SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Usporedba heurističkih algoritama za rješavanje optimizacijskih problema

Tomislav Novak
Voditelj: doc.dr.sc. Domagoj Jakobović

Zagreb, 16. svibnja 2008.

Sadržaj

1. Uvod

Postoji velik broj problema u domeni računarske znanosti za koje još uvijek nije pronađeno efikasno rješenje. Jedan od najpoznatijih je i problem trgovačkog putnika (engl. *traveling salesman problem*). Prilično ga je jednostavno opisati: trgovački putnik želi pronaći najkraću rutu kojom moze obići n gradova te se vratiti u grad iz kojeg je krenuo, tako da svaki grad posjeti točno jedamput. Za problem trgovačkog putnika ne postoji algoritam polinomijalne složenosti koji bi ga rješavao. Naime, neka n-torka $(v_1, v_2, \ldots, v_n)$ predstavlja gradove redom kojim ih se obilazi. Lako se zaključuje kako je broj različitih ruta jednak broju permutacija skupa od n elemenata. Dakle, algoritam koji ispituje sve moguće rute te pronalazi najkraću faktorijelne je složenosti - $O(n!)$. Usprkos brzom razvoju modernih računala, takvom algoritmu trebalo bi više milijardi godina da pronađe rješenje za nešto veću instancu problema.

Budući da probleme kao što je ovaj nije moguće deterministički riješiti u prihvatljivom vremenskom, potrebno je poslužiti se *heurističkim metodama*. Takvi algoritmi ne daju nužno optimalno rješenje, no pokazalo se kako je to rješenje često blizu optimalnom te ga možemo smatrati prihvatljivim (dopustivim). Heuristički algoritmi najčešće kreću od nekog slučajno odabranog rješenja, te ga u svakoj iteraciji modificiraju kako bi se što više približilo optimalnom.

Heuristički algoritmi baziraju se na različitim strategijama. U ovom se radu opisuju četiri različite metode te njihova primjena na problem naprtnjače (engl. *knapsack problem*) i prije spomenuti problem trgovačkog putnika. Oba navedena problema spadaju u klasu NP potpunih (prema engl. *non-deterministic polynomial time*) problema. Tu klasu čine problemi čije se rješenje ne može pronaći u polinomijalnom vremenu, ali se može ispitati je li neko rješenje ispravno. Više o toj klasi problema može se pročitati u [1].

2. Heuristički algoritmi

Prilikom osmišljavanja algoritma za rješavanje nekog optimizacijskog problema (pronalaženje *najboljeg* iz skupa mogućih rješenja) važno je voditi računa o dokazivosti da je vrijeme izvršavanja ograničeno odozgo, te dokazivanju da algoritam doista pronalazi optimalno rješenje. Heuristički algoritam je onaj koji ne zadovoljava barem jedan od ta dva kriterija. Najčešće takav algoritam daje prilično dobra rješenja u prihvatljivom vremenu, no nije dokazano da će to biti slučaj za bilo koju instancu problema.

Osim heurističkih, postoje i aproksimacijski algoritmi koji se koriste u iste svrhe. Razlika imeđu njih je što se za aproksimacijske algoritme može matematički odrediti koliko će se dobro približiti optimalnom rješenju, dok za heuristički algoritam nema dokaza da uvijek radi dobro, ali se u praksi pokazuje zadovoljavajućim.

U nastavku su prikazane četiri heurističke metode: penjanje uzbrdo (engl. *hill climbing*), simulirano kaljenje (engl. *simulated annealing*), tabu pretraživanje (engl. *tabu search*) te genetski algoritam.

Prije opisivanja samih heuristika, potrebno je iznijeti nekoliko formalnih definicija:

- **optimizacijski problem** je uređena četvorka (I, f, m, g), gdje je
  - I skup instanci problema
  - za instancu $x \in I$, $f(x)$ je skup mogućih rješenja problema (*prostor rješenja*)
  - za instancu x i rješenje $y \in f(x)$, $m(x, y)$ predstavlja **mjeru** (dobrotu) tog rješenja (neku vrijedost koja se može uspoređivati, najčešće realan broj)
  - g je funkcija cilja (min ili max)

  Optimalno rješenje y instance problema x je ono rješenje za koje vrijedi da je

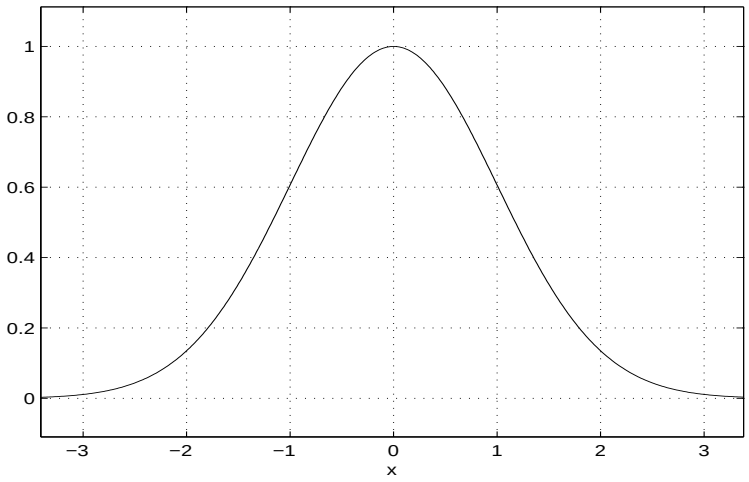
  $$m(x, y) = g\{m(x, y') \mid y' \in f(x)\}$$

- za svako rješenje y iz skupa mogućih rješenja definira se **susjedstvo rješenja** $N(y)$. Relacija susjedstva opisuje se različito za svaki problem. Ako se prostor problema prikaže usmjerenim grafom $G(V, E)$, gdje skup vrhova V predstavlja skup rješenja, tada za svako rješenje $v \in N(u)$ vrijedi da je $(u, v) \in E(G)$.

