I007 Osnove umjetne inteligencije

Tema: Pretraživanje prostora stanja.

17. 3. 2021.

predavač: Darija Marković

asistent: Darija Marković

Rekonstrukcija rješenja

- u strukturi čvora moramo pamtiti i pokazivač na roditeljski čvor:

$$n = (s, d, p), \text{parent}(n) = p$$

function $\text{expand}(n, \text{succ})$
  return $\{ (s, \text{depth}(n) + 1, n) \mid s \in \text{succ}(\text{state}(n)) \}$

- krećemo od ciljnog čvora i pratimo pokazivače unazad:

Rekonstrukcija puta do čvora

function $\text{path}(n)$
  $p \leftarrow \text{parent}(n)$
  if $p = null$ then return [$\text{state}(n)$]
  return $\text{insertBack}(\text{state}(n), \text{path}(p))$

- vremenska složenost je $\mathcal{O}(d)$

Problem ponavljanja stanja

- Rješenje 1: spriječiti povratak u stanje iz kojeg smo došli

Opći algoritam pretraživanja(nadopuna)

function $\mathrm{search}(s_0, \mathrm{succ}, \mathrm{goal})$
  $open \leftarrow$ [$\mathrm{initial}(s_0)$]
  while $open \neq$ [ ] do
    $n \leftarrow \mathrm{removeHead}(open)$
    if $\mathrm{goal}(\mathrm{state}(n))$ then return n
    for $m \in \mathrm{expand}(n)$ do
      if $\mathrm{state}(m) \neq \mathrm{state}\ (\mathrm{parent}(n))$ then $\mathrm{insert}(m, open)$
  return $fail$

Problem ponavljanja stanja

- Rješenje 2: spriječiti nastanak puteva s ciklusima

Opći algoritam pretraživanja(nadopuna)

function $\mathrm{search}(s_0, \mathrm{succ}, \mathrm{goal})$
  $open \leftarrow [\ \mathrm{initial}(s_0)\ ]$
  while $open \neq [\ ]$ do
    $n \leftarrow \mathrm{removeHead}(open)$
    if $\mathrm{goal}(\mathrm{state}(n))$ then return n
    for $m \in \mathrm{expand}(n)$ do
      if $\mathrm{state}(m) \notin \mathrm{path}\ ((n))$ then $\mathrm{insert}(m, open)$
  return $fail$

- sprječava cikluse (osigurava potpunost algoritma)
- ne sprječava ponavljanje na različitim putevima u stablu pretraživanja
- povećava vremensku složenost za faktor $\mathcal{O}(d)$

Problem ponavljanja stanja

- Rješenje 3: spriječiti ponavljanje bilo kojeg stanja

Opći algoritam pretraživanja s listom posjećenih stanja

function $\mathrm{search}(s_0, \mathrm{succ}, \mathrm{goal})$
  $open \leftarrow [\ \mathrm{initial}(s_0)\ ]$
  $visited \leftarrow \emptyset$
  while $open \neq [\ ]$ do
    $n \leftarrow \mathrm{removeHead}(open)$
    if $\mathrm{goal}(\mathrm{state}(n))$ then return n
    $visited \leftarrow visited \cup \{\mathrm{state}(n)\}$
    for $m \in \mathrm{expand}(n)$ do
      if $\mathrm{state}(m) \notin visited$ then $\mathrm{insert}(m, open)$
  return $fail$

- lista (skup) posjećenih stanja pohranjuje stanja, a ne čvorove

Problem ponavljanja stanja - komentari

- korištenje liste posjećenih čvorova osigurava potpunost algoritma (sprječava da zaglavi u beskonačnoj petlji)
- osim toga, može smanjiti prostornu i vremensku složenost: budući da se stanja ne ponavljaju, umjesto složenosti $\mathcal{O}(b^{d+1})$ imamo $\mathcal{O}(\min(b^{d+1}, b|S|))$, gdje je $|S|$ veličina prostora stanja (u praksi je često $b|S| < b^d$)
- lista posjećenih stanja uobičajeno se implementira tablicom raspršenog adresiranja (engl. hash table) (omogućava provjeru "$\text{state}(m) \notin visited$" u vremenu $\mathcal{O}(1)$)

Usmjereno pretraživanje - motivacija

- slijepi postupci raspolažu isključivo egzaktnim informacijama (početnim stanjem, operatorima i ispitnim predikatom)
- ne koriste nikakvu dodatnu informaciju o prirodi problema koja bi mogla poboljšati učinkovitost pretraživanja
- ako otprilike znamo u kojem se smjeru nalazi rješenje, zašto ne iskoristiti to znanje kako bismo ubrzali pretragu?

