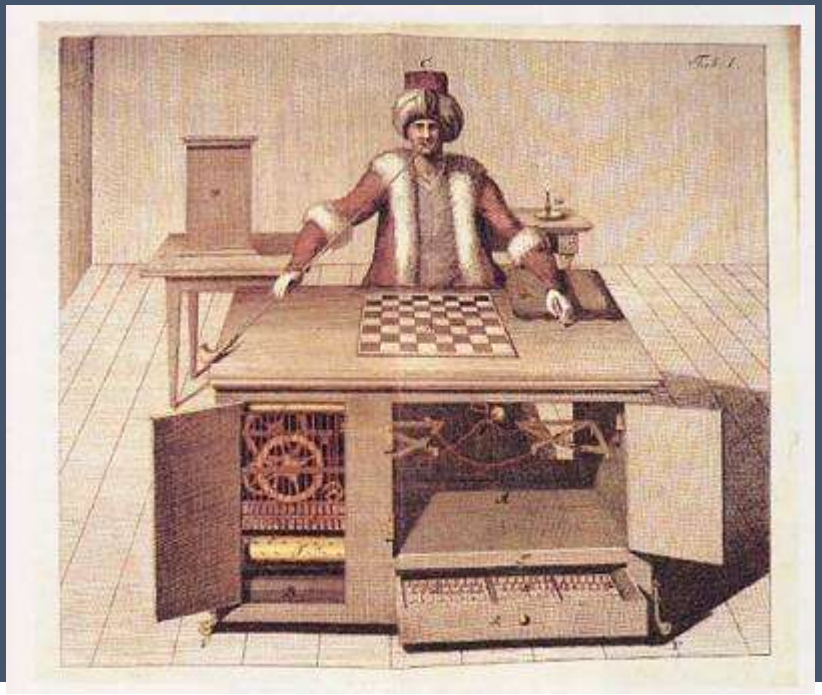
VEŠTAČKA INTELIGENCIJA 2020/21

REŠAVANJE PROBLEMA I TRAŽENJE: INFORMISANI ALGORITMI

Sadržaj

- Best-first
  - Greedy Best-first
  - A*
- Local search algoritmi
  - Hill climbing
  - Simulated anealing



Prof. dr Leonid Stoimenov

Informisano (heurističko) traženje

Heuristika: Omogućava da se izabere čvor koji više obećava od ostalih

Prihvatljiva heuristika: predvidjena cena nikada nije veća od stvarne

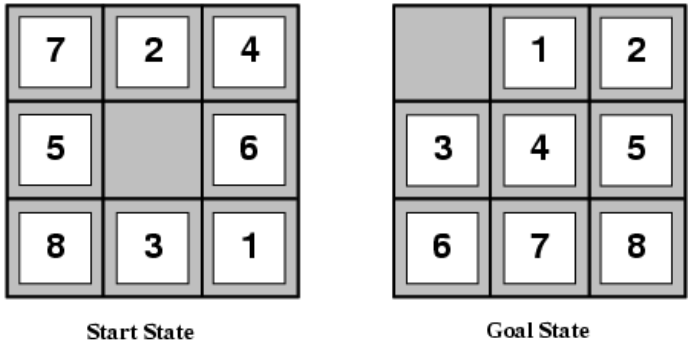
- h(n) – procenjena cena puta od tekućeg čvora do cilja
- h(G)=0 za Ciljni čvor
- h(n)>0 za ostale čvorove

Primeri heuristickih funkcija za 8-puzzle

Dve prihvatljive heuristike:

- h1 = broj pločica na pogrešnom mestu
- h2 = suma razdaljina svih pločica od njihovih krajnjih pozicija (city block distance, Manhattan distance)
- Izabrati onu koja je bolja

za traženje (smanjuje broj koraka)