2.1. Penjanje uzbrdo

Penjanje uzbrdo jedan je od najjednostavnijih (kako idejno, tako i za implementaciju) heurističkih algoritama. Algoritam započinje sa slučajno odabranim (potencijalno lošim) rješenjem problema, te ga u svakoj iteraciji poboljšava. Kada dođe do razine gdje više nije moguće poboljšati rješenje, algoritam završava. Formalno gledano, u svakoj se iteraciji trenutno rješenje u zamijenjuje najboljim rješenjem $v \in N(u)$, ukoliko je ono bolje od rješenja u.

Promotrimo problem nalaženja ekstrema neke *diskretne*[1] funkcije. Slika 2.1 prikazuje graf Gaussove zvonolike krivulje $f(x) = e^{-\frac{1}{2}x^2}$. Skup rješenja ovog problema jednak je domeni funkcije (u ovom primjeru skup cijelih brojeva), dok je mjera rješenja upravo vrijednost funkcije u pripadnoj točki. Optimalno rješenje problema je ono čija je mjera maksimalna - točka x s najvećom vrijednosti $f(x)$.

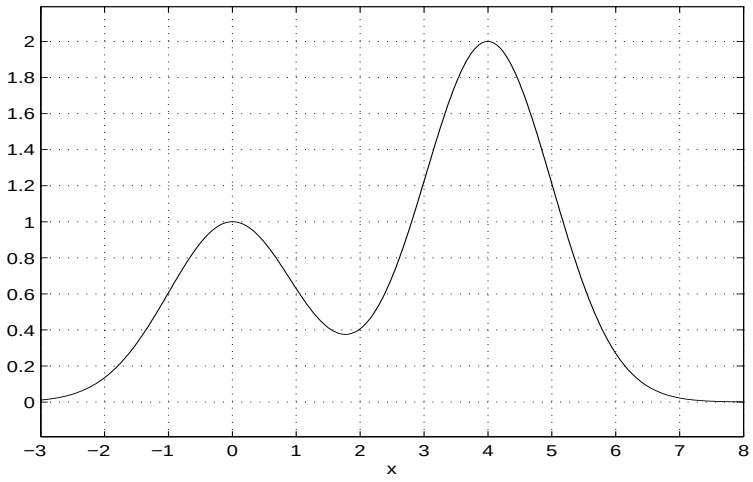


Slika 2.1. Graf funkcije $e^{-\frac{1}{2}x^2}$

Rješenju x susjedna su rješenja točke lijevo i desno od točke x, tj. $N(x) = \{x-1, x+1\}$. Dakle, penjanjem uzbrdo kretat će se, od slučajno odabrane početne točke x, uvijek prema točki za koju je vrijednost funkcije veća, sve dok se ne dođe do maksimuma funkcije (optimalno rješenje).

Promotrimo sada funkciju $f(x) = e^{-\frac{1}{2}x^2} + 2e^{-\frac{1}{2}(x-4)^2}$ (slika 2.2). Ukoliko algoritam započne u točki $x = 7$, nakon tri iteracije pronaći će maksimum funkcije (točka $x = 4$). No, ako se "penjanje" započne u točki $x = -2$, algoritam se neće više pomaknuti iz točke $x = 0$, budući da su oba susjedna rješenja lošija (manja vrijednost funkcije).

[1] Za kontinuirane funkcije prostor rješenja nije konačan.



Slika 2.2. Graf funkcije $e^{-\frac{1}{2}x^2} + 2e^{-\frac{1}{2}(x-4)^2}$

Opisani slučaj najveća je mana penjanja uzbrdo. Naime, za neku instancu problema $x \in I$, penjanjem uzbrdo pronaći ce se **lokalni ekstrem**[2] funkcije $m(x, y)$ (odnosno, pronađeno rješenje bit će **lokalno optimalno**). O dobro odabranom početnom rješenju ovisi hoće li to biti ujedno i globalni ekstrem.[3]

2.2. Simulirano kaljenje

Simulirano kaljenje je heuristiški algoritam za rješavanje optimizacijskih problema s velikim prostorom rješenja. Algoritam sadrži poboljšanje u odnosu na penjanje uzbrdo koje mu omogućuje da se istrgne iz lokalno optimalnog rješenja (osnovni nedostatak penjanja uzbrdo) te istraži drugi dijelove prostora rješenja.

Algoritam je inspiriran procesom kaljenja u metalurgiji. To je tehnika zagrijavanja i kontroliranog hlađenja materijala kako bi se postigla određena bolja svojstva. Naime, zagrijavanjem čestice dobivaju dovoljnu energiju da pobjegnu iz svog početnog položaja (lokalni minimum unutarnje energije) te se kreću po stanjima više energije. Postupnim hlađenjem omogućuje im se da pronađu konfiguracije s nižom unutarnjom energijom od početne.

