FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVOD ZA ELEKTRONIKU, MIKROELEKTRONIKU, RAČUNALNE I INTELIGENTNE SUSTAVE

KONSTRUKCIJSKI PROGRAM

# TEMA: **VIŠEDRETVENA IMPLEMENTACIJA SIMULIRANOG KALJENJA**

Mentor: Prof.dr. LEO BUDIN

**LEONARDO JELENKOVIĆ**

Mat.br. 36328116

Računarska tehnika

Zagreb, Rujan 1996.

# SADRŽAJ

Stranica

# 1. UVOD

**Simulirano kaljenje** efikasna je metoda za riješavanje problema kombinatoričke optimizacije, a ime zaslužuje po analogiji sa kaljenjem metala iz čije se simulacije i razvila. Metoda pokušava pronaći globalni optimum, ili mu se bar približiti, u zadanome prostoru konfiguracija. Najpoznatije primjene metode su kod problema trgovačkog putnika, smještaja VLSI ćelija, podijele grafa, rasporeda zadataka i drugi. U mnogim objavljenim radovima pokazuje se veliki uspjeh metode.

Glavni nedostatak metode jest dugo vrijeme računanja, koje uzrokuje slijedna priroda procesa kaljenja. Preciznije, simulirano kaljenje je strogo slijedna metoda zbog ovisnosti iteracije *i+1* o iteraciji *i*. Naime, iteracija *i* može promjeniti podatke koje koristi iteracija *i+1*. Na paralelizaciji tog algoritma dosta se radilo te su razvijene metode koje možemo svrstati u dvije osnovne kategorije. Prva, *sinkrona*, nastoji slijediti isti niz odluka kao i slijedna metoda, dok druga, *asinkrona* ne, te se ona u ovom radu ne razmatra.

Paralelizacija se sprovodi spekulativnim računanjem, računanjem iteracije *i+1*, *i+2*,... prije nego li je iteracija *i* završila. Prilikom računanja iteracije *i+1* potrebna su dva računanja: jedno koje pretpostavlja prihvaćanje prijelaza prethodne iteracije *i* i drugo koje pretpostavlja odbacivanje. Svaka takva grana za slijedeću iteraciju se račva na dva dijela, tj. na PRIHVATI i ODBACI granu. Tako se formira binarno stablo čija veličina ovisi o broju paralelnih iteracija koje želimo izračunati.

Npr. za 3 iteracije porebno nam je 7 procesa računanja koje, ako imamo na raspolaganju 7 procesora, možemo obaviti u jednom koraku. Metoda je po svojoj strukturi dobila ime *binarna spekulativna metoda*. Binarno stablo u kojem su sve grane iste duljine naziva se *uravnotežno binarno stablo*. Stablo, međutim, ne mora biti u ravnoteži. Ako je npr. veća vjerojatnost odbijanja prijelaza poželit ćemo da grana ODBACI ima veću duljinu, tako da u prosijeku jednim paralelnim računanjem izračunamo veći broj iteracija.

Uzmemo li krajnji slučaj, tj. slučaj samo jedne grane, i to grane ODBACI dobivamo metodu koju je *A. Sohn* nazvao *generalizirana spekulativna metoda*, ili *N-paralelna spekulativna metoda*. Koristeći tu metodu *Sohn* je u svom članku [1] prikazao dobivene rezultate koristeći višeprocesorski (masivno) paralelni sustav sa distribuiranom memorijom.

U ovom radu primjenjena je ta metoda na *problem trgovačkog putnika*, koristeći višedretvenost *Solaris 2.4* UNIX sustava, uvodeći preslikavanje: jedan procesor u jednu dretvu.

# 2. SIMULIRANO KALJENJE

## *2.1 Razvoj algoritama kombinatoričke optimizacije*

U zadnjih nekoliko desetljeća značajno se povećala uloga optimizacije u mnogo različitih područja, kao što su elektrotehnika, operacijska istraživanja, računarstvo i komunikacije. Linearne i nelinearne optimizacije, optimum funkcije kontinuiranih varijabli, doživjeli su značajan napredak tijekom 50-tih i 60-tih godina. Bolji rezultati kombinatoričke optimizacije, optimumi funkcija diskretnih varijabli postignuti su kasnije (70-tih). Npr. za problem trgovačkog putnika napravljeni su vrlo kvalitetni algoritmi koji pronalaze aproksimaciju ili optimum u prihvatljivom vremenu za probleme do određenog broja gradova.

Međutim, problemi kombinatoričke optimizacije većeg razmjera predstavljaju i danas problem, uprkos bržim računalima. Takvi se problemi riješavaju aproksimacijom, gdje je kvaliteta riješenja proporcionalna vremenu računanja.

Kombinatorička optimizacija se može formalizirati kao par $(\mathcal{R},C)$, gdje je $\mathcal{R}$ konačan skup konfiguracija i $C$ funkcija cilja: $C:\mathcal{R}\rightarrow\boldsymbol{R}$, koja svakoj konfiguraciji pridjeljuje broj. Problem je pronaći takvu konfiguraciju za koju je funkcija $C$ najmanja, tj. treba pronaći takvu konfiguraciju i za koju vrijedi:

$$C_{\mathrm{OPT}} = C(i_0) = \min C(i),$$

gdje je $i \in \mathcal{R}$, a $C_{\mathrm{OPT}}$ predstavlja minimalnu vrijednost funkcije cilja.

Postoje dva pristupa tom problemu: prvi, traženje globalnog optimuma po svim konfiguracijama, što je nemoguće u prihvatljivom vremenu, i drugi, aproksimacijski algoritam, koji pokušava dobiti zadovoljavajuće rješenje u prihvatljivom vermenu.

*Algoritam simuliranog kaljenja* spada u ovu drugu skupinu. Algoritam je baziran na stohastičkoj tehnici, ali uključuje mnoge dijelove algoritma *iterativnog poboljšanja*, zato će on biti prije objašnjen.

*Algoritam iterativnog poboljšanja* pretpostavlja zadanu definiciju konfiguracija, funkcije cilja i mehanizam generiranja susjedne konfiguracije malom promjenom. Počevši od početne konfiguracije, generira se niz iteracija, od kojih se svaka sastoji od mogućeg prijelaza u susjednu. Ukoliko ta susjedna konfiguracija daje smanjenje funkcije cilja onda se ona prihvaća, a inače se odbacuje i traži slijedeća susjedna. Algoritam završava kada sve susjedne konfiguracije daju povećanje funkcije cilja.

