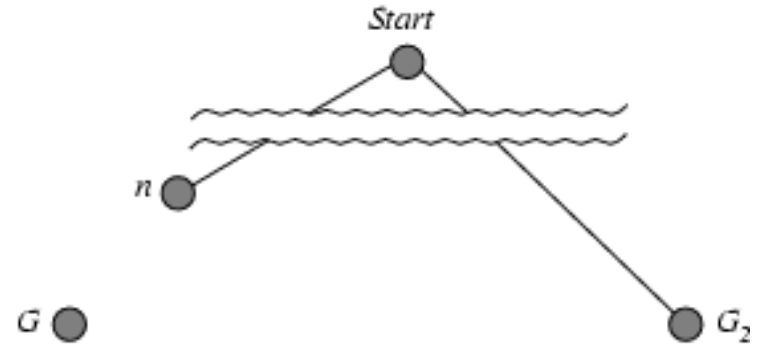
  (ОВА ЕДИНСТВЕНО МОЖЕ ДА СЕ ПРОЦЕНИ, БИДЕЈЌИ НЕ ГО ЗНАЕМЕ ТОЧНО ПАТОТ ОД n ДО ЦЕЛТА, МОЖЕ САМО ДА СЕ НАДЕВАМЕ ДЕКА ПРОЦЕНКАТА НИ Е ДОБРА)

Критериуми за избор на хеуристичката фунцкција

- Хеуристичката функција е прифатлива (admissible) ако:
  - За секој јазел n, $h(n) \leq h^*(n)$, каде $h^*(n)$ е вистинската цена за да се стигне до целта
  - Прифатливата хеуристика не ја преценува вистинската цена (на пр. воздушната линија не е никогаш подолга од вистинската)
- Теорема*: Ако h(n) е прифатлива, пребарувањето на дрво со A^* е оптимално.

Доказ на оптималноста на A* *

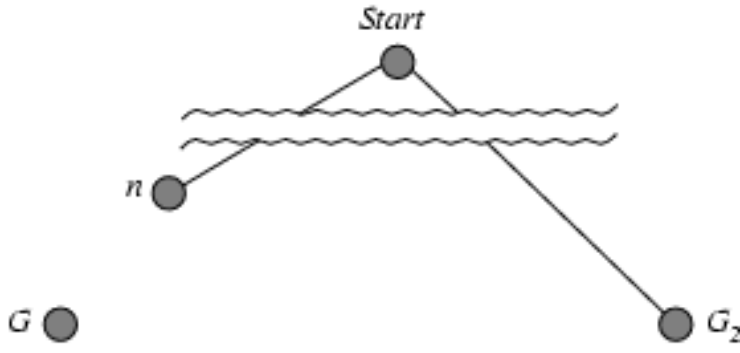
- Нека е генерирана некоја цел G_2 која не е оптимална. Нека n е таков неразложен јазел во околината, којшто лежи на оптималниот пат G.
- $f(G_2) = g(G_2)$ бидејќи $h(G_2) = 0$
- $g(G_2) > g(G)$ бидејќи G_2 не е оптимално
- $f(G) = g(G)$ бидејќи $h(G) = 0$
- Значи, $f(G_2) > f(G)$



Продолжение на доказот *

- $h(n) \leq h^*(n)$ бидејќи h е прифатлива
- $g(n) + h(n) \leq g(n) + h^*(n)$
- $f(n) \leq f(G)$