Za razliku od penjanja uzbrdo, simulirano kaljenje omogućuje da se trenutno pronađeno rješenje s određenom vjerojatnošću zamijeni *lošijim susjednim rješenjem*. Ta je vjerojatnost funkcija koja obično ovisi o razlici "dobrote" trenutnog i susjednog rješenja,

[2]Minimum ili maksimum funkcije na određenom lokalnom području (intervalu).

[3]Minimum ili maksimum funkcije na cijeloj domeni.

tako da je za manje razlike vjerojatnost velika, dok je za veće razlike praktički jednaka nuli. Također, po analogiji s fizikalnim procesom, uvodi se parametar **temperature** T, takav da je vjerojatnost prelaska u lošija stanja veća što je temperatura veća. Temperatura s vremenom (iz iteracije u iteraciju izvršavanja algoritma) opada s faktorom α, tj. $T' = \alpha T$. Ovo svojstvo omogućuje algoritmu da u početku "skače" između različitih rješenja (istražuje prostor rješenja), dok je u kasnijim iteracijama sklonost prelaska u lošija rješenja sve manja, pa algoritam "titra" oko lokalno optimalnog područja.

Neka x predstavlja trenutno rješenje, dok je $x' \in N(y)$ neko susjedno lošije rješenje; $m(x)$ predstavlja mjeru (dobrotu) rješenja. Najčešće se vjerojatnost prelaska u stanje x' opisuje izrazom:

$$\exp(-\frac{m(x) - m(x')}{T})$$

Algoritam simuliranog kaljenja opisan je sljedećim pseudokodom:

SIMULIRANO-KALJENJE

```
1  x ← početno dopustivo rješenje
2  c ← 0
3  opt ← m(x)
4  sve dok vrijedi c < broj_iteracija
5      radi x' ← slučajno odabrano rješenje iz skupa N(x)
6          ako je m(x') > m(x)
7             onda x ← x'
8             inačep ← exp(−(m(x) − m(x'))/T)
9                    r ← slučajan broj iz intervala [0, 1]
10                   ako je r < p
11                      onda x ← x'
12         ako je m(x) > opt
13            onda opt ← m(x)
14         T ← αT
15         c ← c + 1
```

2.3. Tabu pretraživanje

Tabu pretraživanje je heuristički algoritam koji u svakom koraku izvršavanja zamijeni trenutno rješenje x najboljim rješenjem x' iz njegove okoline (skupa $N(x)$).[4] Međutim, kako bi se izbjegli ciklusi (primjerice, u jednoj se iteraciji zamijeni x s x', a već u sljedećoj x' s x) te se osiguralo da algoritam ne zaglavi u lokalnom optimumu, izgrađuje se

[4] Za razliku od penjanja uzbrdo, algoritam neće završiti s radom ukoliko u susjedstvu od x ne postoji bolje rješenje.

tzv. *tabu-lista* koja sadrži neka rješenja koja algoritam **ne smije odabrati** u sljedećem koraku. U tabu-listi se obično nalaze rješenja koja je algoritam posjetio u zadnjih L iteracija.

Tabu-lista ne mora sadržavati rješenja, već je mogu činiti samo neki atributi pojedinih rješenja. Međutim, tada se javlja sljedeći problem: jedan atribut u tabu listi može uzrokovati da više od jednog rješenja bude "tabu", tj. da se ne smije odabrati u sljedećem koraku napredovanja algoritma. Kako bi se doskočilo tom problemu, uvodi se sljedeći kriterij: ukoliko je susjedno rješenje *bolje* od trenutno poznatog najboljeg rješenja, ono se dopušta bez obzira na to nalazi li se u tabu-listi.

2.4. Genetski algoritam

Genetski algoritam je heuristički algoritam koji spada u klasu tzv. *evolucijskih algoritama*, algoritama koji oponašaju mehanizme prirodne selekcije kao što su reprodukcija, mutacija, rekombinacija i selekcija.

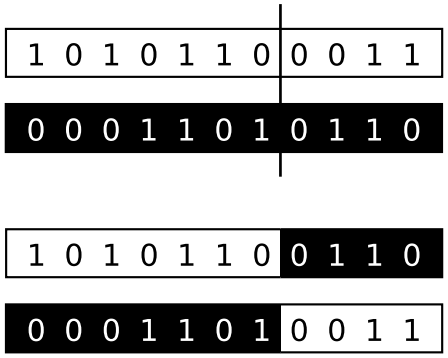
Genetski algoritam možemo nazvati računalnom simulacijom prirodnog odabira, u kojoj *populacija* (skup rješenja) evoluira prema boljim rješenjima. Svako potencijalno rješenje optimizacijskog problema naziva se *jedinkom*, dok se pod terminom *genom* podrazumijeva računalna reprezentacija tog rješenja. Nad populacijom se kroz niz *generacija* (koraka izvođenja algoritma) vrše genetske operacije.

Algoritam obično započinje populacijom slučajno generiranih rješenja (jedinki). Na temelju *dobrote* jedinki (engl. *fitness*), odnosno kvalitete rješenja, odabere se nekoliko jedinki iz trenutne populacije te ih se modificira (križanjem, te eventualno mutacijom) kako bi se stvorila nova populacija. Ta se nova populacija koristi u sljedećem koraku algoritma (generaciji).

Obično se genom zapisuje kao niz bitova (nula i jedinica). Takav zapis omogućuje jednostavno vršenje operacija križanja te mutacije. Na slici 2.3 prikazan je postupak križanja s jednom točkom prekida. Mutacija se uglavnom svodi na izmjenu jednog bita u genomu.