Start State Goal State

- $h_1(S) = 8$
- $\mathbf{h_2(S)} = 3+1+2+2+2+3+3+2 = 18$

h2(n)≥h1(n) zato je h2(n) bolja heuristika

Opšti algoritam za traženje

```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST[problem] applied to STATE(node) succeeds return node
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

Informisani algoritmi:
Način dodavanja i obrade čvorova iz pomoćne strukture *fringe*

Algoritam prvi najbolji
Best-first search

- Kompletan, nije optimalan generalno
- Treba odrediti ***h(n)***
- Ideja: koristiti evaluacionu funkciju *f(n)* za svaki čvor
  - f(n) obezbeđuje **procenjenu totalnu cenu puta.**
  - → Čvor za ekspanziju se bira na osnovu minimalne vrednosti za h(n)
- **Implementacija:**

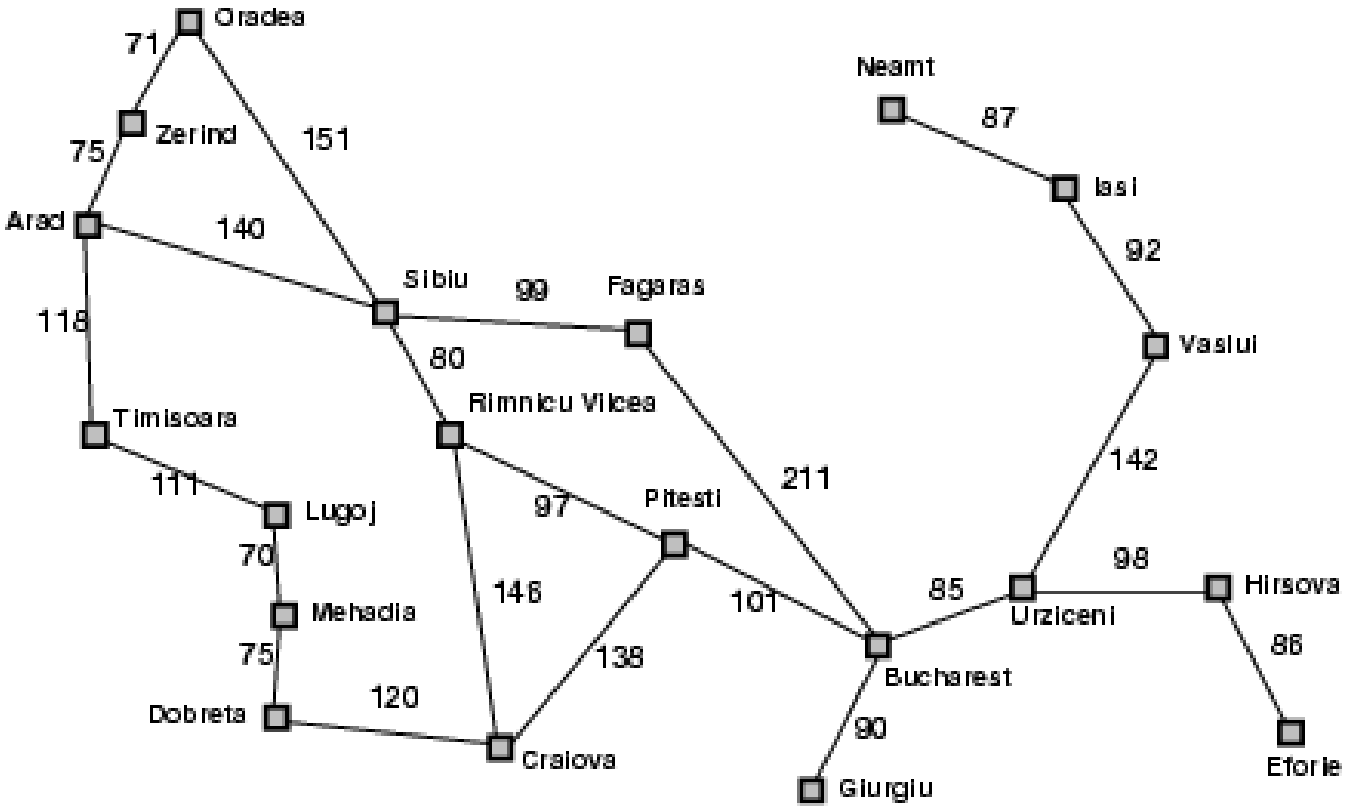
  Urediti čvorove po vrednosti cene puta – **sortirani red.**
- Specijalni slučajevi:
  - **Greedy best-first** traženje – ekspanzija čvora koji izgleda da je najbliži cilju
  - **A*** traženje

Implementacija Best-first search

Implementacija: sortirani red

1. Formirati listu čvorova koja inicijalno sadrži samo startni čvor.
2. Dok se lista čvorova ne isprazni ili se ne dođe do ciljnog čvora, proveriti da li je prvi element liste ciljni čvor
   a) Ako je prvi element liste ciljni čvor, ne raditi ništa.