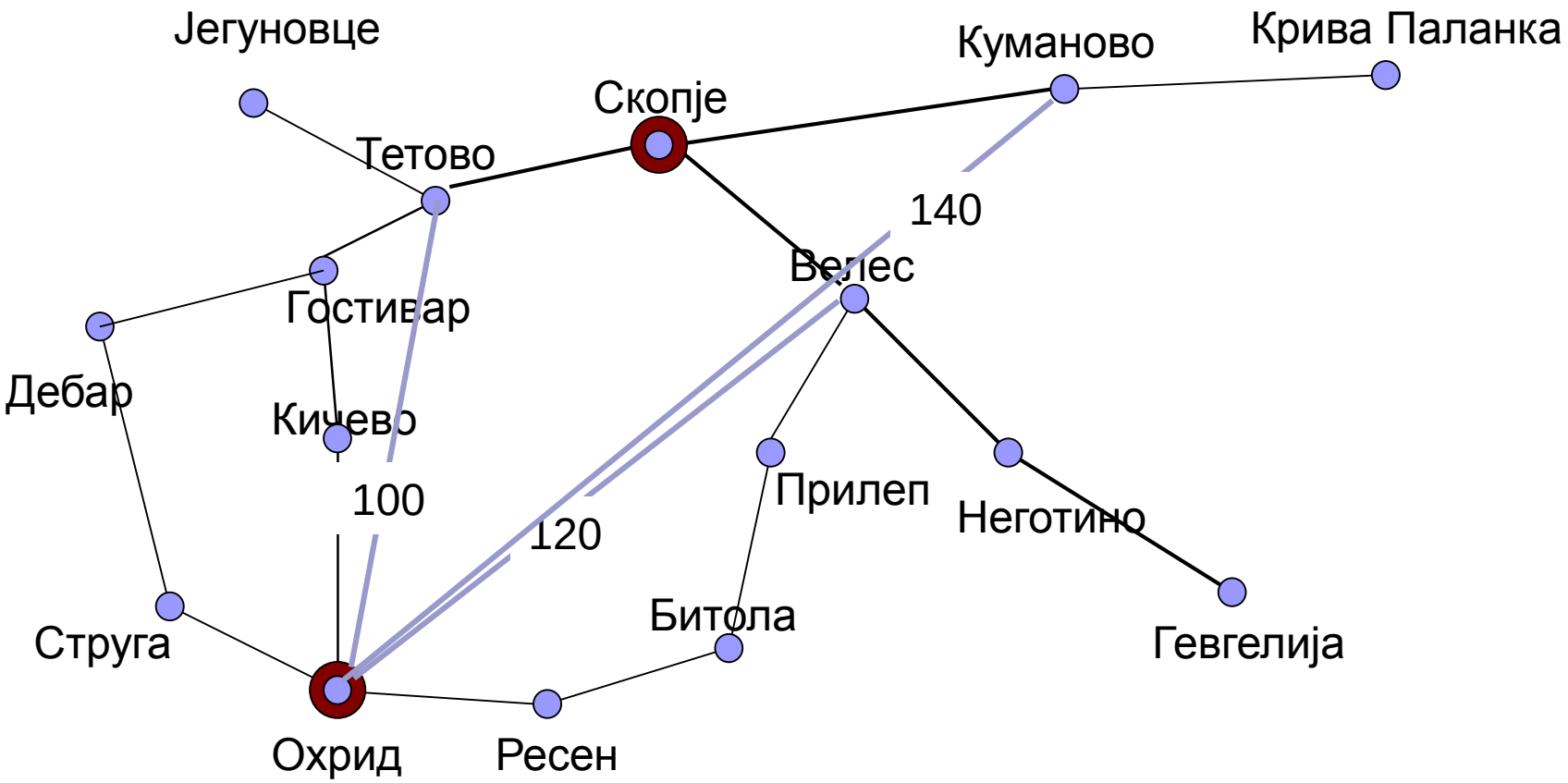
Според тоа што $f(G_2) > f(n)$, A^* никогаш нема да го одбере G_2 за разложување



Текстуална илустрација на најдобриот прв

- Функција на проценка h(n): воздушната линија од јазелот што треба да се развива до ЦЕЛТА, т.е. до Охрид
  - Скопје → Тетово (видете го следниот слајд)
  - Тетово → Јегуновце
  - Јегуновце → Тетово
- Ако Јегуновце се отстрани од картата, тогаш патот е оптималниот
- Врз основа на h(n) **не се проценува ниту точната цена, ниту точното растојание**.

Наоѓање на патот меѓу Скопје и Охрид



Алчен “првиот најдобар”

- Критериумот за избор останува непроменет, при што цената f(n) = h(n)
- Во конкретниов случај:

  $$f(n) = h_{нвл}(n)$$

  каде нвл ја означува најкусата воздушна линија
- Ова е добра хеуристика, но дури и ако се стигне до целта, f(n) **не** е ниту вистинската цена, ниту вистинската должина на патот.

Својства на алчниот алгоритам

- Не е комплетен, затоа што пак може да влезе во јамка (примерот со Тетово и Јегуновце)
- И временската и просторната сложеност се $O(b^m)$, бидејќи сите јазли се чуваат во меморија
- И конечно, не е оптимален.