Nedostaci algoritma su slijedeći:

1. prema definiciji, algoritam završava u lokalnom minimumu i neznamo koliko je on udaljen od globalnog minimuma,
2. dostignuti minimum ovisi o početnoj konfiguraciji.

Prednosti algoritma su njegova opća primjenjivost te relativno kratko vrijeme računanja za jednu početnu konfiguraciju.

Moguća poboljšanja su slijedeća:

1. izvesti algoritam za veliki broj početnih konfiguracija
2. upotreba informacija iz prethodnih pokušaja
3. povećanje skupa susjednih konfiguracija

4. prihvatiti prijelaze koji uzrokuju povećanje funkcije cilja.

Prvi pristup jest tradicionalni način riješavanja problema kombinatoričke optimizacije. Drugi i treći pristup jako ovise o problemu, pa se oni neće niti razmatrati.

Algoritam koji slijedi zadnji pristup poboljšanja predstavili su nezavisno ***S. Kirkpatrik*** ('83.) i ***V. Černy***. Poznat je pod nazivom *simulirano kaljenje*, ali se u literaturi nalazi još i pod imenima *Monte Carlo kaljenje, stohastičko hlađenje, stohastička relaksacija* i *algoritam slučajne zamjene*.

Rezultati dobiveni ovim algoritmom su vrlo blizu optimuma te ne ovise o početnoj konfiguraciji. Tako simulirano kaljenje ne pokazuje nedostatke metode *iterativnog poboljšanja* te daje bolje rezultate, ali je nešto sporija.

## *2.2 Algoritam simuliranog kaljenja*

U svom orginalnom obliku, algoritam simuliranog kaljenja, baziran je na analogiji između simulacije kaljenja metala i problema kombinatoričke optimizacije. U fizici krutih tvari, *kaljenje* predstavlja proces u kojemu se materija zagrije do maksimalne temperature (temp. taljenja) kod koje je unutarnja struktura stohastički organizirana. Slijed postupaka laganog hlađenja uzrokuje da unutarnji raspored elementarnih jedinica poprima stanje sve manje energije, prilagođavajući se temperaturi (unutarnja energija proporcionalna je temperaturi). Pri svakoj temperaturi T materijal dostiže termičku ravnotežu, karakteriziranu vjerojatnošću posjedovanja energije **E** prema *Boltzmanovoj raspodijeli*:

$$P(\mathbf{E}) = \frac{1}{Z(T)} \cdot \exp(-\frac{\mathbf{E}}{k_B \cdot T}),$$

gdje je Z(T) faktor normalizacije, $k_B$ *Boltzmanova konstanta*. Kako se temperatura smanjuje prema nuli, samo ona stanja minimalne energije imaju vjerojatnost pojavljivanja veću od nule.

Također je poznato, da ako je hlađenje prebrzo, tj. ako se pri svakoj temperaturi ne postiže termalna ravnoteža, struktura materijala postaje amorfna, a ne kristalna rešetka niske unutarnje energije.

Da bi simulirao dostizanje termalne ravnoteže, ***N. Metropolis*** je koristio slijedeću metodu. Uz trenutnu konfiguraciju strukture napravi se pomak jedne čestice. Ako je promjena energije uzrokovana tom promjenom $\Delta\mathbf{E}$ negativna onda se nova konfiguracija prihvaća. Ako je pak $\Delta\mathbf{E}>0$ tada je vjerojatnost prihvaćanja određena sa: $\exp(-\Delta\mathbf{E}/k_BT)$. Nakon mnogo ovakvih slučajnih pomaka sustav postiže termalnu ravnotežu. Nakon dostizanja termalne ravnoteže smanjuje se temperatura i ponavlja postupak sve do temperature smrzavanja sustava.

Algoritam je primjenjiv na problem kombinatoričke optimizacije, uz preslikavanje: jedna konfiguracija strukture u jednu konfiguraciju problema, energija u funkciju cilja i temperatura T u kontrolni parametar c.

Slijedi cijeli algoritam u obliku pseudoprograma.

```
procedura simulirano_kaljenje(i0,c0);
    i:=i0;
    c:=c0;
    Ci:=C(i);
    ponavljaj
        ponavljaj
            j:=susjedna_konf(i);
```

```
                Cj:=C(j);
                ΔC:=Cj-Ci;
                     prihvati:=FALSE;
                 ako je ΔC<0 tada
                          prihvati:=TRUE;
                inače
                          ako je exp(-ΔC/c)>random[0,1) tada
                                    prihvati:=TRUE;
                     ako je prihvati=TRUE tada
                               i:=j;
                          Ci:=Cj;
          do termalne_ravnoteže
     do zamrzavanja
kraj.
```

Uz ovaj algoritam treba još odabrati tehniku hlađenja, tj. treba:

- izabrati početnu vrijednost $c_0$
- izabrati konačnu vrijednost $c_f$
- odrediti funkciju hlađenja.

Početna vrijednost $c_0$ odabire se tako da su u početku gotovo svi prijelazi prihvaćeni, tj. da budu prihvaćeni sa vjerojatnoscu $p_0$. Ako sa $\Delta C^+$ označimo prosječno povećanje funkcije cilja, tada vrijedi:

$$p_0 = \exp(-\Delta C^+/c_0),$$

iz čega nam slijedi:

$$c_0 = \Delta C^+/\ln(1/p_0).$$

Što se tiče $c_f$, obično se unaprijed zada određeni broj koraka (10-50). Postoje mnoge funkcije hlađenja predložene u literaturi [2], ali se najčešće koristi najjednostavnija, množenje sa brojem manjim od 1, tipično u granicama [0.5,0.99].

# 3. VIŠEDRETVENOST

Standardni proces u operacijskom sustavu UNIX-a ima samo jednu *dretvu* kontrole toka programa. Za razliku od njega, program koji koristi *višedretvenost* omogućuje pojavu više *dretvi* kontrole programa, koje su međusobno nezavisne i mogu se izvoditi paralelno. *Dretvu* možemo shvatiti kao jedan proces. Ona ima svoje programsko brojilo, vlastiti stog, signalnu masku, prioritet te identifikacijski broj, ali se nalazi u istom adresnom prostoru početnog procesa što *višedretvenost* čini boljim prema više-procesnom radu. *Dretve* tako, dijeleći globalne varijable u slučaju promjene istih, ona je odmah vidljiva svim ostalim *dretvama*.