   b) Ako prvi element liste nije ciljni čvor, ukloniti ga iz liste i dodati njegove sledbenike iz stabla pretrage (ako ih ima i ako nisu već posećeni) u listu.
   Celokupnu listu sortirati po rastućim vrednostima heurističkih funkcija čvorova.
3. Ako je pronađen ciljni čvor, pretraga je uspešno završena; u suprotnom pretraga je neuspešna.

Rumunija, sa cenom puta u kilometrima

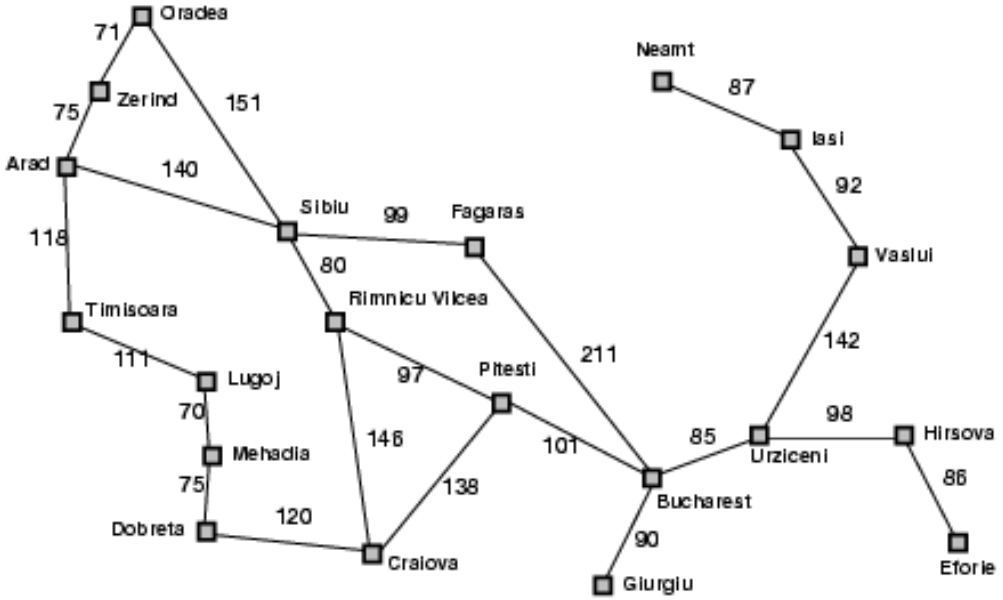


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first traženje

- Evaluaciona funkcija $f(n) = h(n)$ (heuristika)
  = procenjena cena on n do *cilja*
- Greedy best-first traženje vrši ekspanziju čvora koji izgleda da je najbliži cilju
- Primer, $h(n)$ = pravolinjsko rastojanje od n do Bucharest-a

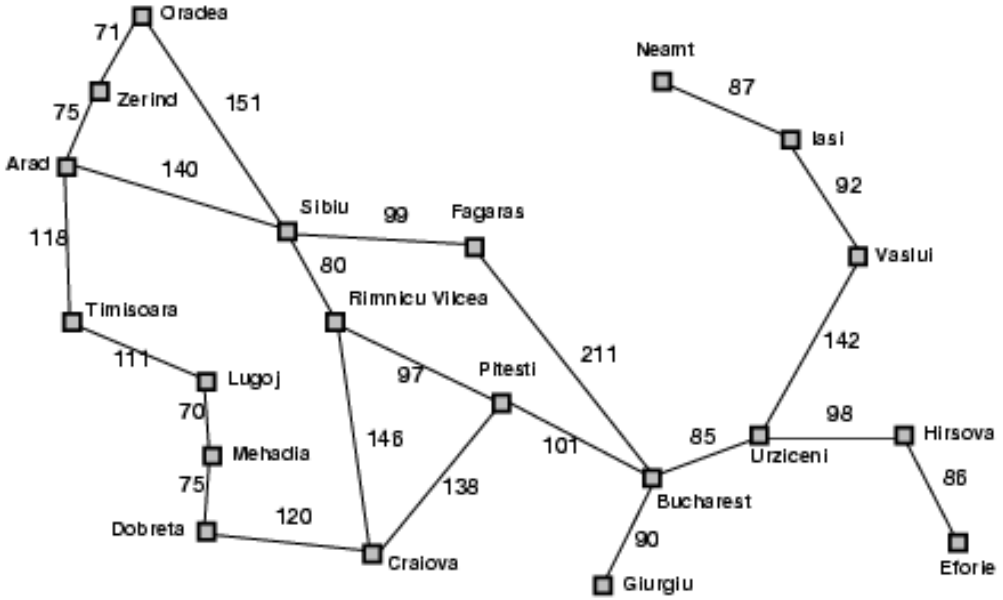
Primer: Greedy best-first search



Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

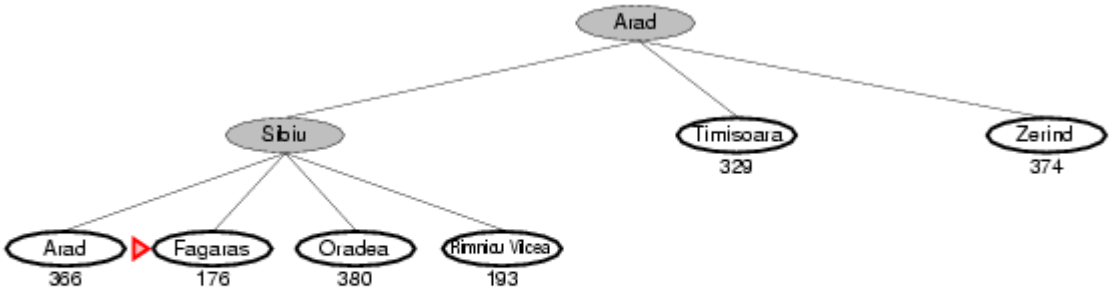
Primer: Greedy best-first search

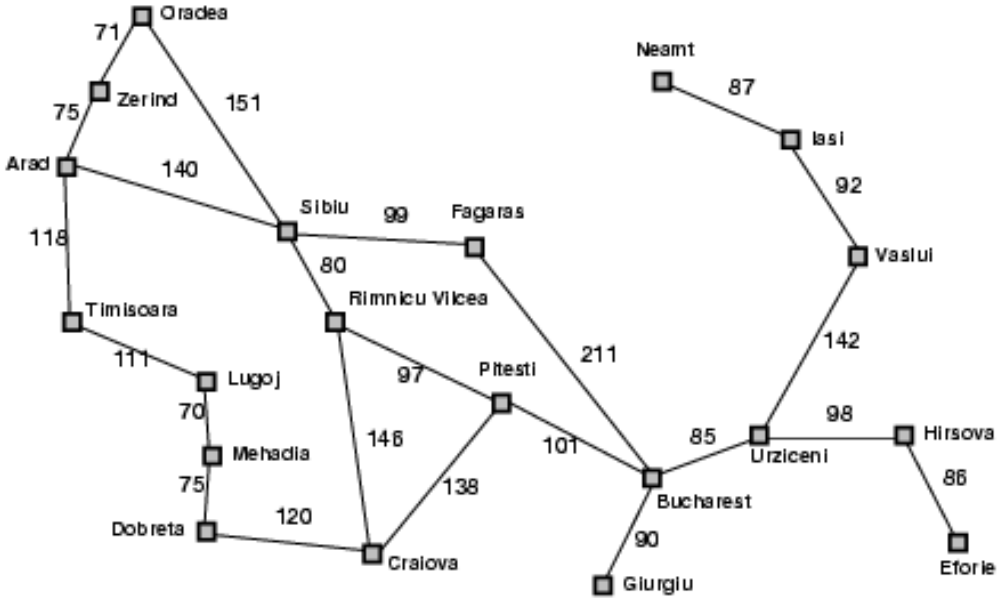


Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Primer: Greedy best-first search

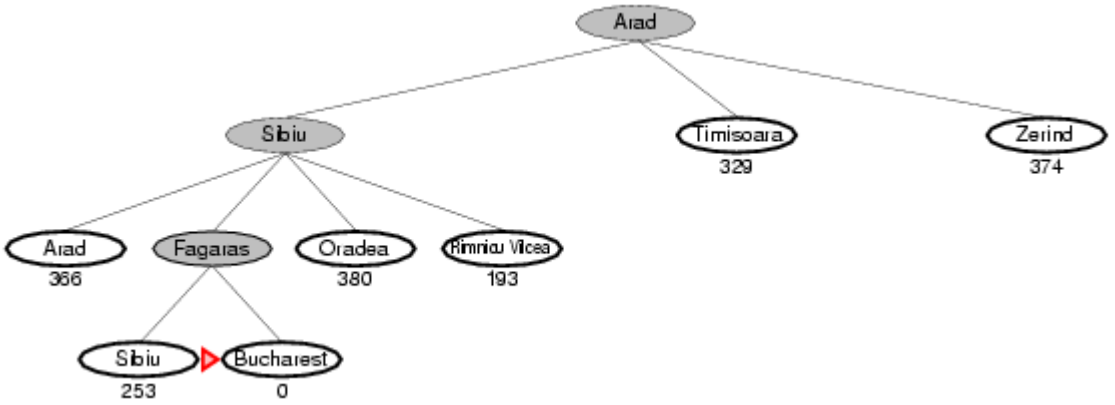


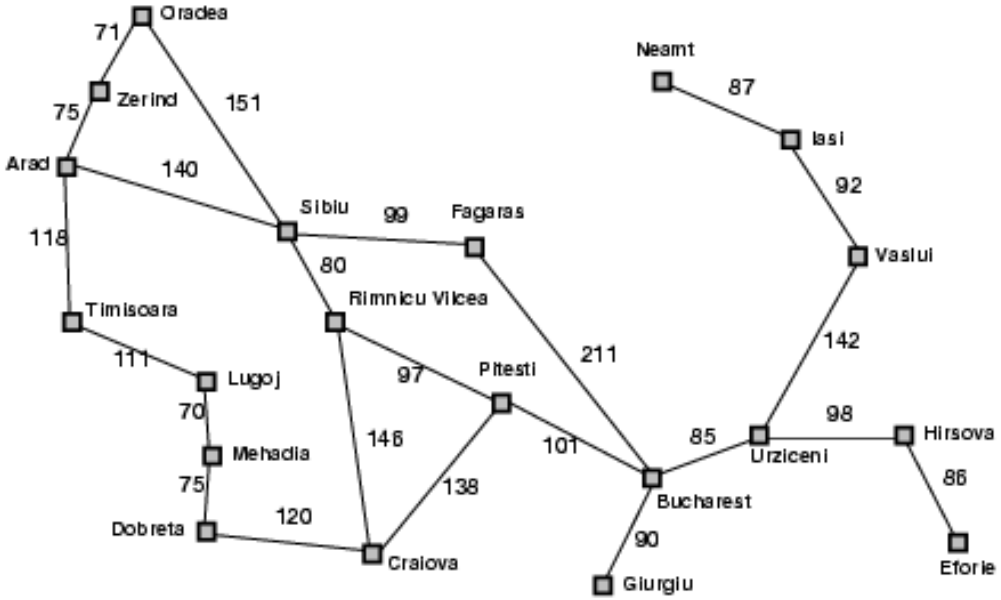


Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Primer: Greedy best-first search





Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

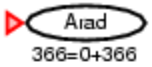
Osobine Greedy best-first traženja

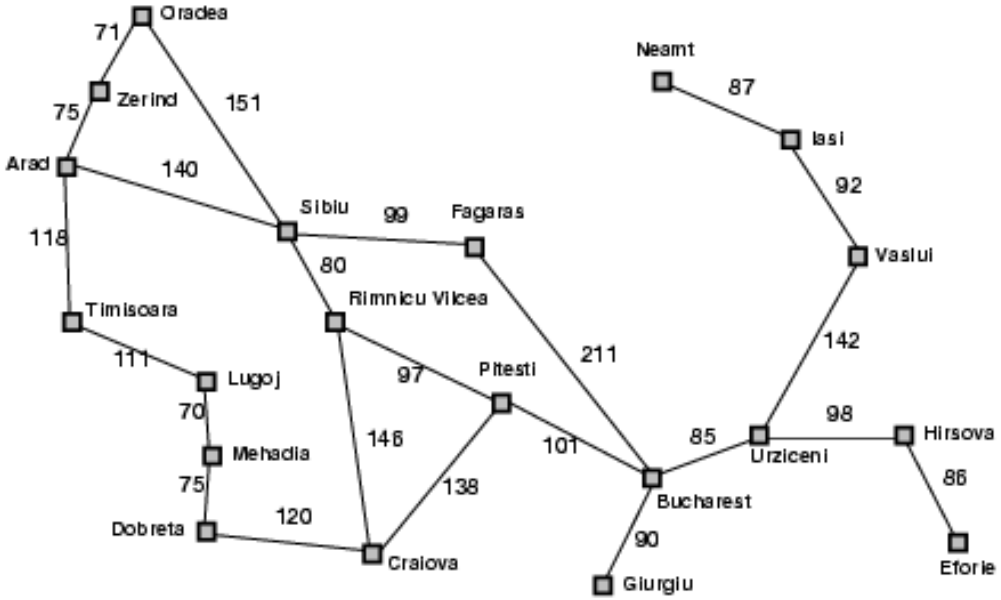
