Adaptivno simulirano kaljenje

Projekat u okviru kursa Računarska inteligencija

Matematički fakultet

Aleksandar Cvetković

Januar 2025

Sadržaj

1 Uvod

Kaljenje je toplotna obrada kojom se postiže otvrdnuće čelika. Sastoji se od zagrevanja čelika na određenu temperaturu, potom održavanja temperature te njeno kontrolisano smanjivanje na sobnu, pri čemu dolazi do poboljšanja osobina čelika. Ova metalurška tehnika je pozajmila ime optimizacionom algoritmu kojim ćemo se baviti u ovom radu.

Simulirano kaljenje je otkriveno u više navrata od strane različitih istraživača tokom sedamdesetih i osamdesetih godina. Ime su mu dali Kirkpatrik, Gelat i Veči [1].
Tokom godina dolazi do više modifikacija algoritma kojim se pokušavaju unaprediti njegova svojstva. Takva je i modifikacija Lestera Ingbera, prvobitno nazvana Veoma brzo simulirano kaljenje [2], potom Adaptivno simulirano kaljenje [3]. Kroz ovaj rad ćemo se ostvrtati na ova dva dela, kao i autorev algoritam napisan u C programskom jeziku [4] koji mi je pomogao u implementaciji algoritma.

2 Problem trgovačkog putnika

Kao popularan problem šezdesetih i sedamdesetih godina prošlog veka, Problem trgovačkog putnika bio je jedan od prvih koji će biti rešen tehnikom simuliranog kaljenja. Ovaj problem će i nama poslužiti kao pokazni, ali i problem na kome ćemo uporediti tehniku kojom se bavimo sa nekim drugim tehnikama.
Poreklo samog problema je nepoznato, a matematički ga je definisao Vilijem Hamilton u 19. veku. Skoro jedan vek matematičari i kasnije informatičari sa svih strana sveta pokušavaju da nađu rešenje za ovaj problem ili neku njegovu generalizaciju kao što je Problem rutiranja vozila. Problem trgovačkog putnika se može razmatrati u raznim oblastima kao što su logistika, planiranje, ali i pravljenje mikrčipova i DNA sekvenciranje.
Sam problem se formuliše kao pronalaženje najkraćeg puta koji trgovački putnik treba da pređe tako da obiđe svaki grad tačno jednom. U terminima teorije grafova, problem se može definisati kao pronalaženje puta najmanje težine u Hamiltonovom grafu. Hamiltonov graf je graf koji ima Hamiltonov ciklus, a Hamiltonov ciklus je put koji prolazi svaki čvor u grafu tačno jednom.
Ispitivanje da li je graf Hamiltonov je NP-kompletan problem[1] što implicira da je Problem trgovačkog putnika NP težak. To znači da ne postoji egzaktan algoritam koji bi rešio ovaj problem tako da rešenje u najgorem slučaju bude kraće od $n!$. Godinama su razvijanje tehnike kojima bismo mogli odrediti najkraće rute nekih većih Hamiltonovih grafova barem približno, u nekom razumnom vremenu. Takva je imeđu ostalog i tehnika simuliranog kaljenja. ona nam često neće dati tačan rezultat, ali će greška biti razumno mala u odnosu na vreme izvršavanja programa.
Postoje dve vreste problema trgovačkog putnika, simetrični i asimetrični. Kod simetričnog su udaljenosti između dva čvora iste u oba suprotna pravca, dok kod asimetričnog to ne mora da bude slučaj. Šta više, suprotna grana ne mora ni da postoji. Mi ćemo sa na dalje baviti simetričnom vrstom ovog problema što nije neko ograničenje, jer je asimetrični graf može konvertovati u simetrični. Takođe, obrađivaćemo podatke zadate pomoću koordinata. Zato ćemo ubuduće imesto pojma 'težina grane' između dva čvora, često koristiti termine kao što su 'udaljenost' ili 'razdaljina' između dva čvora. Iz istog razloga koristimo termin ruta za ciklus. Drugi način je predstavljanje podataka kao matrice gde svaki član M_{ij} predstavlja udaljenost između čvora i i j.

[1] Dokazao Ričard Karp 1972. god.