A* - минимизација на целосната цена

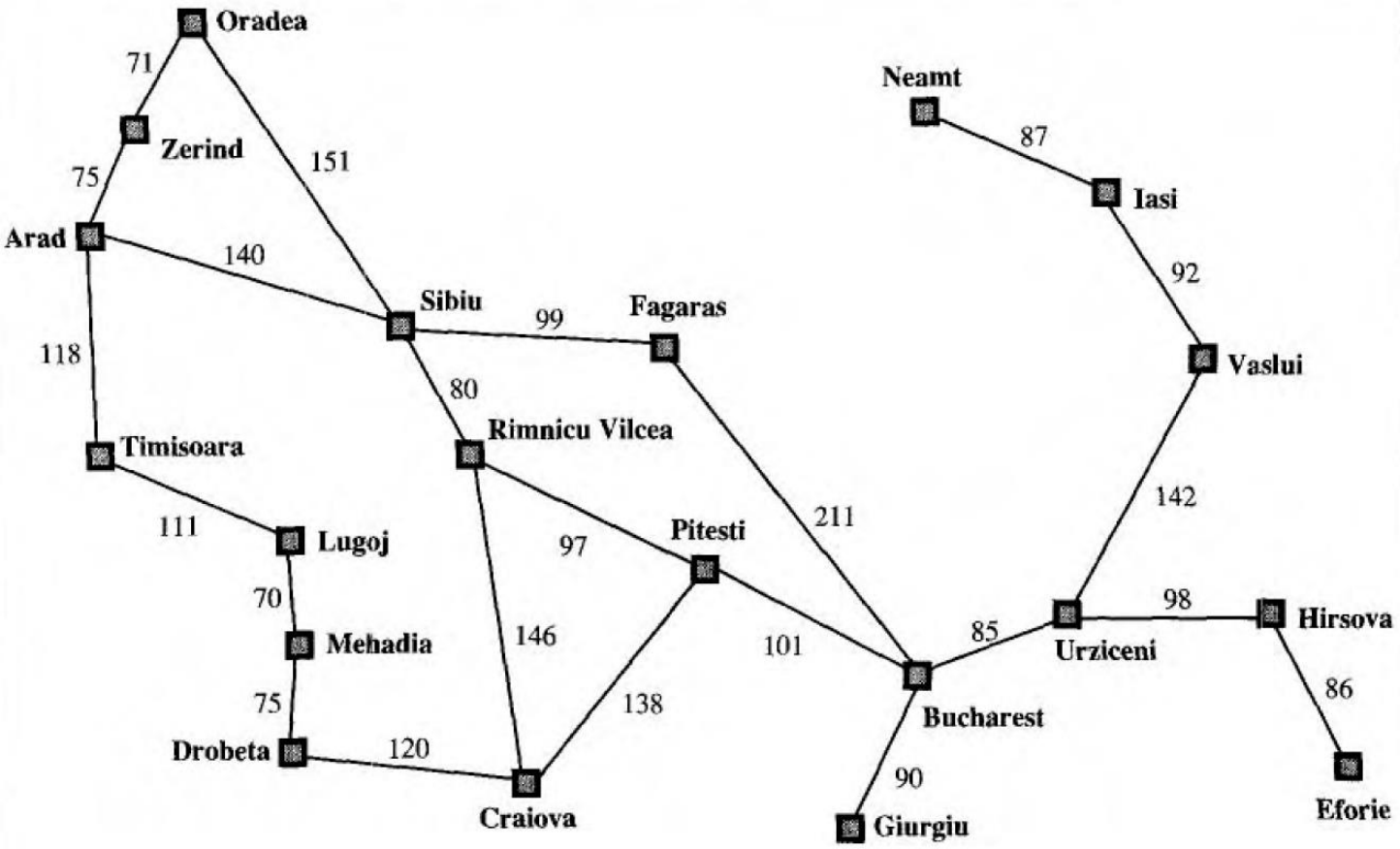
- f(n) = g(n) + h(n)
- g(n) – цена да се стигне до јазелот што е на самиот пат
- h(n) – хеуристика сврзана со проценката од јазелот до крајниот јазел

- Алгоритамот е комплетен и оптимален и ја дава **точната** цена, бидејќи на крајот h(n) = 0.

Пример за A* (1)