O raspodijeli *dretvi* na raspoložive procesore brine sam operacijski sustav, odnosno, ako ima više procesora tada OS svakom pridružuje jednu *dretvu*. Ukoliko ima više *dretvi* nego procesora tada operacijski sustav svakoj *dretvi* pridjeljuje procesor na određeno vrijeme.

*Višedretvenost* donosi mnoga poboljšanja, od kojih je već spomenuto korištenje manje resursa sustava. Slijedeće poboljšanje jest preglednost i razumljivost programa koji su pisani za *višedretveni* sustav, jer je on prirodniji samom paralelnom algoritmu. Također dobivamo na brzini, jer, pored bržeg kreiranja novih *dretvi* od novih procesa, sinkronizacija među *dretvama* je brža od one među procesima. U ovom radu korištena je *višedretvenost* operacijskog sustava ***Solaris 2.4***.

# 4. PROBLEM TRGOVAČKOG PUTNIKA

Problem trgovačkog putnika je najpoznatiji problem kombinatoričke optimizacije. Zadan je sa N gradova i njihovim međusobnim udaljenostima. Zadatak je trgovačkog putnika da krenuvši iz početnog grada $G_0$, obiđe sve gradove samo jedanput i vrati se u početni grad. Problem jest u tome što treba pronaći najkraći put. Sveukupno ima ***(N-1)!*** različitih mogućih puteva što je prepreka egzaktnom traženju minimalnog put. Problem je dakle NP težak, te se zato on riješava aproksimacijskim algoritmima kao što je simulirano kaljenje. Preslikavanje je slijedeće: jedan put predstavlja jednu konfiguraciju, duljina puta funkciju cilja. Susjedna konfiguracija se može generirati na različite načine. U ovom radu koristi se metoda zamijene puta između dva, na slučajan način odabrana, grada. Početnu vrijednost kontrolnog parametra c izračunava se iz prosječnog povećanja duljine puta prema izrazu:

$$c = \Delta C^{+}/\ln(1/p_0),$$

gdje je $p_0$ vjerojatnost prihvaćanja nove konfiguracije na početku iteriranja. Iteriranje unutarnje petlje, koje je u općenitom algoritmu, označeno sa "do termalne ravnoteže" u praksi se provodi tako da se odredi vrijednost koja je proporcionalna opsegu problema. U ovom slučaju je to vrijednost proporcionalna sa $N^2$, odnosno, za neki faktor manja ($\beta N^2$). Vanjska petlja određuje broj koraka hlađenja, a obično iznosi od 10 do 50 koraka. U nastavku se nalazi cijeli algoritam slijednog simuliranog kaljenja

primjenjenog na problem trgovačkog putnika.

```
procedura TSP(N,Dmax,S,p0,α ,DML);
     za i:=1 do N radi
          Gx(i):=random(0,1)*Dmax;
          Gy(i):=random(0,1)*Dmax;
     put:=d[G(1),G(N)];
     za i:=1 do N-1 radi
          put:=put+d[G(i),G(i+1)];
     pros:=0;
     brojac:=0;
     za i:=1 do 100 radi
          a:=random(0,1)*N;
          b:=random(0,1)*N;
          dput:=d[G(a),G(b+1)]+d[G(b),G(a-1)]
          -d(G(a-1),G(a)]-d[G(b),G(b+1)];
     ako je dput>0 tada
          pros:=pros+dput;
          brojac:=brojac+1;
     c:=pros/brojac/ln(1/p0);
     za i:=1 do S radi
          za j:=1 do DML*N2 radi
               a:=random(0,1)*N;
               b:=random(0,1)*N;
```

```
                dput:=d[G(a),G(b+1)]+d[G(b),G(a-1)]
                     -d(G(a-1),G(a)]-d[G(b),G(b+1)];
                prihvati:=FALSE;
                ako je dput<0 tada
                     prihvati:=TRUE;
                inače
                     ako je exp(-dput/c)>random[0,1) tada
                          prihvati:=TRUE;
                ako je prihvati=TRUE tada
                     za n:=0 do (b-a)/2+1 radi
                          pom:=G(a+n);
                          G(a+n):=G(b-n);
                          G(b-n):=pom;
                     put:=put+dput;
          c:=c*α;
kraj.
```

Radi jednostavnosti pretpostavlja se da se gradovi nalaze u kvadratu duljine stranice $D_{MAX}$. Udaljenost između dva grada *i* i *j*, tako predstavlja geometrijsku udaljenost točaka koje reprezentiraje gradove, tj. *d[G(i),G(j)]*. Radi brzine računanja, te se udaljenosti mogu prethodno izračunati i smjestiti u matricu. Kao što se vidi iz algoritma, povećanje duljine puta ne računa se tako da se izračuna duljina novog puta, pa se onda oduzima od starog, već se odmah računa samo promjena. Naime, prije promjene put je, npr. išao od početnog grada *G(1),..., G(a-1), G(a), G(a+1),...,G(b-1), G(b), G(b+1), ... , G(N), G(1),* a nakon zamijene mijenja se dio puta samo između *G(a-1)* i *G(b+1)*, tj. sada put glasi *G(1), ..., G(a-1), G(b), G(b-1), ..., G(a+1), G(a), G(b+1), ..., G(N)*. Pošto je duljina puta neovisna o smjeru obilaska vidljiva je razlika u duljini puta koja je jednaka upravo navedenom izrazu u algoritmu.

# 5. ALGORITAM N-PARALELNOG SIMULIRANOG KALJENJA

U uvodu je objašnjen problem paralelizacije simuliranog kaljenja te neki mogući pristupi. U ovom radu slijedi se tzv. *sinhroni pristup*, odnosno pristup kod kojega se slijedi isti niz odluka kao i kod slijedne metode. Upotrijebljeni algoritam paralelno računa najviše N iteracija pomoću N *dretvi* višedretvenog sustava. Postoji jedna *glavna* dretva, koja nakon završetka jedne razine računanja prikuplja podatke ostalih, *podređenih* dretvi te zamijenjuje gradove koji su određeni onom dretvom sa najmanjim iteracijskim brojem uz potvrdnu odluku o prijelazu. Slijedeća razina započinje od prve slijedeće iteracije. Npr. ako imamo 6 dretvi koje rade paralelno te računaju iteracije 1, 2, 3, 4, 5 i 6, i na kraju računanja ustanovljeno je da dretve koje računaju iteracije 3 i 5 imaju potvrdnu odluku o prijelazu. Tada glavna dretva prihvaća prijelaz treće iteracije, odnosno zamijenjuje gradove prema njoj, a slijedeća razina računanja započinje sa iteracijom 4, odnosno, paralelno se računaju iteracije 4, 5, 6, 7, 8 i 9. Naime, podaci sa kojima su računale dretve u iteracijama 4,5 i 6 nisu valjani te se one moraju ponoviti. Osim što prikuplja i vrši odgovarajuće prijelaze, glavna dretva upravlja sinhronizacijom niti, tj. određuje početak računanja, čeka da sve dretve završe sa računanjem, obavlja zamijenu, određuje novu početnu iteraciju te ponavlja postupak računanja signalizirajući start.