- Kompletan? Ne – može da se zaglavi u petljama, napr, Iasi → Neamt → Iasi → Neamt →
- Vreme? $O(b^m)$, ali sa dobrom heutistikom može da se drastično poboljša
- Prostor? $O(b^m)$ – drži sve čvorove u memoriji
- Optimalan? Ne

Optimalni algoritmi traženja
A* algoritam

- Izbegava puteve koji su već skupi
- Kombinuje stvarnu cenu putanje **g(n)** sa predvidjenom cenom **h(n)**
- Ukupna cena **f(n)=g(n) + h(n)**
  - $g(n)$ = cena dostizanja n
  - $h(n)$ = procenjena cena od n do cilja
  - $f(n)$ = procenjena ukupna cena puta od n do cilja
  - Best First search: $f(n)=h(n)$
- **Implementacija:** kao kod Best-first, svi čvorovi u redu se **sortiraju**, u ovom slučaju po vrednosti **f(n)**

Primer: A* traženje

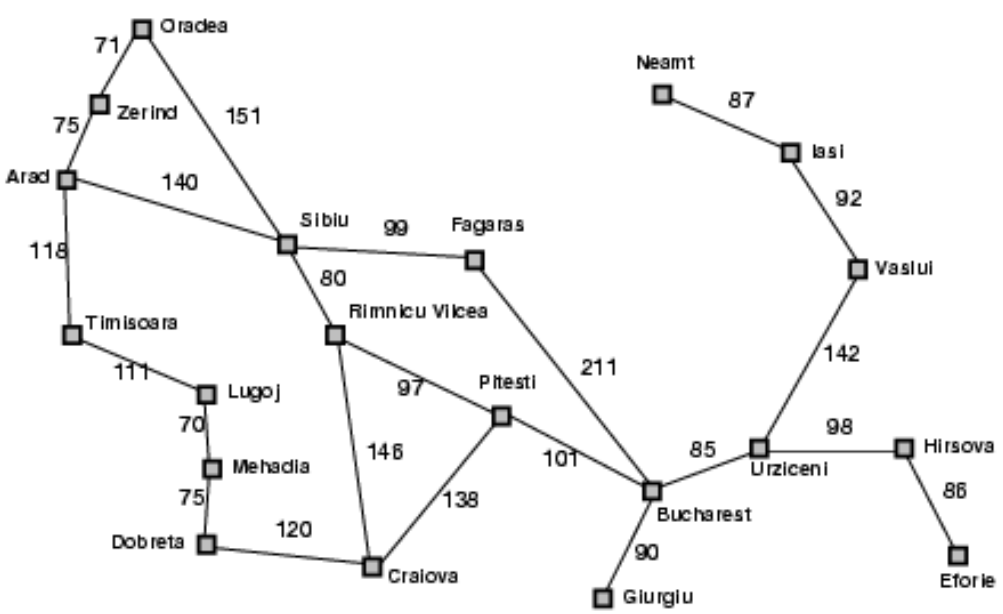




Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

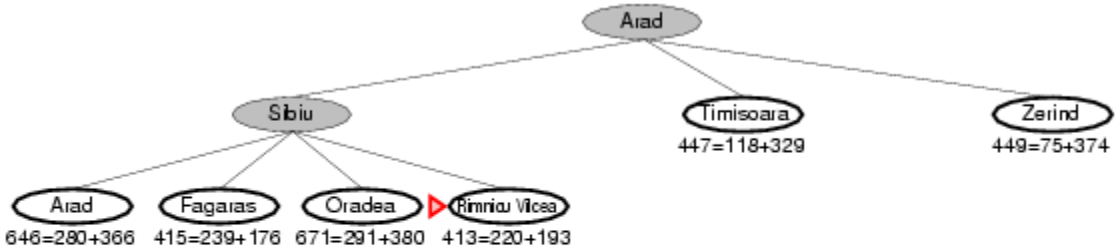
Primer: A* traženje

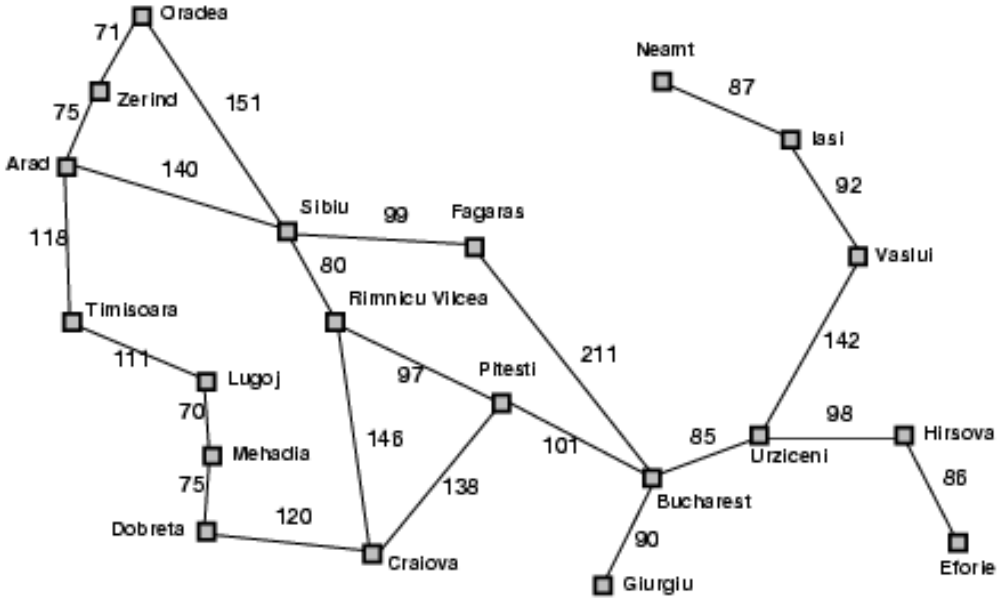


Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Primer: A* traženje

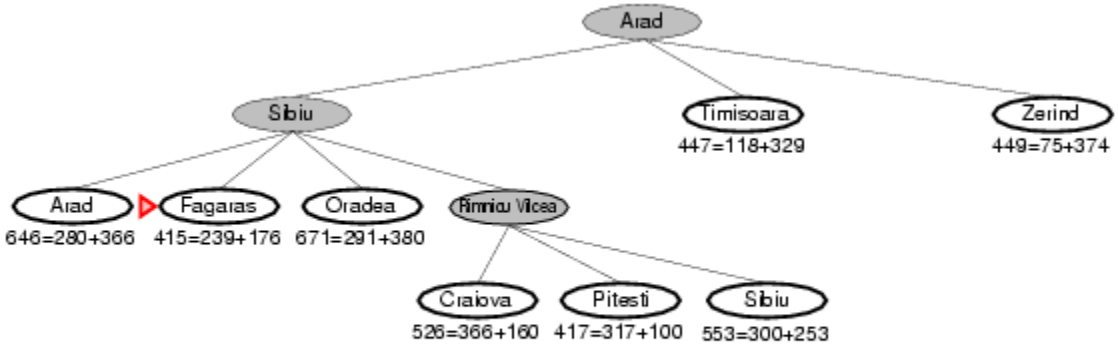


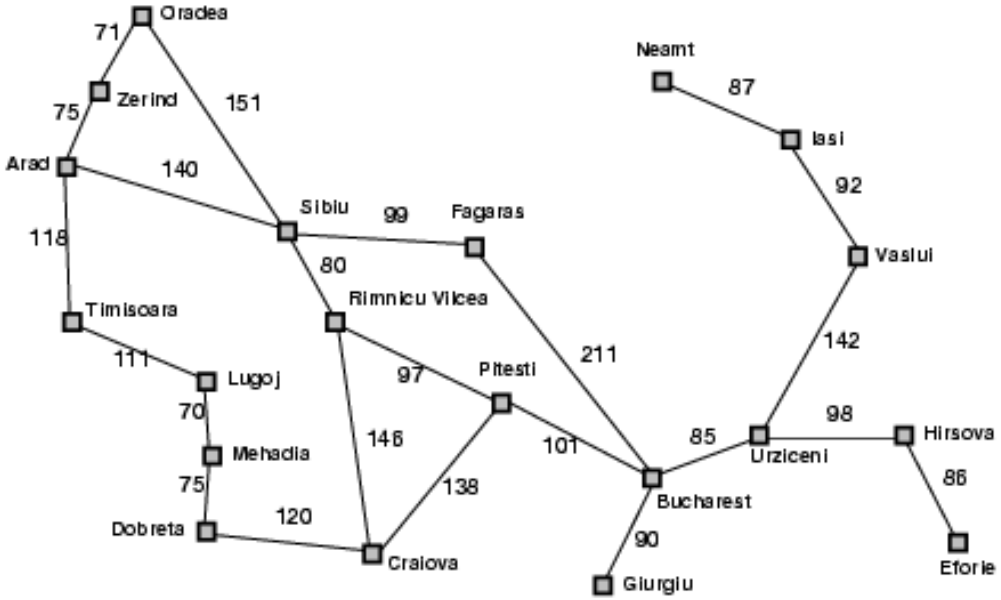


Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Primer: A* traženje

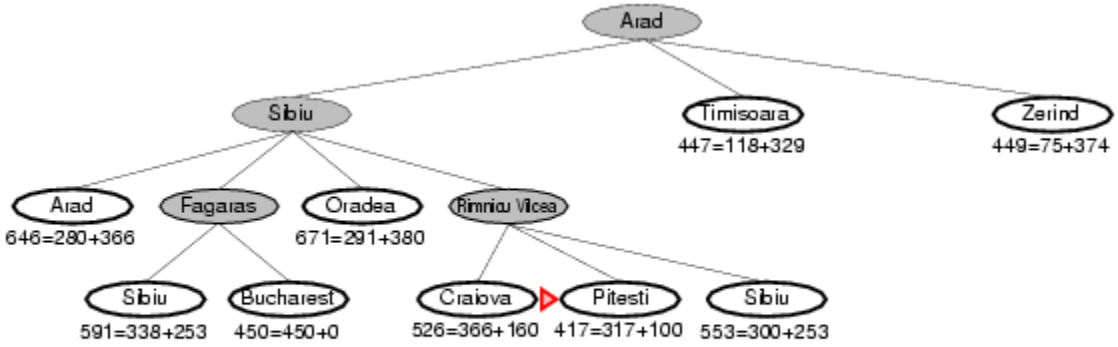


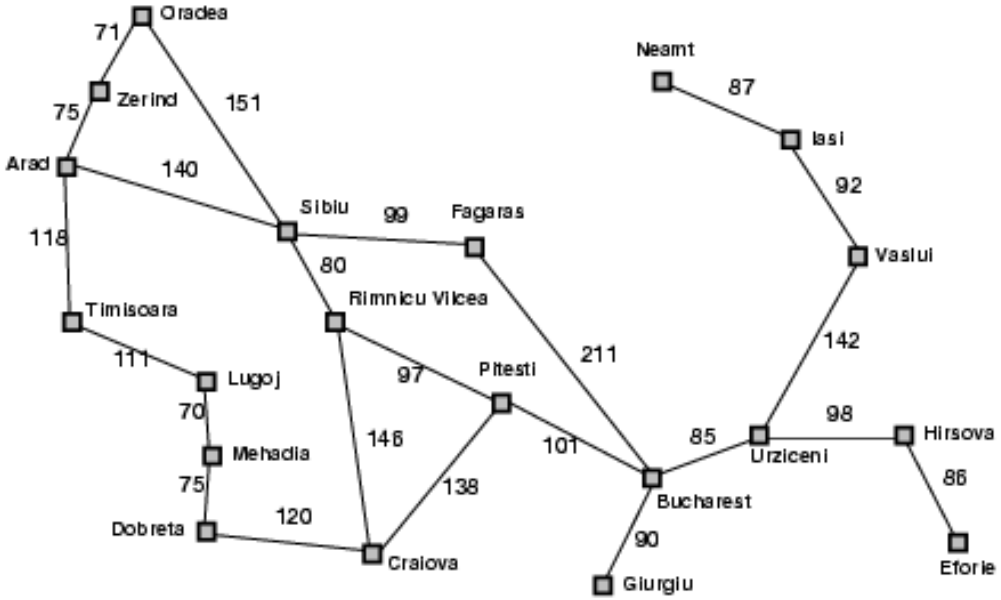


Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Primer: A* traženje