Genetski algoritam opisan je sljedećim pseudokodom:

```
GENETSKI ALGORITAM
1  generiraj početnu populaciju
2  izračunaj dobrotu (fitness) svih jedinki populacije
3  sve dok je c < broj_iteracija
4      radi odaberi najbolje jedinke za reprodukciju (selekcija)
5           stvori potomstvo operacijama rekombinacije i mutacije
6           izračunaj dobrotu potomstva
7           zamijeni najlošije jedinke iz populacije potomstvom
8           c ← c + 1
```



Slika 2.3. Križanje s jednom točkom prekida

Selekcijske metode koje se najčešće koriste su *turnirska* te *eliminacijska selekcija*, a više o njima može se pročitati u [3].

3. Problem naprtnjače

U nastavku teksta opisan je problem naprtnjače, problem iz klase NP, te je prikazan način rješavanja koristeći opisane heurističke metode.

3.1. Opis problema

Problem naprtnjače može se, u kratkim crtama, definirati na sljedeći način: za dani skup predmeta, gdje svaki predmet ima određenu težinu i vrijedost, potrebno je odabrati podskup predmeta tako da je težina odabranih predmeta manja ili jednaka određenom broju (*kapacitetu naprtnjače*), a ukupna vrijednost maksimalna.[1]

Formalna definicija je sljedeća: neka je W kapacitet naprtnjače, a S skup predmeta. Za svaki predmet $x \in S$ definiramo **težinu** $w(x)$ te **vrijednost** $p(x)$. Tražimo podskup $P \subseteq S$ takav da

$$\sum_{x \in P} p(x)$$

bude **maksimalno**, pod uvjetom da je

$$\sum_{x \in P} w(x) \leq W$$

Jedan od mogućih načina rješavanja problema naprtnjače je enumeracija svih rješenja, te odabir najboljeg. Kako je broj podskupova od S jednak 2^N (s N je označen broj elemenata, tj. $N = |S|$), takav je algoritam složenosti $O(2^N)$.

Ukoliko je kapacitet naprtnjače W relativno malen, problem se može riješiti algoritmom temeljenom na *dinamičkom programiranju* koji je vremenske složenosti $O(NW)$, a prostorne (memorijske) $O(W)$. Koncept dinamičkog programiranja prelazi opseg ovog rada, pa će taj algoritam biti opisan u kratkim crtama, bez dokazivanja točnosti.

Definirajmo funkciju $f(w, n)$ kao **optimalnu vrijednost** koju možemo dobiti popunjavanjem naprtnjače kapaciteta w s prvih n predmeta. Očito će konačno rješenje problema naprtnjače biti $f(W, N)$.

[1] Opisana je podvarijanta problema koja se naziva *0-1 knapsack problem*. Postoje i druge varijante: u nekima je moguće uzeti proizvoljni broj svakog predmeta, a u nekima je čak moguće uzeti samo dio određenog predmeta. U ovom radu obrađuje se samo 0-1 problem naprtnjače.

Vrijedi sljedeća rekurzivna relacija:

$$f(w,n) = \max\{\ f(w,n-1), f(w-w(n),n-1)+p(n)\ \}$$

s time da je $f(w,0)=0$. U svakom koraku imamo dvije mogućnosti: uzeti n-ti predmet, nakon čega nam se kapacitet smanjio za $w(n)$, ili ostaviti taj predmet nakon čega je kapacitet i dalje w. Ovako modelirano rješenje zadovoljava dvije osnovne karakteristike dinamičkog programiranja: svojstvo optimalnosti te svojstvo preklapajućih potproblema (više o toj tematici u [1]).

Dobro implementiran, navedeni algoritam ima memorijsku složenost $O(W)$. Za velike kapacitete to nije prihvatljivo, pa je potrebno poslužiti se heurističkim metodama.

3.2. Implementacije

Označimo rješenje vektorom $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, gdje je $x_i \in \{0,1\}$, a n označava broj predmeta. x_i jednako je jedinici ukoliko je u rješenju sadržan predmet s indeksom i. Lako se uočava da skup mogućih rješenja ima kardinalnost 2^n. Dopustivo rješenje je ono koje zadovoljava ograničenje kapaciteta, tj. vrijedi $\sum_{i=1}^{n} w_i x_i \leq W$.

Susjednost rješenja može se definirati na sljedeći način: rješenja $\mathbf{x}$ i $\mathbf{x'}$ su susjedna ($\mathbf{x'} \in N(\mathbf{x})$ te $\mathbf{x} \in N(\mathbf{x'})$) ukoliko postoji *samo jedan* j takav da je $x_j \neq x'_j$. Dakle, gledajući vektore rješenja kao binarne brojeve, oni su susjedni ukoliko se razlikuju samo u jednoj binarnoj znamenci (Hammingova udaljenost jednaka je jedinici). Zaključujemo da susjedno rješenje dobivamo tako da trenutnom rješenju dodamo novi predmet (pritom pazeći da se ne prekorači kapacitet) ili da jedan predmet uklonimo.