Osnovni algoritme glavne i podređenih dretvi:

```
procedura glavna_dretva;
      iter:=1;
      za i:=1 do S radi
           j:=1;
           ponavljaj
               označi_start;
               ΔC:=prijelaz(iter);
               zastavica(1):=FALSE;
               ako je (ΔC<0) ∨ (exp(-ΔE/c)>random[0,1)) tada
                     zastavica(1):=TRUE;
               pričekaj_podređene_dretve;
               n:=0;
               ponavljaj
                     n:=n+1;
               dok je (zastavica(n)=FALSE) ∧ (n<=BROJ_NITI)
               iter:=iter+n;
               j:=j+n;
               ako je (n<=BROJ_NITI) ∧ (j<=N²*DML) tada
                     prihvati_prijelaz(iter);
                     j:=j+1;
                     iter:=iter+1;
          dok je j<=N²*DML;
     c:=c*α;
kraj
```

```
procedura podređena_dretva(ID);
     ponavljaj
          čekaj_start;
          ΔC:=prijelaz(iter+ID);
          ako je (ΔC<0) ∨ (exp(-ΔC/c)>random[0,1)) tada
               prihvati:=TRUE;
          inače
               prihvati:=FALSE;
          označi_kraj;
     do zauvijek;
kraj.
```

Navedenom treba još dodati i inicijalizacijski dio. Prijelaz je jednoznačno određen pomoću broja iteracije *iter* koji služi kao 'sjeme' prilikom generiranja slučajnog prijelaza te je on dovoljna informacija glavnoj dretvi, uz zastavice koje nose odluku.

# 6. PROGRAMSKO OSTVARENJE ALGORITMA

Program je pisan u programskom jeziku C na operacijskom sustavu *Solaris 2.4* koji podržava višedretvenost. Program započinje unosom potrebnih parametara (*N, $D_{MAX}$, $\alpha$, S, DML, BROJ_NITI*), generiranjem zadanog broja gradova te računanjem matrice međusobnih udaljenosti. Slijedi računanje prosječnog povećanja duljine puta za 100 iteracija, iz kojeg slijedi početna vrijednost parametra *c*. Prvi *put* je određen samim generiranjem gradova te se za njega određuje duljina. Prije kreiranja podređenih dretvi potrebno je još incijalizirati varijable međusobne komunikacije, tj. sinhronizacije, te varijable za zaključavanje pristupa 'osjetljivim podacima'. Za sinhronizaciju se koriste uvjetne varijable, koje se koriste prilikom sistemskih poziva *cond_wait*, *cond_signal* i *cond_broadcast* kojima prethodi zaključavanje sa *mutex_lock*. Funkcija *cond_wait* postavlja pozivajuću dretvu u stanje čekanja, kojoj pozivi *cond_signal* ili *cond_broadcast* omogućavaju nastavak. Nakon inicijalizacije, sistemskim pozivima *thr_create* stvaraju se nove dretve, uz postojeću. One počinju izvršavati programski kod određen funkcijom *dretva* navedene u pozivu *thr_create*. Dretva najprije uzima svoj broj (*ID*), signalizira spremnost za računanje i čeka na signal '*start*'. Slijedi računanje iteracije *iter+ID*, odluka o prihvaćanju prijelaza te signalizacija kraja računa. Glavna dretva također računa svoju iteraciju te po završetku iste, pričekavši da sve dretve signaliziraju kraj računanja vrši odgovarajuću promjenu puta. Radi ubrzanja, glavna dretva ne računa ponovno gradove koji su bitni za zamijenu koja se prihvaća već ih čita iz globalnih (zajedničkih) varijabli *a* i *b* (polja), i to uzima one elemente čiji je indeks jednak broju dretve čiji se prijelaz obavlja. Takodjer, ne zamijenjuju se gradovi fizički već samo njihovi pokazivači. Posebno je važan problem sinhronizacije, odnosno zaključavanje i korištenje varijabli sinkronizacije te će u nastavku biti prikazani ovi dijelovi programa.

Glavna dretva:

```
mutex_lock(&kljuc);
for(n=0;n<S;n++){ /* iteracije hladjenja */
     for(m=0;m<M;){ /*iter.dostizanja term.ravnoteze*/
          gotovi=1;
          cond_broadcast(&start);/* kreni!! */
          mutex_unlock(&kljuc);

          /*generiranje susjedne konfiguracije,racunanje
          promjene duljine puta i odluka o prihvacanju*/
          /* ceka da sve dretve zavrse racunanje*/
          mutex_lock(&kljuc);
          while(gotovi<br_niti)
               cond_wait(&kraj,&kljuc);

          /* promjena */
     }
     T*=ALPHA; /* smanjenje temperature */
}
```

Podređena dretva:

```
mutex_lock(&kljuc);
```

```
for(;;){
     cond_wait(&start,&kljuc); /* cekaj start */
     mutex_unlock(&kljuc);

     /*generiranje susjedne konfiguracije, racunanje
     promjene duljine puta i odluka o prihvacanju */

     mutex_lock(&kljuc);
     gotovi++;
     cond_signal(&kraj);
}
```

Iz prikazanog je vidljivo da je samo postupak računanja prijelaza, povećanje duljine puta i odluke, izvan zaključanog dijela programa, što znači da kada se neka dretva nalazi u tom dijelu, a ne čeka na pozivu *cond_wait*, koji dok čeka otključava prolaz, sve ostale ili čekaju na zaključavanje ili računaju. Time je izbjegnuta pojava paralelnog korištenja sinhronizacijskih varijabli, što bi moglo rezultirati totalnim zastojem. Glavna dretva pričeka da sve podređene niti čekaju start, postavlja varijablu *gotovi* na 1 (ne treba kad završi račun povećavati) te označava start i odključava. Podređene dretve tada kreću u račun, a po završetku zaključavaju prolaz ostalima, povećavaju vrijednost gotovi za 1 te signaliziraju kraj računanja. Glavna dretva, nakon završetka računanja, provjerava da li su sve podređene dretve završile pregledavanjem varijable *gotovi*. Ako nisu, onda čeka na signale kraja sve dok sve ne završe.