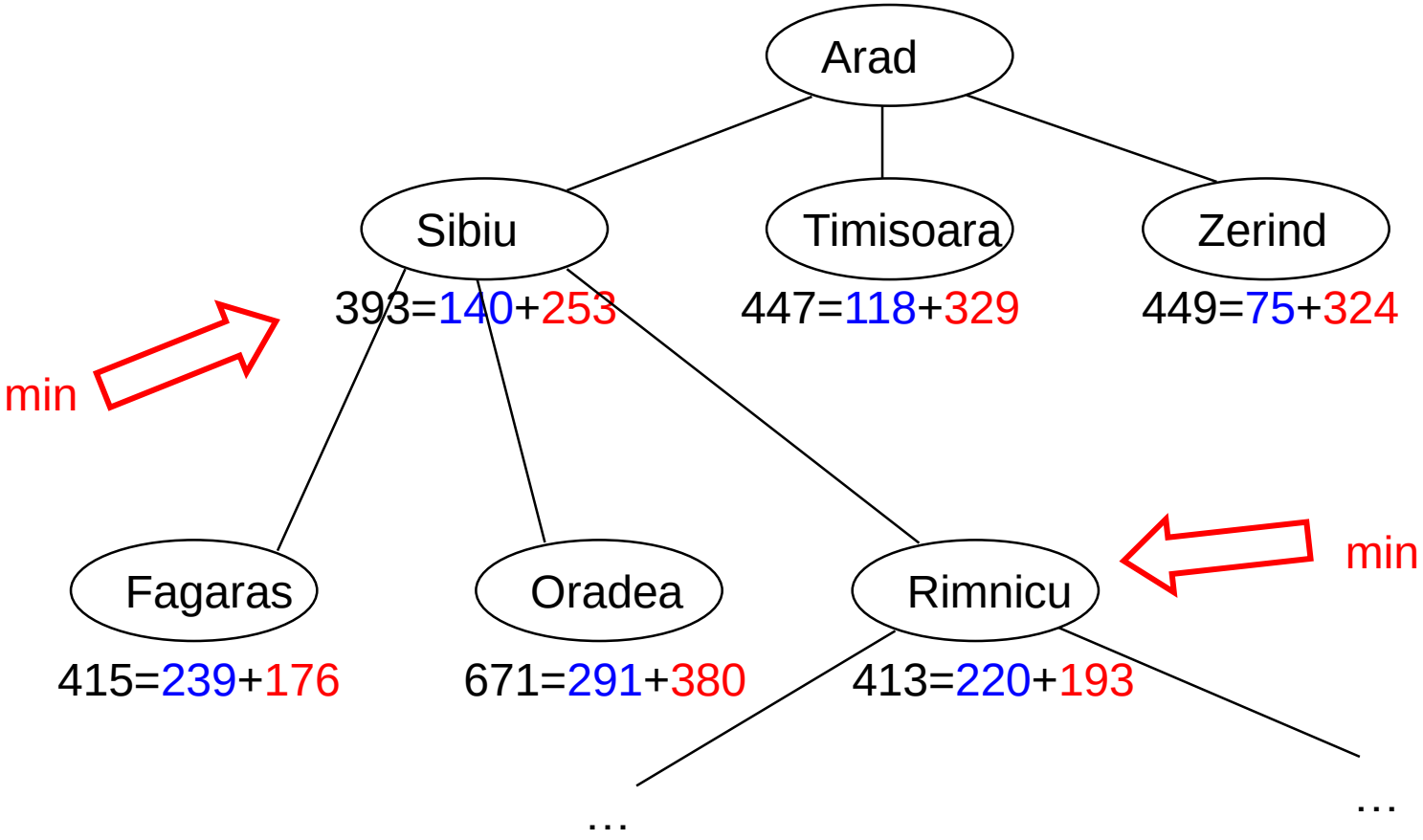
- Барање на најкраток пат од Арад до Букурешт

нвл ја означува најкусата воздушна линија од даден град до Букурешт



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

Пример за A* (2)



Но споредете ги h(n)=176 за Faberas и h(n)=193 за Rimnicu!
Ако одеше по алчност (по нвл), ќе тргнеше за Faberas, а не за Rimnicu

Проблеми на A* алгоритамот

- Преголема комплексност која се должи на меморирањето на сите јазли
- Опасност од комбинаторна експлозија
- Решение – мемориски ограничено хеуристичко пребарување:
  - A* алгоритам со итеративно зголемување на длабочината
  - Рекурзивен A* алгоритам
  - Алгоритам кој учи како да пребарува подобро

Итеративно зголемување на длабочината IDA*

- IDA* = Iterative deepening A* (Korf 1985)
- Се дефинира ценовен прекин (cost cut-off):

$$\hat{f}(n_0) = \hat{g}(n_0) + \hat{h}(n_0) = \hat{h}(n_0)$$

каде n_0 е почетниот јазел

Развивање на јазлите кај IDA*

- Јазлите се развиваат по длабочина, секогаш кога вредноста на следбеникот не го надминува ценовниот прекин.
- Ако со овој пристап се стигне до целта, тогаш е најден оптималниот пат.
- Во спротивно, вредноста на ценовниот прекин се зголемува и пребарувањето почнува одново.

RBFS – Recursive Best-First Search

- Варијанта на IDA*
- Ја следи (keeps track) вредноста f на најдобриот **алтернативен пат**
- Ако со тековниот јазел се надминува границата, рекурзивно се олеснува критериумот и се преминува кон најблискиот алтернативен пат.
- Со постепено олеснување, вредноста f на секој јазел се заменува со најдобрите вредности f на неговите деца.
- Тоа значи дека алгоритамот ги помни само вредностите на најдобрите листови на заборавеното поддрво, а подоцна одлучува дали да го разложи или не.