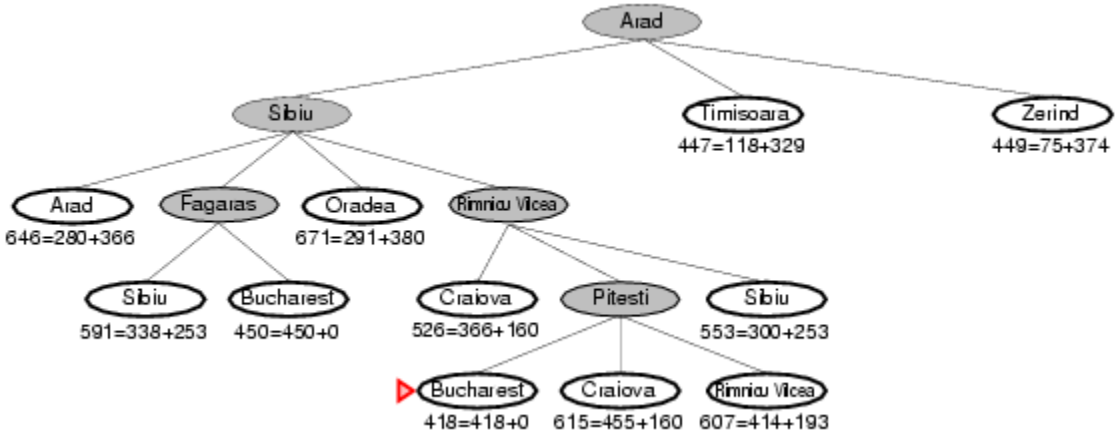


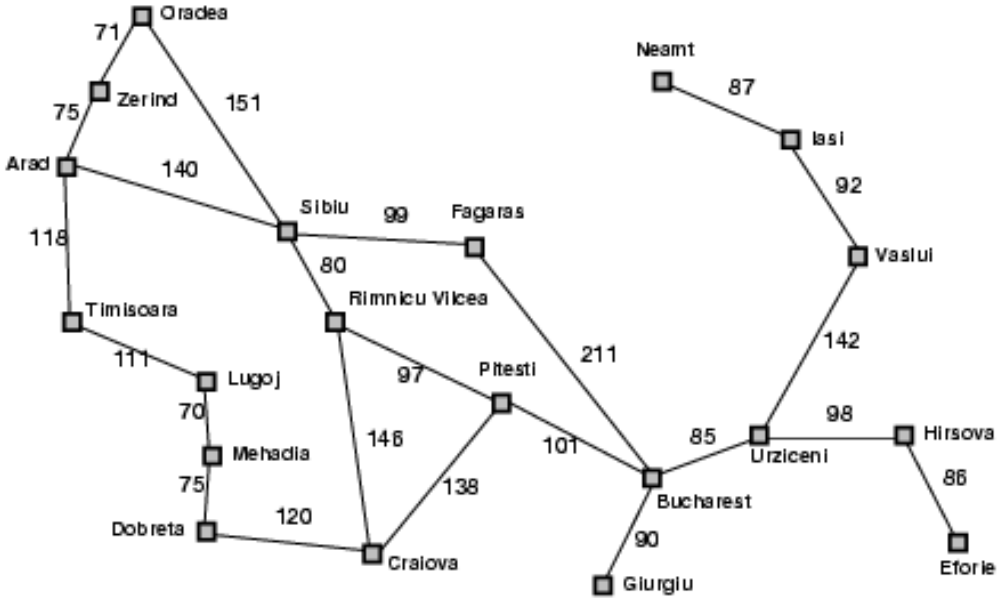


Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Primer: A* traženje





Straight–line distance to Bucharest

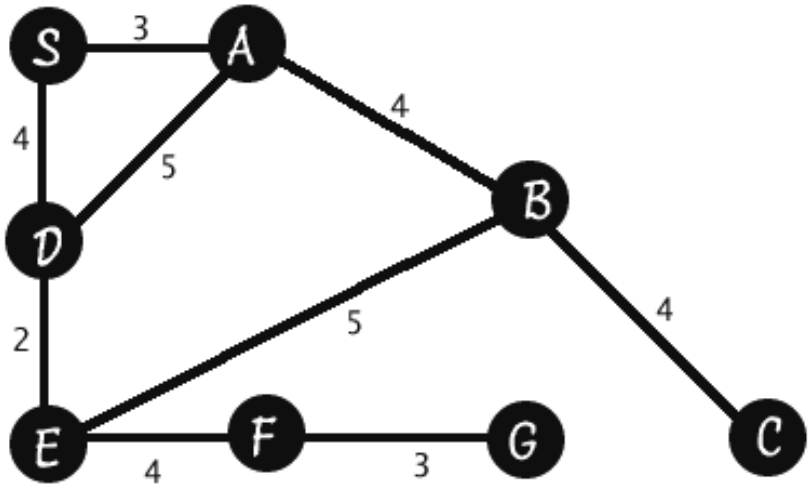
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Osobine A*

- Kompletan? Da (osim ako nema beskonačno mnogo čvorova sa f $\leq f(G)$)
- Vreme? Eksponencijalno, b^d
- Prostor? Čuva sve čvorove u memoriji
- Optimalan? Da

Optimalni algoritmi traženja
Algoritam grananja i ograničavanja

- **Branch and Bound**
- Implementacija: sortiranje pomoćne strukture, stvarna cena putanje g(n)



- **S** (rastojanje = 0) => rastojanje se odnosi na dužinu puta napr do čvora, a ne na procenjeno rastojanje tog čvora do cilja (kao kod Hill Climbing)
- Dodati: S->A (rastojanje = 3), S->D (rastojanje = 4) ,
  Sortiranje: **S->A** (rastojanje = 3), S->D (rastojanje = 4)
- Dodati: S->A->B(3 + 4 (rastojanje od A do B) = 7), S->A->D(8),
  Sortiranje: **S->D(4)**, S->A->B(7), S->A->D(8)

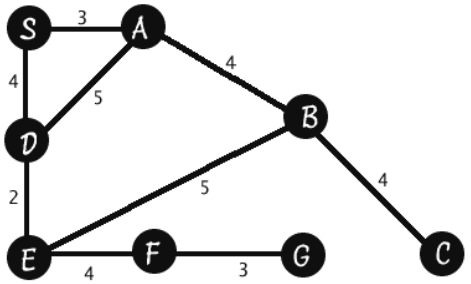
Rad algoritma B&B (nastavak)

- S->D->A(9), S->D->E(6), S->A->B(7), S->A->D(8),

  Sortiranje: **S->D->E(6)**, S->A->B(7), S->A->D(8), S->D->A(9)
- S->D->E->B(11), S->D->E->F(10), S->A->B(7), S->A->D(8), S->D->A(9),

  Sortiranje: **S->A->B(7)**, S->A->D(8), S->D->A(9), S->D->E->F(10), S->D->E->B(11)