# 7. REZULTATI IZVOĐENJA PROGRAMA

Program je izveden za probleme od 20, 50, 100, 200 i 500 gradova, uz upotrebu 1, 2, 4 i 6 dretvi. Ostali parametri su većinom isti, osim broja koraka hlađenja koji je za manje probleme manji te konstante *DML* koja je manja za veće probleme. Ulazni parametri su prikazani u tablici 1.

| ***N*** | ***S*** | $\alpha$ | ***DML*** | $P_0$ |
|---|---|---|---|---|
| 20 | 40 | 0.85 | 0.3333 | 0.8 |
| 50 | 70 | 0.93 | 0.3333 | 0.8 |
| 100 | 70 | 0.9 | 0.3333 | 0.8 |
| 200 | 70 | 0.7 | 0.3333 | 0.8 |
| 500 | 70 | 0.5 | 0.1 | 0.8 |

*N* - ukupan broj gradova
*S* - broj koraka hlađenja
$\alpha$ - faktor smanjenja temperature
$P_0$ - vjerojatnost prihvaćanja prve lošije konfiguracije
*DML* - faktor za određivanje broja iteracija unutarnje petlje.

**Tablica 1 Ulazni parametri**

Ulazni parametri određeni su tako da kvaliteta rezultata bude prihvatljiva, a da vrijeme računanja ne postane preveliko.

Nakon izvođenja programa uz navedene parametre dobio sam rezultate koji su tabelirani u tablicama 2, 3, 4. U tablici 2 nalaze se oni podaci koji ne ovise o broju dretvi, već samo o problemu. Broj dretvi utječe samo na vrijeme izvođenja.

| N | $d_0$ | $d_f$ | #iter | #prijelaza | $d_{min}$ |
|---|---|---|---|---|---|
| 20 | 12040 | 3993 | 5188 | 1092 | 3993 |
| 50 | 27097 | 6166 | 58310 | 16545 | 6166 |
| 100 | 51776 | 8242 | 233310 | 47616 | 8166 |
| 200 | 101060 | 11731 | 293305 | 53274 | 11231 |
| 500 | 254940 | 17925 | 875000 | 70840 | 17576 |

$d_0$ - početna duljina puta
$d_f$ - konačna duljina puta
#iter - ukupan broj iteracija
#prijelaza - ukupan broj prijelaza
$d_{min}$ - najkraći postignuti put.

**Tablica 2 Izlazni rezultati**

Program je izveden na jednoprocesrorskom sustavu SPARCstation-10 pod UNIX operacijskim sutavom. Vrijeme je mjereno funkcijom *clock(),* koja zbraja potrošeno korisničko vrijeme te potrošeno vrijeme sustava. Tako je vrijeme ts navedeno u tablicama upravo to vrijeme koje daje funkcija *clock()*. Vrijeme računanja i vrijeme promjena računano je tako da je upotrijebljena samo jedna dretva (glavna) te su sve funkcije sinhronizacije isključene (nepotrebne su). Tako je dobiveno vrijeme koje je jednako sumi ukupnog vremena računanja i ukupnog vremena izmjena.

Potom je isključen i dio koji mijenja konfiguracije i mjereno je samo vrijeme računanja. Iz razlike prethodna dva slijedi vrijeme potrebno za sve promjene. Podijelivši vrijeme promjene sa brojem promjena dobiva se prosječno vrijeme jedne promjene. Isto se napravi i sa vremenom računanja. Ove vrijednosti su prikazane u tablici 4 i na slici 1. Iz njih je vidljivo da prosječno vrijeme promjene raste sa porastom problema, što je i normalno jer se zamjena mora obaviti između prosječno većeg broja gradova. Prosječno vrijeme računanja jedne iteracije je međutim jako malo (ili čak niti nije) ovisno o veličini problema, pa je razlika koja se pojavlja vjerojatno nepreciznost mjerenja.

| #dretvi | #razina | $t_R$/#dretvi | $t_R$ | $t_P$ | $t_S$ | $t_K$ | $t_{OPT}$ |
|---|---|---|---|---|---|---|---|
| N=20 | | | | | | | |
| 1 | 5188 | 0.0933 | 0.0933 | 0.0079 | 0.12 | 0.0182 | 0.101 |
| 2 | 3003 | 0.0540 | 0.1080 | | 0.37 | 0.254 | 0.062 |
| 4 | 1965 | 0.0353 | 0.1422 | | 0.46 | 0.310 | 0.043 |
| 6 | 1641 | 0.0295 | 0.1770 | | 0.63 | 0.445 | 0.037 |
| N=50 | | | | | | | |
| 1 | 58310 | 1.048 | 1.048 | 0.393 | 1.51 | 0.068 | 1.441 |
| 2 | 35580 | 0.638 | 1.276 | | 4.46 | 2.80 | 1.033 |
| 4 | 24953 | 0.449 | 1.796 | | 5.9 | 3.71 | 0.842 |
| 6 | 21649 | 0.389 | 2.334 | | 8.16 | 5.43 | 0.782 |
| N=100 | | | | | | | |
| 1 | 233310 | 4.195 | 4.195 | 1.725 | 6.56 | 0.64 | 5.92 |
| 2 | 135462 | 2.436 | 4.872 | | 17.67 | 11.07 | 4.16 |
| 4 | 88403 | 1.589 | 6.356 | | 21.58 | 13.50 | 3.31 |
| 6 | 73483 | 1.321 | 7.926 | | 29.00 | 19.35 | 3.05 |
| N=200 | | | | | | | |
| 1 | 293305 | 5.86 | 5.86 | 2.93 | 9.57 | 0.87 | 8.79 |
| 2 | 167333 | 3.347 | 6.69 | | 23.77 | 14.15 | 6.28 |
| 4 | 106586 | 2.13 | 8.53 | | 28.27 | 16.81 | 5.06 |
| 6 | 87269 | 1.75 | 10.47 | | 36.13 | 22.73 | 4.68 |
| N=500 | | | | | | | |
| 1 | 875000 | 18.735 | 18.735 | 13.625 | 34.61 | 2.25 | 24.94 |
| 2 | 466208 | 9.977 | 19.954 | | 77.36 | 43.81 | 23.60 |
| 4 | 264440 | 5.685 | 22.740 | | 81.97 | 45.60 | 19.31 |
| 6 | 198110 | 4.259 | 25.554 | | 96.67 | 56.49 | 17.88 |