RBFS алгоритам (1)

- Се дефинира резервна (backed-up) вредност $\hat{f}(m)$ според формулата:

$$\hat{f}(m) = \min_{m_i} \hat{f}(m_i)$$

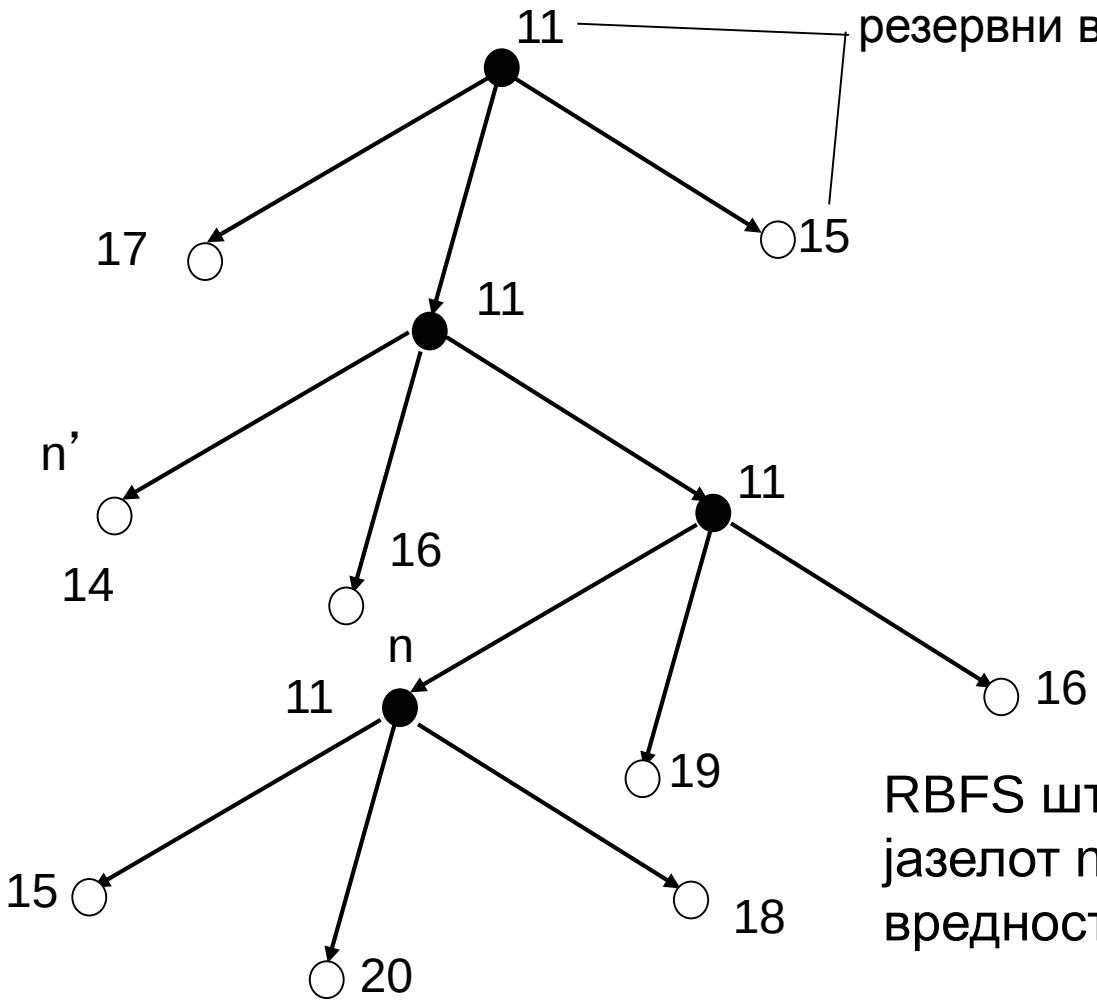
каде $\hat{f}(m)$ е резервната вредност за јазелот m_i

- Ако некој од следбениците на јазелот n кој штотуку бил разложен има помала вредност од $\hat{f}$ за сите разложени јазли, тогаш тој станува следен кандидат за разложување.

RBFS алгоритам (2)