Penjanje uzbrdo

Kao što je opisano u odjeljku 2.1., algoritam penjanje uzbrdo u svakom će koraku zamijeniti trenutno rješenje **najboljim** rješenjem iz njegovog susjedstva, ukoliko je to rješenje bolje od trenutnog. Primijenjeno na problem naprtnjače, najbolje susjedno rješenje je ono koje se dobiva iz trenutnog dodavanjem **najvrijednijeg** predmeta takvog da se njegovim dodavanjem ne prekoračuje kapacitet naprtnjače. Ni u jednom koraku neće doći do uklanjanja određenog predmeta, budući da taj potez samo smanjuje "dobrotu" rješenja.

Promotrimo ponašanje algoritma penjanje uzbrdo na primjeru. Neka je kapacitet naprtnjače jednak 4, a dostupni predmeti prikazani tablicom 3.1. Ukoliko je početno rješenje $(0,0,0)$ (prazna naprtnjača), algoritam će u prvom koraku uzeti predmet vrijednosti 5 i težine 3, odnosno zamijeniti trenutno rješenje najboljim iz njegove okoline, a to je rješenje $(1,0,0)$. Međutim, daljnji koraci nisu mogući jer je najmanji od preostalih predmeta težine 2, a takav ne stane više u naprtnjaču. Kažemo da je algoritam

Tablica 3.1. Ulazni podaci za problem naprtnjače

	Težina	Vrijednost
1.	3	5
2.	2	3
3.	2	3

pronašao lokalno optimalno rješenje. To rješenje nije i globalno optimalno, budući da je bolje rješenje ono koje sadrži predmete 2 i 3, čija je ukupna vrijednost 6.

Lako se uoči kako, implementiran na ovaj način, algoritam penjanje uzbrdo nije nimalo bolji od **pohlepnog algoritma** – sortiranja predmeta po vrijednosti te uzimanja prvih m predmeta koji stanu u naprtnjaču. Međutim, algoritam penjanje uzbrdo najčešće kreće od nekog **slučajnog** dopustivog rješenja, što mu omogućuje da pronađe i druge lokalne optimume, a moguće i globalni optimum.[2]

Potrebno je još napomenuti kako se često u implementacijama penjanja uzbrdo koristi i metoda *ponovnog pokretanja* (engl. *restart*). Naime, već nakon nekoliko iteracija može se uočiti kako je algoritam došao do lokalnog optimuma, pa se trenutno rješenje sprema, a algoritam započinje iznova, s drugim slučajno generiranim početnim rješenjem. Algoritam bilježi i vraća najbolje pronađeno rješenje.

Simulirano kaljenje

Jedna od mogućih implementacija simuliranog kaljenja na problemu naprtnjače u svakom će koraku odabrati **slučajno susjedno rješenje** kojim će, ukoliko je bolje, zamijeniti trenutno rješenje. Ako je novo rješenje lošije od trenutnog, postoji određena vjerojatnost (koja je funkcija temperature i razlike dobrote tih rješenja, kao što je opisano u odjeljku 2.2.) da će se ipak prijeći u to rješenje.

Objašnjeno matematičkim rječnikom, u svakoj će se iteraciji na sreću odabrati broj j iz intervala $[1, n]$, te će se izvršiti jedna od sljedeće dvije radnje:

- ako je $x_j = 0$ te se dodavanjem predmeta s indeksom j ne prekoračuje kapacitet naprtnjače, postavlja se $x_j = 1$ (novo rješenje je bolje od prethodnog)
- ako je $x_j = 1$, postavlja se $x_j = 0$ (uklanja se predmet s indeksom j, odnosno prelazi u lošije rješenje) s vjerojatnošću[3] $e^{-p_j/T}$.

Važno je napomenuti kako rezultat dobiven simuliranim kaljenjem uvelike ovisi o parametrima T i koeficijentu α (s kojim temperatura linearno opada), kao i o broju

[2] Primjerice, ukoliko bi početno rješenje bilo $(0, 1, 0)$, algoritam bi odmah u prvom koraku došao do rješenja $(0, 1, 1)$ što je i globalno optimalno rješenje.

[3] Jasno je da je razlika dobrote trenutnog i novog rješenja jednaka upravo vrijednosti predmeta koji se uklanja.

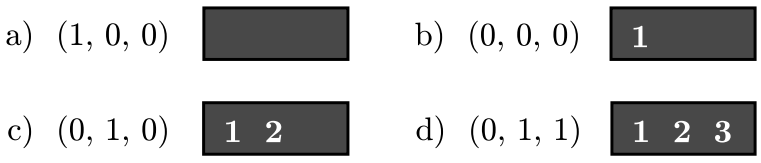
iteracija. Primjerice, ako su vrijednosti predmeta mali prirodni brojevi, velika temperatura uzrokovat će da vjerojatnost prelaska u lošija rješenja bude jako blizu jedinici, neovisno o razlici dobrote dvaju rješenja, što je izrazito nepoželjno.

Općenito je slučaj s heurističim algoritmima, zbog njihove prirode, da je potrebno podešavati parametre ovisno o instanci problema, kao i izvršavati algoritam više puta jer su kod svakog pokretanja mogući različiti rezultati.

Tabu pretaživanje

Tražeći najbolje rješenje u susjedstvu trenutnog rješenja prilikom rješavanja problema naprtnjače tabu pretraživanjem, pokazalo se ([2]) da je bolje gledati **omjere vrijednosti i težine** umjesto samih vrijednosti. Dakle, jedna od mogućih implementacija je sljedeća: tabu lista sadrži indekse $j \in [1, n]$, tako da predmet čiji je indeks na tabu listi nije moguće dodavati ili uklanjati iz trenutnog rješenja. Algoritam u svakoj iteraciji radi jedno od sljedećeg:

- ako postoji barem jedan indeks j takav da je $x_j = 0$, dodavanjem predmeta s indeskom j ne prekoračuje se kapacitet naprtnjače te j nije na tabu listi, odabire se onaj j za koji je omjer p_j/w_j **maksimalan** te se postavi $x_j = 1$
- u protivnom, odabire se indeks j takav da je $x_j = 1$, j nije na tabu listi, a p_j/w_j je **minimalno**, te se postavi $x_j = 0$.