- S->A->B->C(11), S->A->B->E(12), S->A->D(8), S->D->A(9), S->D->E->F(10), S->D->E->B(11),

  Sortiranje: **S->A->D(8)**, S->D->A(9), S->D->E->F(10), S->D->E->B(11), S->A->B->C(11), S->A->B->E(12),



Nadalje: [Najkraći put] => [lista of puteva])

- S->A->D(8) => S->A->D->E(10)
- S->D->A(9) => S->D->A->B(13)
- S->D->E->F(10) => **S->D->E->F->G(13)** cilj je dostignut sa cenom 13!

Razvoj preostalih puteva, i traženje da li postoji neki sa cenom manjom od 13.

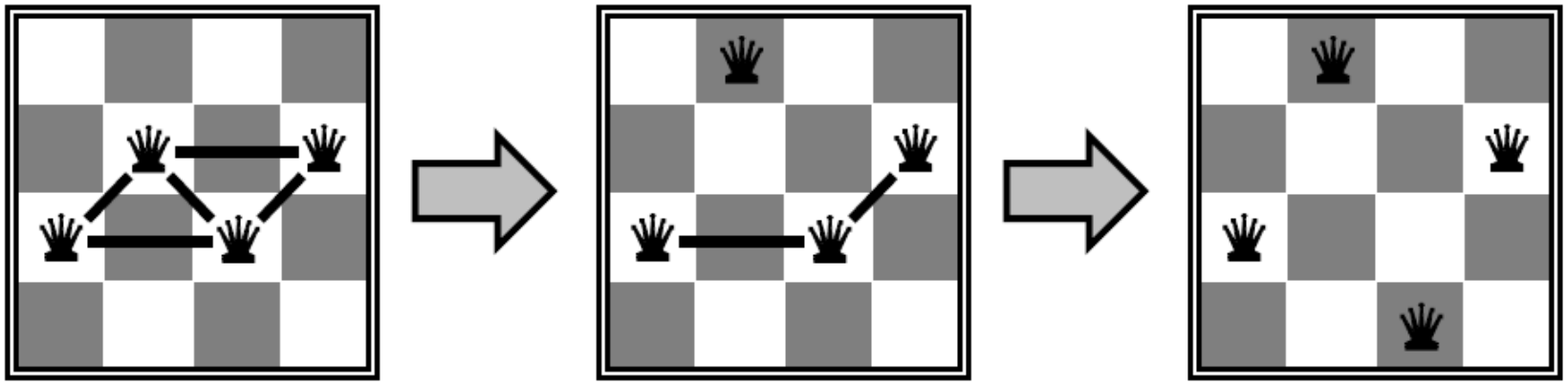
- S->A->D->E(10) ima manju cenu u narednom koraku, razvijanjem dobijamo:
- S->A->D->E->B(15), S->A->D->E->F(14), veći su od 13, ignorišemo ih, **itd**.

Local search algoritmi

- Kod mnogih optimizacionih problema, put do cilja je irelevantan;
- Kod takvih problema samo **ciljno stanje je rešenje**
- Prostor stanja = skup „kompletnih" konfiguracija (rešenja)
- Pronađi konfiguraciju koja zadovoljava ograničenja
- U takvim slučajevima, koristimo *local search* algoritme
- Čuva se jedno, „tekuće" stanje, pokušava se da se ono poboljša

Primer problema za Local Search: *n*-queens

- Postavi *n* kraljica na tabli *nxn* tako da se ne napadaju (nema dve kraljice u istoj vrsti, koloni ili dijagonali)



- Pomeri kraljicu da smanjiš broj konflikta

Metod Hill-climbing (Planinarenje)

"Like climbing Everest in thick fog with amnesia„