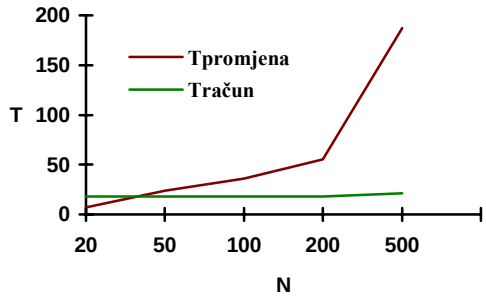
**Tablica 3 Vremena izvođenja programa**

Legenda:
$t_R$ - ukupno vrijeme računanja
$t_P$ - ukupno vrijeme promjena
$t_S$ - vrijeme koje javlja sustav (*clock()*)
$t_K$ - razlika između vremena sustava $t_S$ i vremena koje se dobije kao zbroj vremena računanja svih dretvi i vremena promjena konfiguracija
$t_{OPT}$ - najkraće moguće vrijeme izvođenja (na sustavu koji bi imao dovoljan broj procesora ) umanjeno za vrijeme komunikacija.

Rezultati prikazani u tablici 3 pokazuju da je najbrži onaj program sa samo jednom niti. To je ovdje i logično pošto je sustav jednoprocesorski. To međutim nije jedini razlog.

| N | $T_{promjena}$ µs | $T_{račun}$ µs |
|---|---|---|
| 20 | 7.3 | 17.98 |
| 50 | 23.78 | 17.98 |
| 100 | 36.23 | 17.98 |
| 200 | 55.32 | 19.98 |
| 500 | 187.00 | 21.4 |

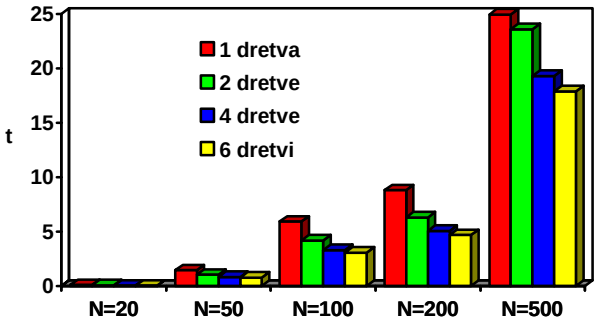
**Tablica 4 Prosječna vremena**



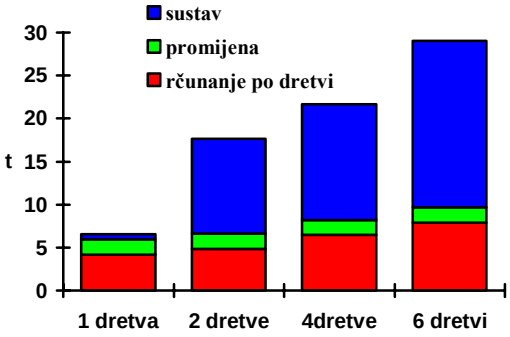

**Slika 1 Prosječna vremena**

Ako pogledamo koliko se vremena troši na računanje i promjene vidimo da je to manje od 50% ukupnog vremena. To znači da se veći dio vremena troši na sinhronizaciju, ali i ne samo nju. Pogledom na vrijeme koje troši sustav pri samo jednoj dretvi vidimo da je on prosječno ispod 6% ukupnog, ali je ipak značajan. Objašnjenje leži u tome što je to višekorisnički sustav sa podijelom vremena te on nakon određenog vremena rada sa jednim procesom sprema trenutno stanje procesa i učitava drugi. To je vrijeme jako kratko, ali kada se zbroji mnogo takvih promjena ono postane zamijetno.

Upravo je to slučaj kod više dretvi gdje sustav još mora mijenjati rad među dretvama. Svaka dretva nakon jedne iteracije računanja prepušta sustav drugoj. Na svakoj razini računanja tako imamo barem onoliko promjena konteksta koliko imamo dretvi. Tako npr. pri problemu 200 gradova i 4 dretvi izvrši se najmanje 106586*4=426344 promjena konteksta. Kad bi broj procesora bio veći onda bi taj broj bio sa istim faktorom manji.




**Slika 2 Trajanje izvođenja u idealnom slučaju**




**Slika 3 Raspodjela vremena**

U tablici 3 je sa $t_{OPT}$ označeno vrijeme dobiveno zbrajanjem vremena računanja po jednoj dretvi i vremena promjena. Kad bi se dretve paralelno izvršavale, te kada komunikacija ne bi bila značajno sporija, onda bi to zapravo bilo vrijeme izvršavanja programa. Iz slike 2 vidljivo je skraćenje vremena koje bi se tada postiglo.

Na slici 3 vidljiva je raspodijela vremena na pojedine dijelove programa.

# 8. ZAKLJUČAK

Metoda sinhronog N-paralelnog simuliranog kaljenja korištenjem višedretvenosti je perspektivna metoda ukoliko se ona provodi na višeprocesorskom sustavu sa zajedničkom memorijom i ukoliko je vrijeme samog računanja jedne nove susjedne konfiguracije dovoljno veliko da vrijeme komunikacije ne postane dominantno. Taj se zaključak može izvesti iz dobivenih rezultata. Razultati, naime, pokazuju da prilikom korištenja višedretvenosti velika većina vremena se troši na samom sustavu. U to vrijeme se uključuje vrijeme komunikacije među dretvama, ali i neizbježno vrijeme promjena konteksta pošto je korišteni jednoprocesorski višekorisnički sustav. Kada bi vrijeme računanja bilo mnogo veće, onda bi se ta vremena mogla zanemariti.

Ubrzanje bi se povećalo i smanjivanjem početne vrijednosti temperature. Što je, naime vrijednost temperature manja to je veća vjerojatnost odbacivanja nove konfiguracije. Tako bi navedena metoda koja koristi samo ODBACI grane binarnog stabla postigla najbolje rezultate, s obzirom na vrijeme izvođenja. Sa stanovišta kvalitete rezultata ta se početna temperatura ipak nebi smjela previše smanjiti.