Slika 3.1. Prikaz rada tabu pretraživanja.

Slika 3.1 prikazuje trenutno rješenje te izgled tabu liste u svakoj iteraciji algoritma tabu pretraživanje na primjeru problemu naprtnjače kapaciteta 4, s predmetima navedenim u tablici 3.1. Početno, slučajno generirano rješenje je $(1, 0, 0)$, a tabu lista je prazna. Kako nema predmeta koji bi se mogao dodati u naprtnjaču, odabire se predmet s najmanjim p_j/w_j za uklanjanje. Jedini takav predmet je onaj s indeksom 1, pa se on uklanja, a njegov se indeks dodaje u tabu listu. U sljedećem koraku odabire se predmet kojem je omjer p_j/w_j maksimalan te se taj predmet dodaje rješenju. Najveći omjer ima predmet s indeksom 1, no, budući da se nalazi u tabu listi, preskače se te

se rješenju dodaje predmet 2. U zadnjem koraku dodaje se predmet 3, te se postiže i optimalno rješenje s vrijednošću 6.[4]

Veličina tabu liste i broj iteracija su, dakako, ključni parametri te se podešavaju ovisno o konkretnom problemu.

Genetski algoritam

Zapis rješenja vektorom $(x_1, x_2, \ldots, x_n)$ idealan je za obradu genetskim algoritmom. U tablici u sljedećem odjeljku prikazani su rezultati izvršavanja genetskog algoritma prema implementaciji iz [4], koja koristi metodu križanja s jednom točkom prekida (slika 2.3), te tzv. *roulette wheel* metodu selekcije roditelja nad kojima će se vršiti križanje i mutacija.

Usporedba

Tablica 3.2 prikazuje prosječan rezultat izvršavanja (svaki od programa pokrenut je deset puta) opisanih algoritama na nekoliko manjih instanci problema. Usporedbe radi, za svaku instancu navedeno je i optimalno rješenje. Algoritmi penjanje uzbrdo i simulirano kaljenje izvršeni su u 10.000 iteracija, a tabu pretraživanje u 200 iteracija. Parametri SA algoritma su $T = 10000$ i $\alpha = 0.99999$, a duljina tabu liste je $L = 10$. Genetski algoritam izvršen je u 200 generacija, veličina populacije bila je 20, a vjerojatnost mutacije 0.02.

Tablica 3.2. Usporedbe rezultata pojedinih algoritama

W	N	optimalno	HC	SA	TS	GA
4	3	6	5.7	6	6	6
100	10	2677	1594.8	2677	2469.4	2546.2
100	20	2748	1940.2	2748	2687.8	TODO
1000	30	2562	1452.5	2541	2538.8	TODO
1000	50	4533	2392.9	4293.4	4335.6	TODO

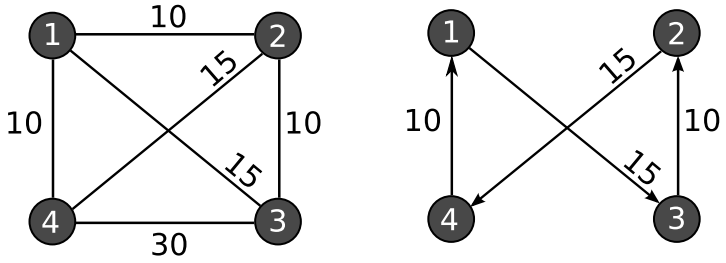
[4] Važno je još jednom naglasiti kako nije nužno da će dobiveno rješenje biti globalno optimalno. Primjerice, da je algoritam započeo s rješenjem $(0, 0, 0)$, najbolje rješenje koje bi pronašao bilo bi $(1, 0, 0)$, koje ima vrijednost 5.

4. Problem trgovačkog putnika

U uvodnom dijelu rada spomenut je problem trgovačkog putnika. U ovom poglavlju taj je problem matematički formuliran, te je prikazan način rješavanja određenim heurističkim metodama.

4.1. Opis problema

Problem trgovačkog putnika, definiran rječnikom teorije grafova, je sljedeći: u **potpunom grafu** $G = (V, E)$ pronaći Hamiltonov ciklus[1] najkraće duljine.