```
function HILL-CLIMBING( problem) returns a state that is a local maximum
   inputs: problem, a problem
   local variables: current, a node
                    neighbor, a node
   current ← MAKE-NODE(INITIAL-STATE[problem])
   loop do
      neighbor ← a highest-valued successor of current
      if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
      current ← neighbor
```

- uvek ide prema cilju
- koristi heuristiku za nalaženje pravca koji vodi najbrže (najbliže) cilju
- analogija sa planinarenjem, noću, ako je logor na vrhu brda.

Hill-climbing – prednosti i nedostaci

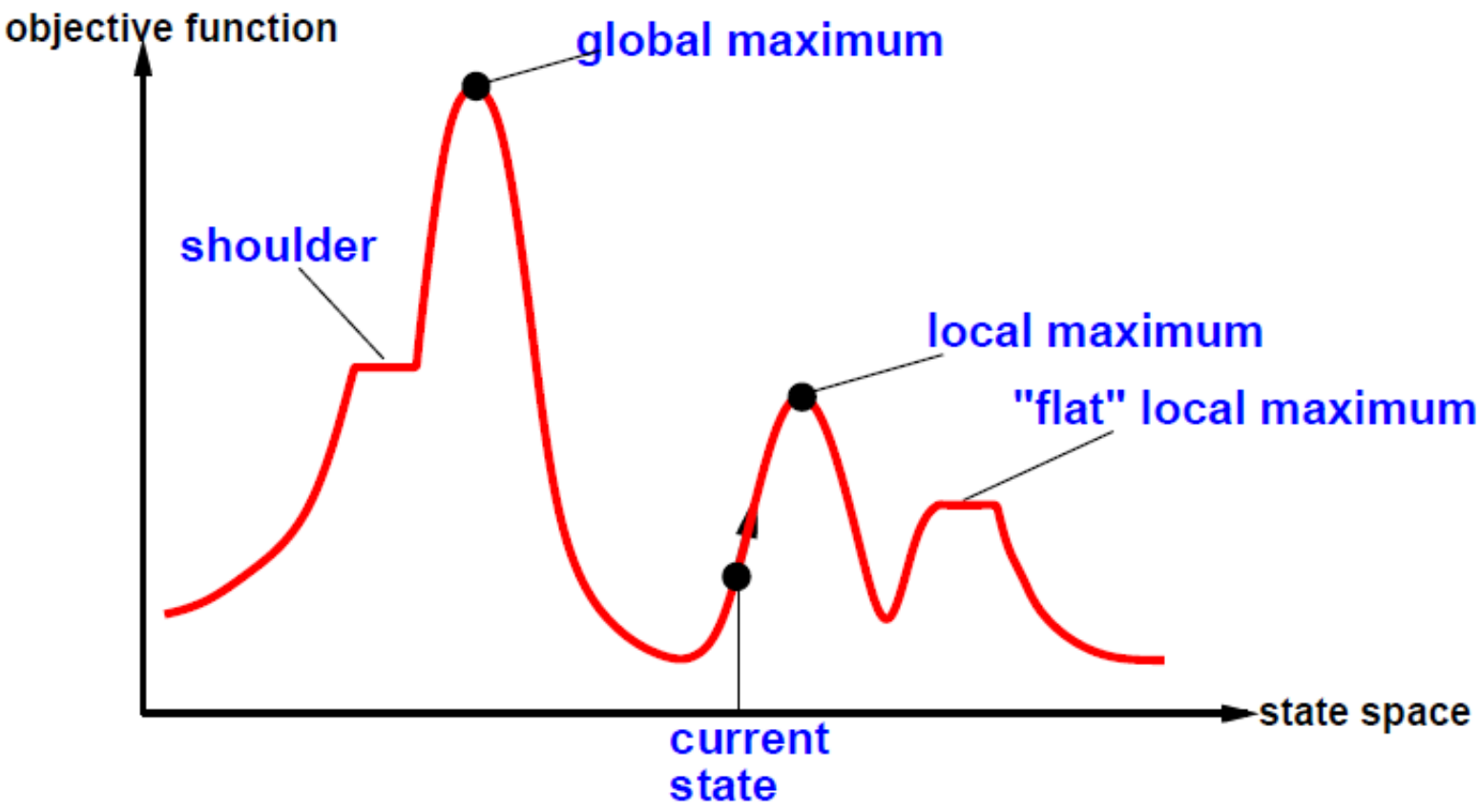
Prednosti:

- Smanjuje broj čvorova koje treba obići
- Daje efikasnije ili isto rešenje kao DFS

Nedostaci:

- "Lažni vrhovi " u kojima dolazi do izrazitih povrataka
- Problem ako svi čvorovi izgledaju kao podjednako dobri
- Neophodna heuristika
- Može se desiti da se ne nađe rešenje iako ono postoji

“Predeo” traženja



Implementacija Hill-climbing

Napr. DFS + rastojananja svakog čvora do cilja (**kao h(n)**)

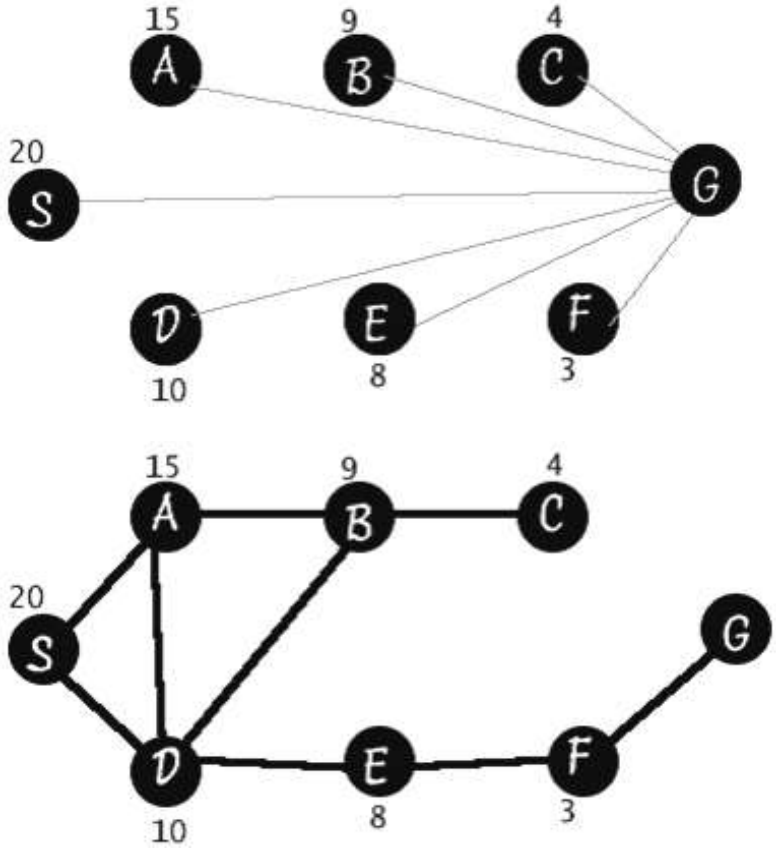
- Formirati listu čvorova koja inicijalno sadrži samo startni čvor.
- Dok se lista čvorova ne isprazni ili se ne dođe do ciljnog čvora, proveriti da li je prvi element liste ciljni čvor
  - Ako je prvi element liste ciljni čvor, ne raditi ništa.
  - Ako prvi element liste nije ciljni čvor, ukloniti ga iz liste i sortirati njegove sledbenike iz stabla pretrage (ako ih ima i ako nisu već posećeni) po rastućim vrednostima heurističke funkcije čvora (rastojanje od ciljnog čvora).
    Zatim te sledbenike dodati na početak liste tako da prvi element liste bude sledbenik sa najmanjom vrednošću heurističke funkcije.
- Ako je pronađen ciljni čvor, pretraga je uspešno završena; u suprotnom pretraga je neuspešna.

Rad algoritma Hill-climbing

Stanje magacina

- S (rastojanje 20)
- Sledbenici: S->A(15), S->D(10) => sortiranje I dodavanje u listu: *S->D(10)*, *S->A(15)*
- Sledbenici za D: S->D->A(15), S->D->B(9), S->D->E(8)

  => sortiranje I dod.: S->D->E(8), S->D->B(9), S->D->A(15), S->A(15)
- *S->D->E->F(3)*, *S->D->B(9)*, *S->D->A(15)*, *S->A(15)*
- ***S->D->E->F->G(0)***, *S->D->B(9)*, *S->D->A(15)*, *S->A(15)*



Hill-climbing varijante

- Stochastic hill-climbing
  - Slučajna selekcija nekog od „uphill“ stanja.
  - Verovatnoća selekcije može da varira od strmosti „uphill“ stanja.
- First-choice hill-climbing
  - Stohastic hill climbing sa generisanjem sledbenika slučajno sve dok se ne nađe bolji
  - Korisno kada ima veliki broj sledbenika
- Random-restart hill-climbing
  - Pokušava da izbegne zaglavljivanje u lokalnom maksimumu.
  - Više varijanti restartovanja

Simulirano kaljenje
Simulated Annealing

- Kaljenje je termički proces obrade (metala) u dva koraka:
  - Povećaj temperaturu na maksimum (na kojoj se metal topi).
  - Smanjuj pažljivo temperaturu sve dok se ne postigne minimum energije (čestice se međusobno organizuju).
- Algoritam na osnovu ovog procesa, simulira se Metropolis algoritmom, koji se zasniva na Monte Carlo tehnikama.
- Analogija:
  - Rešenje problema je ekivalentno sanju fizičkog sistema.
  - Cena rešenja je ekvivalentna “energiji” stanja.

Simulated annealing traženje

Simulated Annealing = hill-climbing with non-deterministic search

- Ideja: izbeći lokalni minimum tako što se dozvoli izbor „loših" stanja ali sa postepenim smanjivanjem njihove učestanosti
-