Heuristika

- heuristika - iskustvena pravila o prirodi problema i osobinama cilja čija je svrha pretraživanje brže usmjeriti k cilju

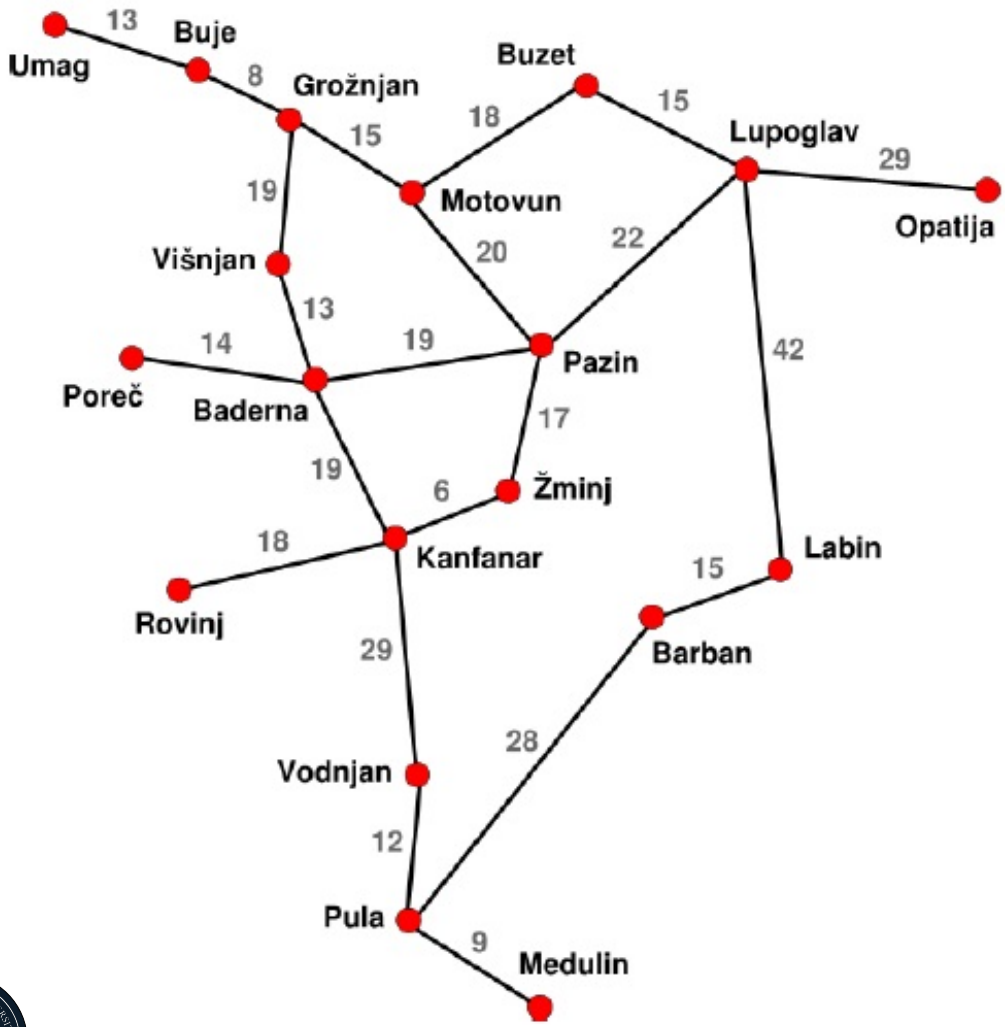
Heuristička funkcija

Heuristička funkcija $h : S \to \mathbb{R}^+$ pridjeljuje svakom stanju $s \in S$ procjenu udaljenosti od tog stanja do ciljnog stanja.

Što je vrijednost $h(s)$ manja, to je čvor s bliži ciljnome stanju. Ako je s ciljno stanje, onda $h(s) = 0$

- postupke pretraživanja koji koriste heuristiku kako bi suzili prostor pretraživanja nazivamo usmjerenim ili heurističkim

Primjer: Putovanje kroz Istru



Zračne udaljenosti do Buzeta:

Baderna	25
Barban	35
Buje	21
Grožnjan	17
Kanfanar	30
Labin	35
Lupoglav	13
Medulin	61
Motovun	12
Opatija	26
Pazin	17
Poreč	32
Pula	57
Rovinj	40
Umag	31
Višnjan	20
Vodnjan	47
Žminj	27

Primjer: Slagalica 3×3

početno stanje:

8		7
6	5	4
3	2	1

ciljno stanje:

1	2	3
4	5	6
7	8	

Heuristička funkcija?