Također, treba uzeti u obzir vrijeme potrebno za obavljanje prijelaza u novu konfiguraciju. Iz rezultata programa je vidljivo da povećavanjem broja gradova vrijeme potrebno za promjenu značajno raste, smanjujući tako ubrzanje paralelnog rada. Eventualnom promjenom metode odabira susjedne konfiguracije, čija bi promjena kraće trajala, postigli bi se bolji rezultati. Npr. metodom zamjene samo dva grada čija promjena je kratka i neovisna o veličini problema.

Metoda je dakle primjenjiva, ali samo za određene probleme.

## 9. Literatura


1. **A. Sohn,** "Parallel N-ary Speculative Computation of Simulated Annealing", *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, pp. 997-1005,Oct. 1995**.**
2. **P.J.M. van Laarhoven, E.H.L. Aarts**, "Simulated annealing: Theory and Applications", *D.Reidel Publishing Company*, Dordrecht, 1987.
3. "Solaris 2.4: Multithreaded Programing Guide", *SunSoft, Sun Microsystems*, Mountain View, California, 1994.

## 10. Prilog: Ispis programa

```
/* Program za rjesavanje problema trg.putnika (TSP) */
/* Metoda: simulirano kaljenje, spekulativno paralelno */
/* Platforma: visedretveni Solaris 2.4 UNIX sustav */
#include <stdio.h>
#include <signal.h>
#include <math.h>
#include <thread.h>
#include <string.h>
#include <time.h>
#define TRUE 1
#define FALSE 0

/* globalne varijable (zajednicka memorija svih niti)  */
short *g,br_niti,N,*d;
/*    g - polje indeksa gradova
      br_niti - broj niti koje simultano racunaju
      N - ukupan broj gradova
      d - matrica medjusobnih udaljenosti gradova */
short *a,*b,*zast,broj,gotovi;
/*    a,b - polja ciji i-ti elementi predstavljaju gradove izmedju
      kojih je i-ta nit pokusala zamijeniti redoslijed prolaza
      zast - polje zastavica odluka o prihvacanju promjena
      broj - pomocna varijabla pri identifikaciji niti
      gotovi - pomocna variabla sinkronizacije    */
unsigned int iter;
/*    iter - brojac tekuce iteracije   */
mutex_t kljuc;
cond_t start,kraj;
/*    kljuc - variabla preko koje se zakljucava pristup zajedn.mem
      start - glavna nit preko nje oznacava pocetak nove iteracije
      kraj - podredjene niti javljaju zavrsetak racuna */
double T;
long d_put,*raz;
/*    T - ekvivalent temperature kaljenja
      d_put - ukupna duljina puta
      raz - polje (indeks ID niti) promjena duljina puta prilikom
      neke promjene konfiguracije*/

void *nit(void *x);
```

```
/* procedura glavne niti */
void main(void){
      long pros,M,brojac=1,brpr=0;
      unsigned int sjeme;
      short n,m,Dmax;
      short pom,S;
      double ALPHA,DML,Po;
      char argv[80];
/*    pros - prosjecno pov.f. cilja
      brojac - brojac razina racunanja
      n,m,pom,argv - pomocne variable       */

      struct koor{
            short x;
            short y;
      } *k;

      printf("\n\nProgram za optimizaciju problema trgovackog
putnika!!!\n");
      printf("-------------------------------------------------\n");
      printf("Unesi nove ili potvrdi zadane vrijednosti parametara");
      do{
            N=100;
            printf("Ukupan broj gradova (%d): ",N);
            gets(argv);
            if(argv[0]>48)
                  N=atoi(argv);
      }while(N<5);
      do{
            Dmax=1000;
            printf("Maksimalna koordinata (cjelobr.) (%d): ",Dmax);
            gets(argv);
            if(argv[0]>48)
                  Dmax=atoi(argv);
      }while(Dmax<5);
      do{
            Po=0.8;
            printf("Vjerojatnost prihv. promj. konf Po (%g): ",Po);
            gets(argv);
            if(argv[0]>47)
```

```
          Po=atof(argv);
}while(Po<0.000001 || Po>0.99999);
do{
     S=50;
     printf("Broj koraka hladjenja S (%d): ",S);
     gets(argv);
     if(argv[0]>48)
          S=atoi(argv);
}while(S<1);
do{
     ALPHA=0.9;
     printf("Faktor promjene vrijednosti temp. (%g): ",ALPHA);
     gets(argv);
     if(argv[0]>47)
          ALPHA=atof(argv);
}while(ALPHA<0 || ALPHA>1);
do{
     DML=0.3333;
     printf("Fak.za odr.br.promj.konf.pri istoj T(%g): ",DML);
     gets(argv);
     if(argv[0]>47)
          DML=atof(argv);
}while(DML<0 || DML>1);
do{
     br_niti=sysconf(_SC_NPROCESSORS_ONLN);
     printf("Broj niti (%d):  ",br_niti);
     gets(argv);
     if(argv[0]>47)
          br_niti=atoi(argv);
}while(S<1);
M=DML*N*N;
printf("\nN=%d,Dmax=%d,S=%d,ALPHA=%lf,DML=%lf,br_niti=%d,M=%d\n
     ",N,Dmax,S,
ALPHA,DML,br_niti,M);
/* zauzece memorije   */
g=(short*) malloc((N+2)*sizeof(short));
g+=1;
d=(short*) malloc(N*N*sizeof(short));
raz=(long*) malloc(br_niti*sizeof(long));
a=(short*) malloc(br_niti*sizeof(short));
b=(short*) malloc(br_niti*sizeof(short));
```

```
zast=(short*) malloc(br_niti*sizeof(short));
k=(struct koor*) malloc(N*sizeof(struct koor));

if(g==NULL || d==NULL || k==NULL || a==NULL || b==NULL ||
zast==NULL || raz==NULL){
      printf("Nema dovoljno memorije!\n");
      exit(1);
}
/* inicijalizacija*/
k[0].x=0;
k[0].y=0;
g[0]=0;
iter=1;
for(n=1;n<N;n++){
      g[n]=n;
      k[n].x=rand_r(&iter)%Dmax;
      k[n].y=rand_r(&iter)%Dmax;
}
g[-1]=g[N-1];
g[N]=g[0];
for(n=0;n<N;n++){
      for(m=0;m<N;m++)