Slika 4.1. Potpuni težinski graf s 4 vrha i njegov Hamiltonov ciklus

Kao što je navedeno u uvodnom dijelu, algoritam koji rješava problem trgovačkog putnika ispitivanjem svakog ciklusa ima složenost $O(2^N)$, gdje je N broj vrhova grafa ($N = |V|$). Međutim, postoji i bolje rješenje koje nalazi najkraći Hamiltonov ciklus. Promotrimo ciklus $v_1 \rightsquigarrow v_{k-1} \rightarrow v_k \rightarrow v_{k+1} \rightsquigarrow v_n \rightarrow v_1$, $v_i \in V(G)$. Uočimo kako je dovoljno promatrati samo **najkraći** put između vrhova v_1 i v_k koji **prolazi vrhovima** $v_2, \ldots, v_{k-1}$ (ne nužno tim redosljedom, dakako). Princip po kojim optimalno rješenje problema u sebi sadrži optimalna rješenja podproblema (manjih instanci istog problema) naziva se *principom optimalnosti* te je jedan od temelja dinamičkog programiranja. Primjenom dinamičkog programiranja dobiva se algoritam za rješavanje problema trgovačkog putnika čija je vremenska složenost $O(N^2 \cdot 2^N)$. Takav je algoritam puno bolji od iscrpne pretrage koja ima faktorijelnu složenost ($N!$ puno brže raste

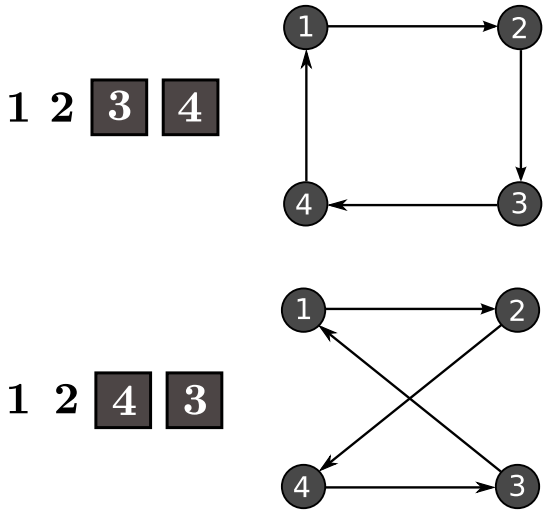
[1] Ciklus koji prolazi svakim vrhom točno jedamput.

od 2^N), međutim, prostorna složenost takvog algoritma je $O(N \cdot 2^N)$, pa je primjenjiv za brzo pronalaženje Hamiltonovog ciklusa u grafovima s najviše dvadesetak vrhova.

Za približno rješavanje problema trgovačkog putnika u razumnom vremenu mogu se upotrijebiti opisane heuristike. U nastavku je navedena jedna od mogućih implementacija

4.2. Implementacije

Označimo rješenje vektorom $\mathbf{v} = (v_1, v_2, \ldots, v_n)$, $v_i \in V(G)$. Taj vektor predstavlja *redosljed obilaska* vrhova u grafu. Susjedno rješenje **v'** može se definirati kao svako rješenje koje se može dobiti iz **v** slučajnim odabirom dva vrha te zamjenom redosljeda obilaska ta dva vrha (slika 4.2).[2] Dobrota rješenja funkcija je čiji je iznos duljina ciklusa pojedinog rješenja, odnosno $\sum_{i=2}^{n} w(v_{i-1}, v_i) + w(v_n, v_1)$, te se ona nastoji minimizirati.



Slika 4.2. Zamjena redosljeda obilaska dva vrha

Definiravši susjedstvo, implementacija algoritama ne razlikuje se puno od implementacije na problemu naprtnjače. Početno stanje se u mnogim implementacijama ne odabire slučajno, već se pronalazi pohlepnim algoritmom (gradi se ciklus brid po brid, birajući uvijek kao sljedeći brid onaj koji je dopustiv[3], a najmanje je duljine).

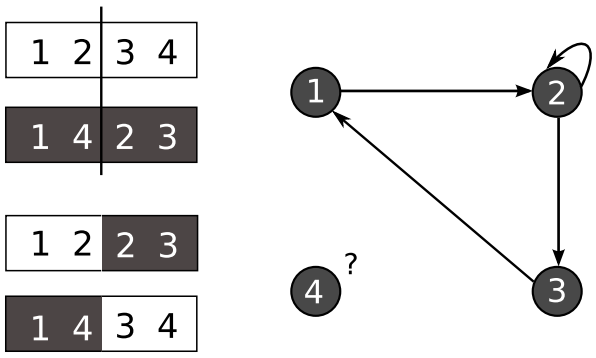
Za razliku od implementacije na problemu naprtnjače gdje je sadržavala po jedan broj, tabu lista kod *tabu search* algoritma primijenjenog na problem trgovačkog putnika

[2]Susjedstvo se može definirati na više različitih načina, te o dobrom odabiru ovisi uspješnost algoritma. Loše definiranom relacijom susjedstva bit će istražen samo manji dio prostora rješenja. U [5] navedeno je nekoliko načina modeliranja relacije susjedstva.

[3]Nadovezuje se na prethodni vrh, te ne zatvara ciklus prije nego prođe svim vrhovima.

sadrži parove vrhova kojima je u jednoj od prethodnih iteracija izmjenjen redosljed obilazenja ([6]).

Zapis rješenja vektorom **v** pogodan je i za genetski algoritam. Međutim, poteškoća se javlja kod operacije križanja. Naime, ukoliko se izvrši križanje s jednom točkom prekida, vrlo je vjerojatno da rješenja-potomci neće sadržavati Hamiltonov ciklus. U [7] predlaže se tzv. metoda *pohlepnog križanja* – odabere se prvi vrh u ruti jednog od roditelja te se uspoređuju njemu susjedni vrhovi u oba rješenja-roditelja; odabire se najbliži vrh te se njime nastavlja ciklus.



Slika 4.3. Križanje s jednom točkom prekida genoma **v**

Tablica 4.1 prikazuje prosječan rezultat izvršavanja algoritama penjanje uzbrdo i simulirano kaljenje na manjim instancama problema. Broj iteracija je 1 000 000, $T = 1000$, $\alpha = 0.99999$.