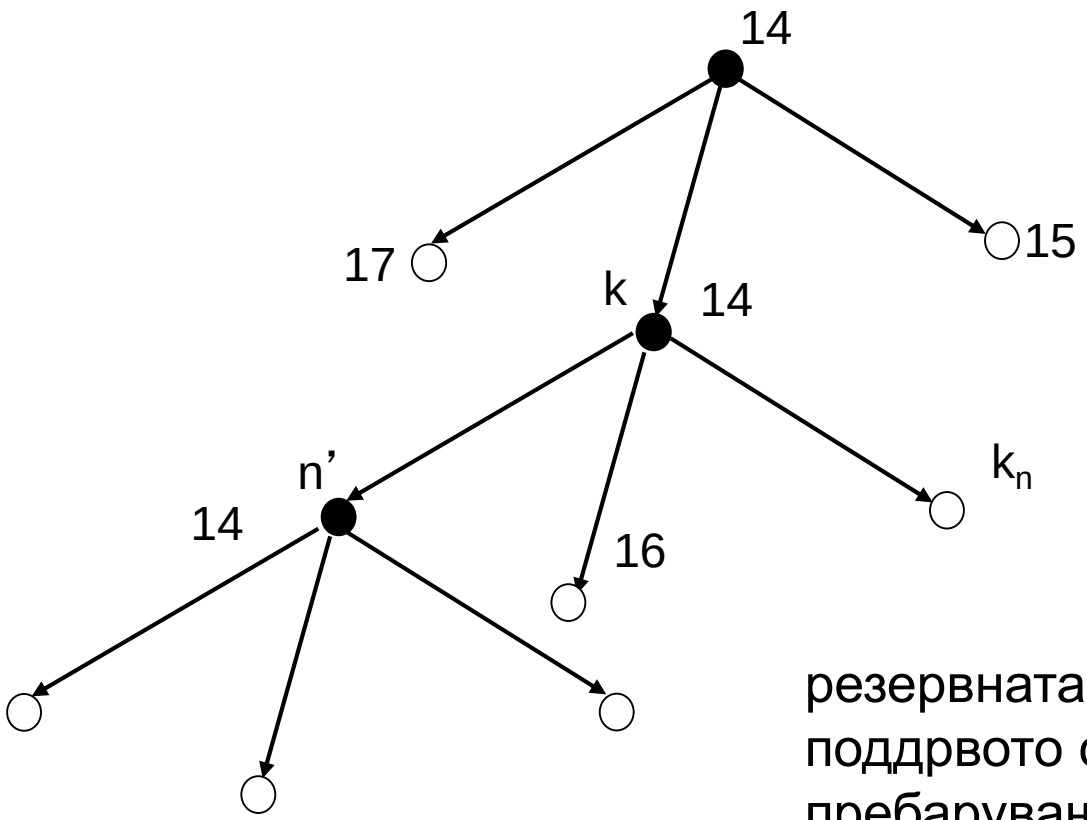
- Тогаш, алгоритамот го наоѓа најнискиот заеднички претходник (родител) меѓу n и n’
- Овој родител го означуваме со k
- Следбеникот на k на патот кон n нека е k_n
- Тогаш алгоритамот го отстранува поддрвото чиј корен е k_n, а k_n станува отворен јазел со вредност иста со вредноста на $\hat{f}$ која е добиена одејќи наназад.
- Пребарувањето продолжува под отворениот јазел.

Илустрација преземена и прецртана од Nils J. Nilsson



RBFS штотуку го разложил јазелот n, но не создал резервна вредност за наследниците

Илустрација преземена и прецртана од Nils J. Nilsson



резервната вредност е создадена, поддрвото се отсекува и пребарувањето продложува под n'

Проблеми на мемориски ограничените алгоритми

- Премалку меморија
- IDA* меѓу итерациите чува само една вредност – тековниот лимит на цената
- RBFS има комплексност O(bd), но таа реално не се искористува

- Решение: упростен мемориски ограничен A* алгоритам (SMA* = Simplified Memory-bounded A*)

Упростен мемориски ограничен A* алгоритам (SMA*)

- SMA* (Simplified Memory-bounded A*)
- Се однесува како A*, разложувајќи го најдобриот лист додека меморијата е полна.
- SMA* го отстранува најлошиот јазел (јазелот со најголема вредност f)
- Вредноста на заборавениот јазел се резервира
- Следбеникот на заборавеното поддрво го знае најдобриот пат во своето поддрво.
- Благодарение на тоа, SMA* го обновува заборавеното поддрво, само ако се покаже дека сите останати патишта изгледаат полошо.