            d[n*N+m]=(long)sqrt((k[n].x-k[m].x)*(k[n].x-
            k[m].x)+(k[n].y-k[m].y)*(k[n].y-k[m].y));
}
d_put=0;
for(pom=0;pom<N;pom++)
      d_put+=d[g[pom-1]*N+g[pom]];

printf("\nPocetni put= %d\n",d_put);

pros=0;
m=0;
for(n=0;n<100;n++){ /* racunanj pros.pov.duljine puta */
      int p,q;
      p=rand_r(&iter)%N;
      q=rand_r(&iter)%N;
      if((q>p&&(q-p<3||q-p>N-3))||
            (p>q&&(p-q<3||p-q>N-3))||p==q)
            continue;
```

```
      raz[0]=d[g[p]*N+g[q-1]] + d[g[q]*N+g[p+1]]
      -d[g[q]*N+g[q-1]] - d[g[p]*N+g[p+1]];

      if(raz[0]>=0){
            m++;
            pros+=raz[0];
      }
}

T=pros/m*1.;
printf("Prosjecno povecanje puta: %g\n",T);
T/=log(1/Po);
if(T==0) T=1;
printf("Pocetna temperatura: %g\n",T);

mutex_init(&kljuc,USYNC_THREAD,NULL);
cond_init(&start,USYNC_THREAD,NULL);
cond_init(&kraj,USYNC_THREAD,NULL);

broj=1;
iter=1;

/* kreiranje podredjenih niti */
for(n=1;n<br_niti;n++)
      if(thr_create(NULL,0,nit,NULL,0,NULL)){
            printf("Greska u kreiranju thredova\n");
            exit(1);
      }

thr_setconcurrency(sysconf(_SC_NPROCESSORS_ONLN));
mutex_lock(&kljuc);
while(broj<br_niti)
      cond_wait(&kraj,&kljuc);

/* racunanje */
printf("\nRacunam ");
clock();
for(n=0;n<S;n++){     /* iteracije hladjenja */
      int pr=0;  /* oznacivac promjene konf.*/
      for(m=0;m<M;){  /* iter. dostizanja term.ravnoteze    */
            short p,q;
```

```
unsigned int sjeme;
gotovi=1;/* "nitko jos nije zavrsio" */
cond_broadcast(&start);/* kreni!! */
mutex_unlock(&kljuc);

if(brojac%1000==0){   /*sminka*/
      printf(".");
      fflush(stdout);
}
/* generiranje susjedne konfiguracije (promjene
red izm. gr p i q */
sjeme=iter;
p=rand_r(&sjeme)%N;
q=rand_r(&sjeme)%N;

zast[0]=FALSE;
if(p>q){ int pom=p; p=q; q=pom; }

/* racunanje povecanja duljine puta; odluka o
prihvcanju nove konf. */
if(!(p==q || q-p>N-2)){
      raz[0]=d[g[p]*N+g[q+1]] + d[g[q]*N+g[p-1]]
      -d[g[q]*N+g[q+1]] - d[g[p]*N+g[p-1]];
      if(raz[0]<0 ||
      exp(-raz[0]/T)>rand_r(&sjeme)/32767.)
            zast[0]=TRUE;
}
a[0]=p;
b[0]=q;

/* ceka da sve niti zavrse racunanje  */
mutex_lock(&kljuc);
while(gotovi<br_niti)
      cond_wait(&kraj,&kljuc);

/* cija je konf. prihvatljiva uz najmanji broj
iteracije */
for(pom=0;pom<br_niti;pom++)
      if(zast[pom])
            break;
iter+=pom;
```

```
            m+=pom;
            brojac++;
            if(m>=M){
                  iter-=m-M;
                  break;
            }
            /* zamijena redoslijeda gradova (indeksa) u slucaju
            prihv. promjene */
            if(pom<br_niti){
                  short pom2,t;
                  m++;
                  iter++;
                  brpr++;
                  pr=1;
                  p=a[pom];
                  q=b[pom];
                  pom2=(q-p)/2+1;
                  for(t=0;t<pom2;t++){
                        short pom;
                        pom=g[p+t];
                        g[p+t]=g[q-t];
                        g[q-t]=pom;
                  }
                  g[-1]=g[N-1];
                  g[N]=g[0];
                  d_put+=raz[pom]; /* nova duljina puta */
            }
      }
      T*=ALPHA; /* smanjenje temperature */
      if(pr==0) break;
}
/* ispis rezultata */
Po=1.*clock()/CLOCKS_PER_SEC;
printf("\n\nKonacni put=%d\nTemperatura zamrzavanja=
%g\n",d_put,T);
printf("Ukupan broj razina racunanja = %d\n",brojac);
printf("Ukupan broj iteracija racunanja = %d\n",iter);
printf("Ukupan broj promjena puta = %d\n",brpr);
printf("Ukupno potroseno vrijeme sustava: %g [s]\n",Po);
printf("Zelis li ispis koordinata gradova (d/N): ");
gets(argv);
```

```
      if(argv[0]=='d'){
            printf("\n.\n%d\n",N);
            for(n=0;n<N;n++)
                  printf("%d %d\n",k[g[n]].x,k[g[n]].y);
      }
}
/* kraj procedure glavne niti */

/* procedura podredjene(ih) niti */
void *nit(void *x){
      int ja;    /* identifikacijski broj niti */
      short pom,p,q;
      unsigned int sjeme;

      /* dobivanje identifikacije */
      mutex_lock(&kljuc);
      ja=broj;
      broj++;
      cond_signal(&kraj);

      /* racunanje */
      for(;;){
            cond_wait(&start,&kljuc); /* cekaj start */

            mutex_unlock(&kljuc);

            /* generiranje susjedne konfiguracije (promjena redosl.
            izm. gr p i q */
            sjeme=iter+ja;
            p=rand_r(&sjeme)%N;
            q=rand_r(&sjeme)%N;

            zast[ja]=FALSE;
            if(p>q){ int pom=p; p=q; q=pom; }

            /* racunanje povecanja duljine puta; odluka o prihvcanju
            nove konf. */
            if(!(p==q || q-p>N-2)){
                  raz[ja]=d[g[p]*N+g[q+1]] + d[g[q]*N+g[p-1]]
                  -d[g[q]*N+g[q+1]] - d[g[p]*N+g[p-1]];
```

```
                if(raz[ja]<0 ||
                exp(-raz[ja]/T)>rand_r(&sjeme)/32767.)
                      zast[ja]=TRUE;
          }
          a[ja]=p;
          b[ja]=q;
          /* signalizacija kraja iteracije */
          mutex_lock(&kljuc);
          gotovi++;
          cond_signal(&kraj);
     }
}
/* kraj procedure niti */
```