Tablica 4.1. Usporedbe rezultata pojedinih algoritama

N	optimalno	HC	SA
4	12	12	12
4	195	195	195
6	179	179	179
6	223	252	223
10	172	228	174.2
20	207	305	239.4

5. Zaključak

U prethodnim poglavljima opisana su dva problema čije se optimalno rješenje ne može pronaći u razumnom vremenu. U stvarnosti postoji velik broj takvih problema. Osim navedenih, neki od poznatijih su i problem N kraljica (raspoređivanje N kraljica na šahovsku ploču dimenzija NxN tako da se međusobno ne napadaju), problem faktorizacije velikih cijelih brojeva, ispitivanje izomorfnosti grafova, te n-SAT (od engl. *satisfiability*) problem [1].

U praksi je kod takvih problema često zadovoljavajuće i ono rješenje koje je približno jednako optimalnom. Primjerice, trgovački putnik će se sigurno zadovoljiti rutom koja je nešto duža od optimalne, umjesto da čeka stotine godina da se takva ruta pronađe. Heuristički algoritmi su oni algoritmi koji će često dati rješenje koje je blizu optimalnom, no ne može se garantirati da će to uvijek biti slučaj.

Primjerice, problem trgovačkog putnika mogli bismo pokušati rijesiti sljedećim algoritmom:

```
TSP-RANDOM
1  opt ← ∞
2  sve dok nije prekoračeno vremensko ograničenje
3      radi y ← slučajno generirano rješenje
4          ako je m(x) < opt
5              onda opt ← m(x)
```

Jasno je kako vjerojatnost da takav algoritam pronađe optimalno rješenje opada s porastom broja mogućih rješenja. Heuristički algoritmi opisani u ovom radu pametniji su načini pretrage prostora problema. Penjanje uzbrdo kreće od na sreću odabranog rješenja te ga poboljšava dok ne dođe do razine kad više ne može unaprijediti rješenje. Simulirano kaljenje je unaprijeđenje penjanja uzbrdo koje u početku dopušta prelaske u lošija rješenja. Tabu pretraživanje koristeći memorijsku strukturu onemogućava ponavljanje istih rješenja, ili rješenja s istim karakteristikama, što ima za posljedicu pretragu većeg dijela prostora rješenja, dok genetski algoritam u svakom koraku održava veći broj rješenja (populaciju), te različitim operacijama nad njima postiže bolja rješenja (preživljavanje najboljih).

Važno je naglasiti kako je ključni dio konstrukcije algoritma definicija modela, odnosno zapisa rješenja. Kod problema naprtnjače pokazalo se kako vrlo jednostavan zapis, u kojem se susjedna rješenja razlikuju u jednoj binarnoj znamenci, daje prilično dobre rezultate. Također, veliki utjecaj na rezultat heurističkih algoritama imaju i parametri, kao što su temperatura kod simuliranog kaljenja ili veličina populacije te vjerojatnost mutacije kod genetskog algoritma. Parametre valja podešavati ovisno o problemu, odnosno instanci problema.

Heuristički algoritmi primjenjivi su na velikom broju optimizacijskih problema te, uz dobru implementaciju, mogu dati iznenađujuće dobre rezultate.

6. Sažetak

U prvom dijelu rada opisana su četiri heuristička algoritma: penjanje uzbrdo, simulirano kaljenje, tabu pretraživanje i genetski algoritam, te su prikazane neke njihove mane i prednosti. Definirani su i ključni pojmovi kao št su *optimizacijski problem*, *prostor rješenja* itd. U drugom dijelu opisana su dva NP problema: problem naprtnjače i problem trgovačkog putnika. Prikazani su algoritmi temeljeni na dinamičkom progrmiranju koji optimalno rješavaju manje instance tih problema, a zatim je opisan prikaz rješenja te nekoliko mogućih načina rješavanja tih problema primjenom opisanih heurističkih metoda. Zaključuje se kako su heuristički algoritmi moćan alat te su, uz dobar zapis rješenja, primjenjivi na velikom broju različitih optimizacijskih problema.

Bibliografija

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein: *Introduction to Algorithms*, The MIT Press, 2001.

[2] A. Nakić: *Heuristički algoritmi za 0-1 problem naprtnjače*, http://e.math.hr/heuristicki/ (10. 3. 2008.)

[3] M. Golub: *Genetski algoritam*, http://www.zemris.fer.hr/~golub/ga/ga.html (10. 3. 2008.), Fakultet elektrotehnike i računarstva, 2004.

[4] J. S. Cameron: *An Overview Of Artificial Life With A Focus On Genetic Algorithm And Genetic Programming*, http://www.alesdar.org/oldSite/IS/ (10. 3. 2008.)

[5] C. Nilsson: *Heuristics for the Traveling Salesman Problem*, http://www.ida.liu.se/~TDDB19/reports_2003/htsp.pdf (10. 3. 2008.)

[6] S. Jayaswal: *A Comparative Study of Tabu Search and Simulated Annealing for Traveling Salesman Problem*, http://www.eng.uwaterloo.ca/~sjayaswa/projects/MSCI703_project.pdf (10. 3. 2008.)

[7] CodeProject: *Genetic Algorithms and the Traveling Salesman Problem*, http://www.codeproject.com/KB/recipes/tspapp.aspx (10. 3. 2008.)