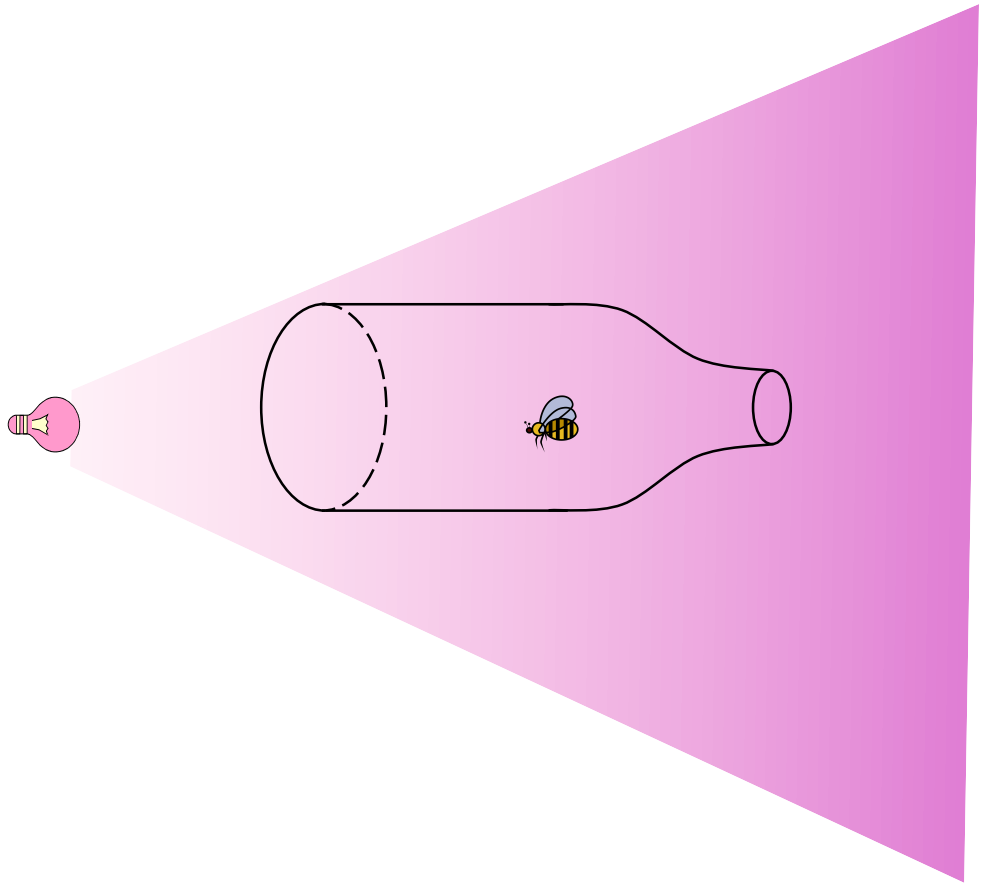
Beam search (насочено пребарување како зрак)

- Наместо да ги подредува сите јазли по функцијата за процена па по тој редослед да ги разложува сите, ги зема предвид само првите n неколку јазли (пример само првите два “најветувачки” јазли, за n=2) и само нив ги разложува натаму

Искачување кон врвот на ридот (hill climbing)

- Искачувањето кон врвот е техника за оптимизација на **локалното пребарување**.
- Се применува кога постојат поголем број решенија, при што го понудува решението што е најблиско до оптималното, со неспоредливо помал напор.
- Се базира на градиентна функција за проценка
- Пример: проблемот на трговскиот патник

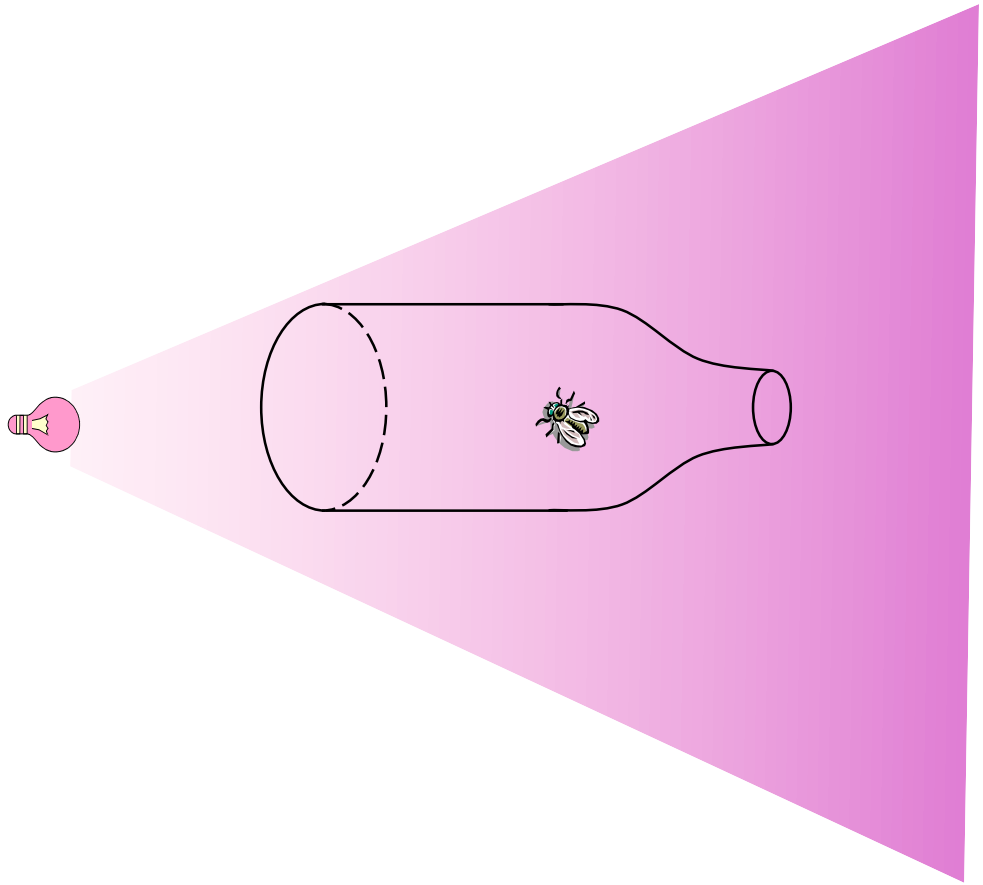
Пример за пребарување според градиентот на евалуацискатa ф-ја