- broj pločica koje nisu na svome mjestu

$$h_1\left(\begin{array}{|c|c|c|}\hline 8 & & 7 \\ \hline 6 & 5 & 4 \\ \hline 3 & 2 & 1 \\ \hline\end{array}\right) = 7$$

- zbroj Manhattan udaljenosti(L1) pločica od svoga mjesta

$$h_2\left(\begin{array}{|c|c|c|}\hline 8 & & 7 \\ \hline 6 & 5 & 4 \\ \hline 3 & 2 & 1 \\ \hline\end{array}\right) = 21$$

Usmjereno pretraživanje

Razmotrit ćemo:

1. pretraživanje “najbolji prvi” (engl. greedy best-first search)
2. algoritam A^*
3. pretraživanje usponom na vrh (engl. hill-climbing search)

Pretraživanje “najbolji prvi”

- proširuje čvor s najboljom heurističkom vrijednošću
- ovo je tzv. pohlepan (engl. greedy) algoritam “najbolji prvi”
- pohlepno pretraživanje: odabire se onaj čvor koji se čini najbliži cilju, ne uzimajući u obzir ukupnu cijenu puta
- odabrani put možda nije optimalan, no algoritam nema mogućnost oporavka od pogreške! Dakle, algoritam nije optimalan
- nije potpun (osim ako koristimo listu posjećenih stanja)
- vremenska i prostorna složenost: $\mathcal{O}(b^m)$

Algoritam A^*

- algoritam “najbolji prvi” koji u obzir uzima i heuristiku i cijenu ostvarenog puta, tj. kombinira “najbolji prvi” i pretraživanje s jednolikom cijenom
- kao i kod pretraživanja s jednolikom cijenom, pri proširenju čvora ažurira se cijena do tada ostvarenog puta:

function $\mathrm{expand}(n, \mathrm{succ})$
  return $\{ (s, g(n) + c) \mid (s, c) \in \mathrm{succ}(\mathrm{state}(n)) \}$

Ukupna cijena računa se na temelju:

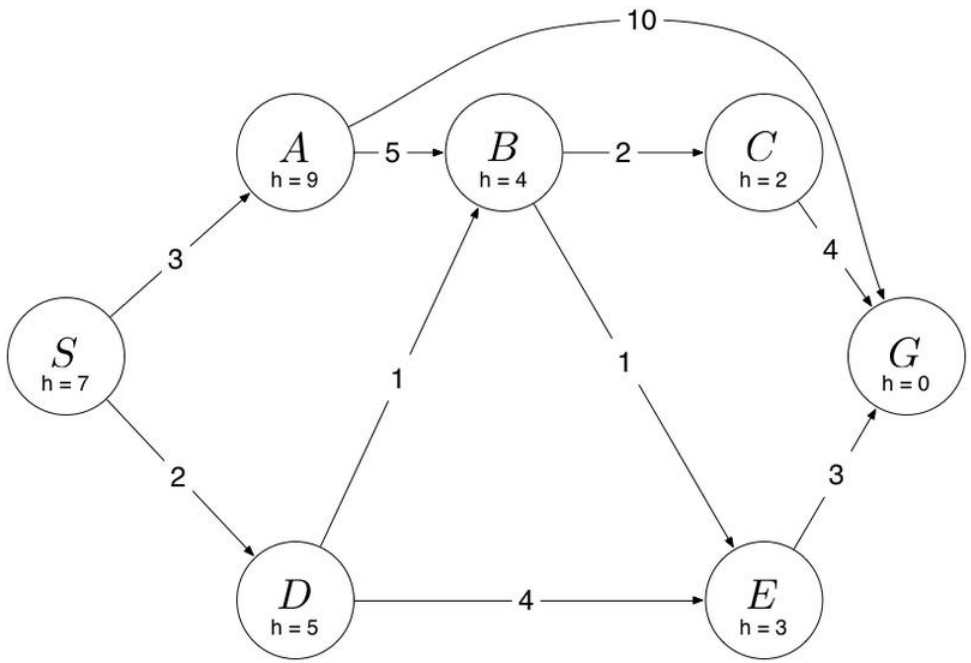
$g(n)$ - stvarna cijena puta od početnog čvora do čvora n

$h(s)$ - procjena cijene puta od stanja s do cilja

$$f(n) = g(n) + h(\mathrm{state}(n))$$

Primjer 2.