```
function SIMULATED-ANNEALING( problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"
  local variables: current, a node
                   next, a node
                   T, a "temperature" controlling prob. of downward steps

  current ← MAKE-NODE(INITIAL-STATE[problem])
  for t ← 1 to ∞ do
      T ← schedule[t]
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← VALUE[next] – VALUE[current]
      if ΔE > 0 then current ← next
      else current ← next only with probability e^(Δ E/T)
```

Slično Hill climbing, ali sa **slučajnim** izborom sledećeg stanja, umesto izbora najboljeg stanja

poboljšanje, izaberi stanje

Inače, izaberite stanje s verovatnoćom da opada eksponencijalno sa „loše" kod promene stanja.

Koja je verovatnoća kada:

T → inf?
T → 0?
Δ=0?
Δ→-∞?

Simulated Annealing - detalji

- Neka su 3 stanja moguća, sa promenama *objective function* d1 = -0.1, d2 = 0.5, d3 = -5. (neka je T = 1).
- Izaberi stanje slučajno:
  - Ako je to d2, izaberi ga.
  - Ako je d1 ili d3, računaj verovatnoću = exp(d/T)
    - stanje 1: prob1 = exp(-0.1) = 0.9, odnosno, 90%
    - stanje 3: prob3 = exp(-5) = 0.05, tj., 5%
- Parametar T = "temperatura"
  - high T => verovatnoća za "locally bad" stanje je veća
  - low T => verovatnoća za "locally bad" stanje je manja
  - T se smanjuje u svakom koraku algoritma – postoji „raspored" promene

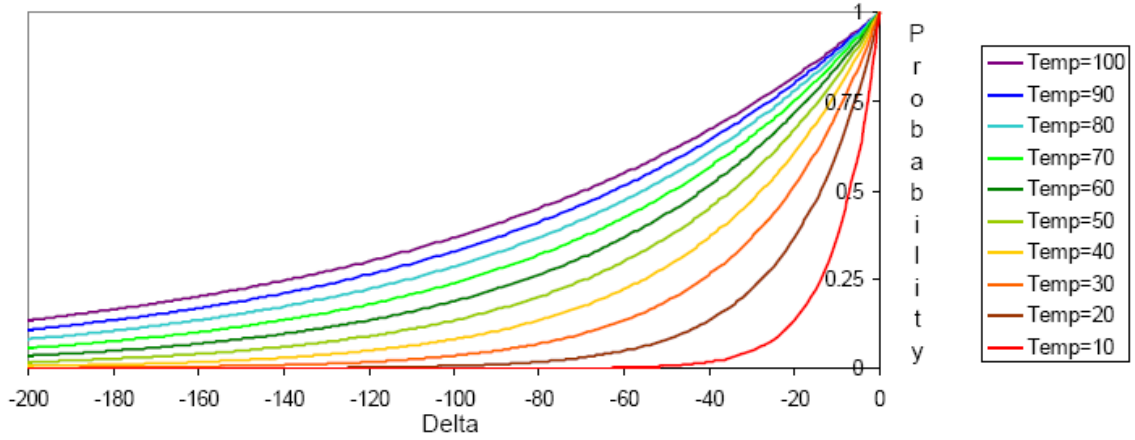
Osobine *Simulated annealing* traženja

- Može se dokazati: ako se T smanjuje dovoljno sporo, tada će traženje simuliranim kaljenjem pronaći globalni optimum sa verovatnoćom koja se približava 1

Kriterijum prihvatanja i raspored hlađenja (cooling schedule)

if (delta>=0) accept

else if ($random < e^{delta/Temp}$) accept, else reject /* 0<=random<=1 */



Initially temperature is very high (most bad moves accepted)

Temp slowly goes to 0, with multiple moves attempted at each temperature

Final runs with temp=0 (always reject bad moves) greedily “quench” the system

Simulated annealing: Praktična realizacija

- Početak na temperaturi gde se 50% loših stanja prihvata.
- Svaki korak hlađenja smanjuje temperaturu za 10%
- Broj iteracija na svakoj temperaturi treba da pokuša 1-10 puta svaki “element” stanja.
- Finalna temperatura ne treba da prihvata loša stanja (korak gašenja).

Simulated annealing: Primena

Široka primena algoritma: VLSI layout, airline scheduling, itd

- Osnovni problemi:
  - Traveling salesman
  - Graph partitioning
  - Matching problems
  - Graph coloring
  - Scheduling
- Inženjering:
  - VLSI design
    - Placement
    - Routing
    - Array logic minimization
    - Layout
  - Facilities layout
  - Image processing
  - Code design in information theory

PITANJA?

Dileme?

Komentari?