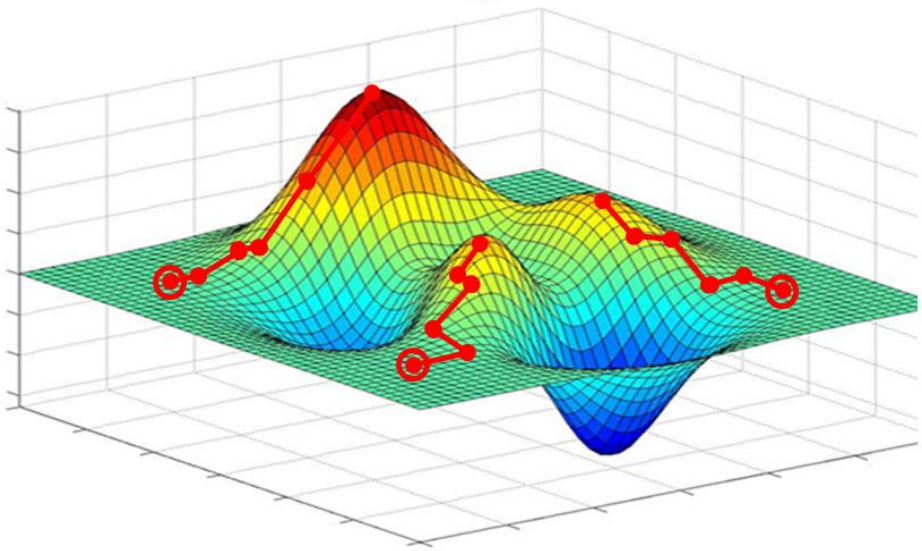
Симулирано калење (simulated annealing search)

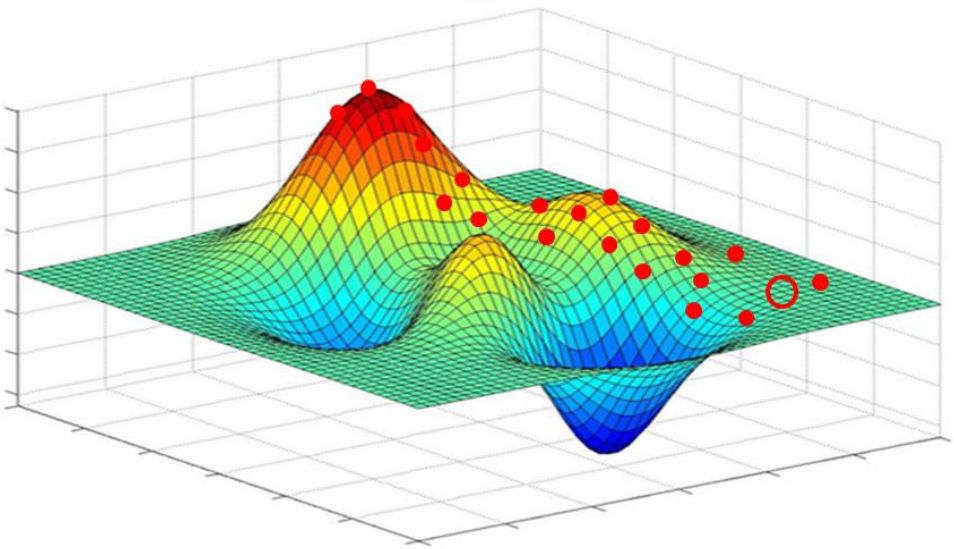
- Искачувањето кон врвот не гарантира оптималност, бидејќи може да застане во некој локален минимум.
- Симулираното калење го надминува овој проблем, користејќи случајно поминување во некое поддрво кон некој од следбениците.
- Овој алгоритам е комплетен, но е крајно неефикасен.

Пример за пребарување базирано на случаен избор (симулирано калење)



Споредба на искачување кон врвот на ридот со некои стохастички методи (симулирано калење, генетски алгоритми, ...)





Специјални пребарувања

- Апроксимативно пребарување
  - Пребарување базирано на острови
  - Хиерархиско пребарување
  - Пребарување ограничено со хоризонт
- Учење хеуристики
  - Експлицитни графови
  - Имплицитни графови
- Пребарување кое не се стреми кон целта, туку кон награди

Користена литература

Првенствено книгите:

- Artificial Intelligence, A Modern Approach 2nd edition, Russel and Norvig
- Artificial Intelligence, A New Synthesis, Nils J. Nilsson

но исто така и статиите кои потекнуваат од основната статија за Search Algorithms во англиската Википедија

Прашања?