Koristeći algoritam A^* razmotrit ćemo redosljed posjećivanja čvrova, prateći frontu (listu otvorenih čvorova). U pozicijama gdje imamo nekoliko izbora za proširivanje, dat ćemo prednost izboru s manjom abecednom vrijednošću, npr. $S \to X \to A$ proširujemo prije $S \to X \to B$ i slično $S \to A \to Z$ proširujemo prije $S \to B \to A$.



Algoritam A^* - svojstva

- vremenska i prostorna složenost: $\mathcal{O}(\min(b^{d+1}, b|S|))$ (u praksi veći problem predstavlja prostorna složenost)
- potpunost: da, jer u obzir uzima cijenu puta
- optimalnost: da, ali pod uvjetom da je heuristika h optimistična:

Optimističnost heuristike

Heuristika h je optimistična ili dopustiva (engl. optimistic, admissible) akko nikad ne precjenuje, tj. nikad nije veća od prave cijene do cilja:

$$\forall s \in S \quad h(s) \leq h^*(s),$$

gdje je $h^*(s)$ prava cijena od stanja s do cilja.

- ako heuristika nije optimistična, može se dogoditi da pretraga zaobiđe optimalni put jer se on čini skupljim nego što zapravo jest

Konzistentna heuristika

- uz pretpostavku optimistične heuristike, vrijedi $f(n) \leq C^*$ (funkcija cijene je odozgo ograničena)
- duž staze u stablu pretraživanja, $f(n)$ može općenito rasti i padati, a u ciljnom stanju vrijedi $f(n) = g(n) = C^*$
- poželjno je da $f(n)$ monotono raste:

$$\forall n_2 \in \text{expand}(n_1) \Longrightarrow f(n_2) \geq f(n_1)$$

- ako $f(n)$ monotono raste, svaki čvor koji prvi generiramo za neko stanje bit će čvor s najmanjom cijenom za to stanje
- to znači da se niti za jedno stanje ne može dogoditi da na listi zatvorenih čvorova nađemo jeftiniji čvor s istim stanjem, pa ne moramo provjeravati cijenu već zatvorenih čvorova

Konzistentna heuristika

Konzistentnost heuristike

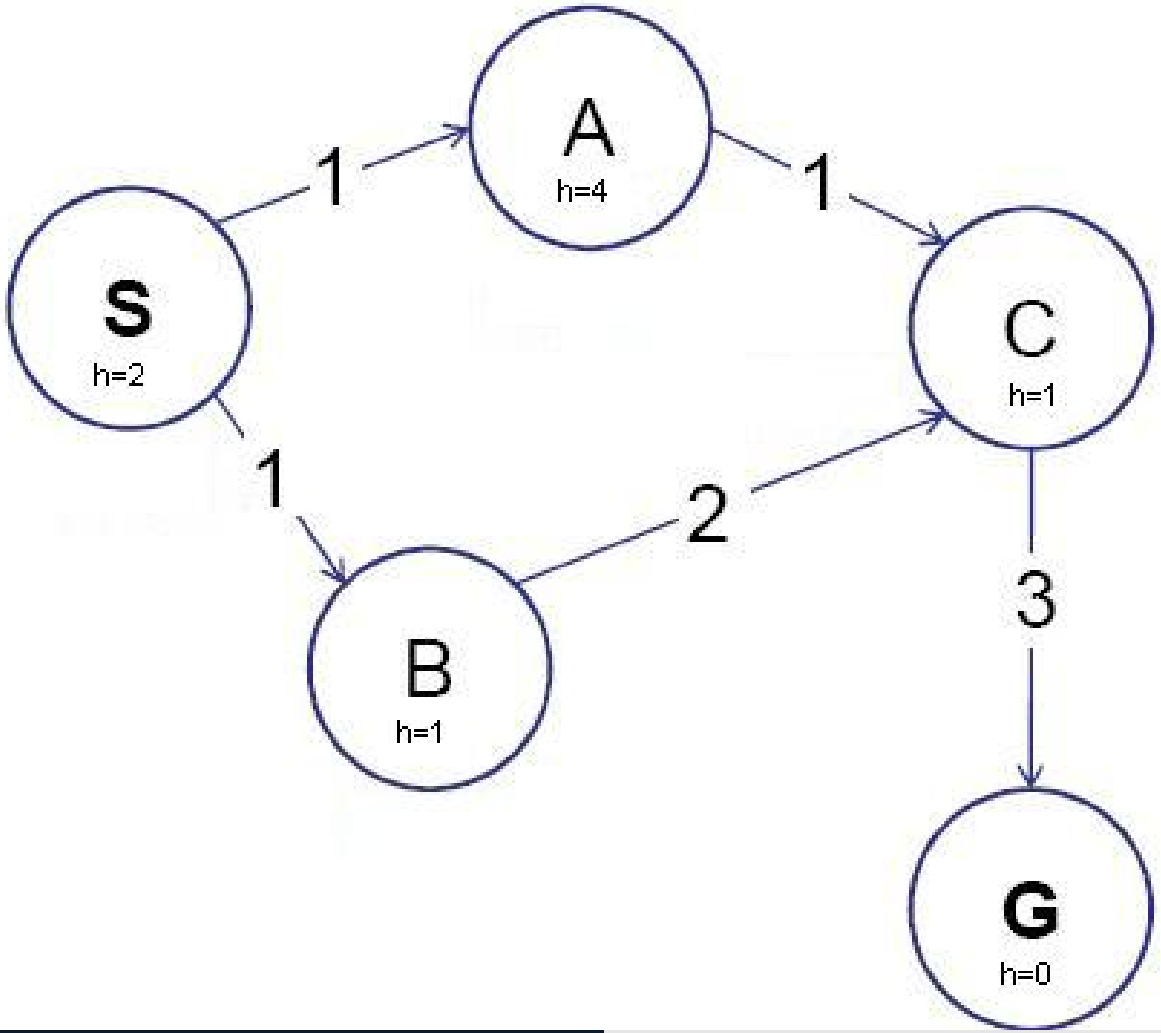
Heuristika h je konzistentna ili monotona akko:

$$\forall (s_2, c) \in \mathrm{succ}(s_1) \quad h(s_1) \leq h(s_2) + c$$

- konzistentna heuristika je nužno optimistična, a u praksi su optimistične heuristike ujedno i konzistentne

Primjer 3.

Je li zadana heuristika optimistična? Je li konzistentna?



Svojstvo dominacije

Dominacija

Neka su A_1^* i A_2^* dva optimalna algoritma s optimističnim heurističkim funkcijama h_1 i h_2. Algoritam A_1^* dominira nad algoritmom A_2^* akko:

$$\forall s \in S \quad h_1(s) \geq h_2(s)$$

Također kažemo da je algoritam A_1^* obavještеniji od algoritma A_2^*.

- obavještеniji algoritam općenito pretražuje manji prostor stanja od manje obaviještenog algoritma
- npr. za slagalicu vrijedi: $h^*(s) \geq h_2(s) \geq h_1(s)$, tj. L1-heuristika daje obavještеniji algoritam od heuristike koja broji razmještene pločice
- pritom u obzir treba uzeti i složenost izračunavanja heuristike!

Dobra heuristika

- Dobra heuristika je:
  1. optimistična
  2. što obavještenija
  3. jednostavno izračunljiva

Pesimistične heuristike?

- ako ne trebamo baš optimalno rješenje, nego neko rješenje koje je dovoljno dobro, možemo koristiti heuristiku koja nije optimistična (heuristiku koja precjenjuje)
- uporaba takve heuristike dodatno će smanjiti broj generiranih čvorova
- radimo kompromis izmedu kvalitete rješenja i složenosti pretraživanja

- kako oblikovati dobru heuristiku za neki zadani problem?

Oblikovanje heuristike

1. Relaksakcija problema
   - stvarna cijena relaksiranog problema je optimistična heuristika izvornog problema
   - npr. relaksacija slagalice 3×3: pločice se mogu pomicati bilo kuda $\Rightarrow$ L1-udaljenost
   - dobivamo optimističnu i ujedno konzistentnu heuristiku
2. Kombiniranje optimističnih heuristika
   - Ako su $h_1, h_2, \ldots, h_n$ optimistične, možemo ih kombinirati u dominantnu heuristiku koja će također biti optimistična:

$$h(s) = \max(h_1(s), h_2(s), \ldots, h_n(s))$$

3. Cijena rješavanja podproblema
   - baza uzoraka koja sadržava optimalne cijene pojedinih podproblema
4. Učenje heuristike
   - primjena metoda strojnog učenja. Npr. učimo koeficijente w_1 i w_2: $h(s) = w_1 x_1(s) + w_2 x_2(s)$, gdje su x_1 i x_2 značajke stanja

Algoritam uspona na vrh

- kao pohlepno pretraživanje “najbolji prvi”, s tom razlikom da se prošireni čvorovi uopće ne pohranjuju u memoriji
- nije potpun i nije optimalan
- lako zaglavljuje u tzv. lokalnim optimumima
- učinkovitost uvelike ovisi o izboru heurističke funkcije
- uobičajeno se koristi tehnika slučajnog ponovnog starta (engl. random-restart)
- vremenska složenost: $\mathcal{O}(m)$
- prostorna složenost: $\mathcal{O}(1)$