3 Simulirano kaljenje

3.1 Standardni algoritam

U Uvodu smo spomenuli algoritam Kirkpatricka. Njegov algoritam je uopštenje Metropolis Monte Carlo algoritma za integraciju(1953) sa dodatkom smanjivanja temperature. Kasnije će biti dokazano[2] da funkcija opadanja temperature ne treba da bude sporija od $1/log(k)$ gde k predstavlja trenutni broj izvršenih iteracija algoritma. Za funkciju opadanja temperature kasnije se koristi i $1/k$. Takav algoritam se zove Brzo kaljenje ili Brzo Košijevo kaljenje.
Danas se u raznim impementacijama u funkciji opadanja temperature često koristi izraz $T_{k+1} = T_k - \alpha * T_k$, gde je T_k temperatura u trenutku k, a α koeficijent opadanja temperature.

Sama ideja algoritma je jednostavna: vršimo pretragu po prostoru rešenja i uvek prihvatamo bolja rešenja dok lošija prihvatamo sa nekom verovatnoćom. Ta verovatnoca vremenom opada kako i temperatura opada.
Pogledajmo sada pseudokod standardne verzije algoritma.

Slika 1 Pseudokod standardnog algoritma

```
1:  procedure SimuliranoKaljenje
2:      MaxIters ← Maksimalni broj iteracija algoritma
3:      Ciklus ← Inicijalni ciklus
4:      NajboljiCiklus ← Ciklus
5:      while k < MaxIters do
6:          NoviCiklus ← GenerišiNoviCiklus(Ciklus)
7:          r ← Nasumični broj iz intervala (0, 1)
8:          if Dužina(NoviCiklus) < Dužina(Ciklus) then
9:              Ciklus ← NoviCiklus
10:             if Dužina(Ciklus) < Dužina(NajboljiCiklus) then
11:                 NajboljiCiklus ← Ciklus
12:         else if  r < 1 / k  then
13:             Ciklus ← NoviCiklus
14:         k ← k + 1
15:     return Ciklus
```

[2] Geman and Geman, 1984

U ovoj verziji algoritma, nigde se eksplicitno ne pojavljuje temperatura, nego nam izraz $\frac{1}{k}$ (gde k predstavlja iteraciju algoritma) predstavlja baš funkciju temperature, ali i verovatnoću odabira novog ciklusa. Kako je r broj iz intervala $(0, 1)$ a izraz $\frac{1}{k}$ inicijalno uzima vrednost 1 i vremenom sve sporije opada, tako će i algoritam sa sve manjom verovatnoćom prihvatati cikluse koji su duzi od trenutno najboljeg.
Prilikom ispitivanja da li je rezultat na koji smo naišli bolji od trenutnog ciklusa, trebamo ispitati i da li je taj rezultat bolji najboljeg ciklusa koji smo našli tokom čitavog izvršavanja algoritma. Takav ciklus moramo zapamtiti jer bismo u suprotnom izgubili informaciju o njemu, birajući lošiji ciklus u nekoj narednoj iteraciji algoritma. Na ovaj detalj vratićemo se nešto kasnije, kad budemo diskutovali o Adaptivnom simuliranom kaljenu.
Funkcija **GenerišiNoviCiklus(***Ciklus***)** kao argument prihvata ciklus i kao povratnu vrednost ima isti taj ciklus nad kome je izvršena promena. Promenu je moguće izvršiti na više načina kao što su umetanje nasumičnog elementa ili invertovanje podniza ciklusa između dva nasumična indeksa. Promena je u našem slučaju zamena mesta dva nasumična grada.
Funkcija **Dužina(***Ciklus***)** predstavlja meru kvaliteta datog ciklusa[3]. Kao i kod drugih algoritama, ne postoji pravilo koje određuje kako treba da izgleda ova funkcija. Kod nas će ona ona jednostavno sabrati dužine između svaka dva čvora u ciklusu.
Funkcija se izvršava dok ne brojač k ne postane jednak maksimalnom broju iteracija koji smo unapred odredili. Postoji mogućnost da se napiše posebna funkcija zaustavljanja koja može zavisiti od broja iteracija tokom koje nismo imali promenu *NajboljegCiklusa* ili se znak za prestanak izvršavanja može uneti preko tastature, ali zbog jednostavnosti biramo prvi način.

3.2 Bolcmanovo simulirano kaljenje

Bolcmanovo simulirano kaljenje je bitan trenutak u razvoju algoritma.
Do sad smo lošije cikluse birali potpuno nasumično, a zašto bismo to radili? Ako potpuno nasumično prelazimo na nove cikluse, možemo algoritam dovesti u stanje gde mu treba dosta vremena da se vrati na mesto u kojem se već nalazimo, ili u gorem slučaju odvesti pretragu u potpuno pogrešni pravac.
Umesto toga, prilikom biranja nasumičnog ciklusa, želimo da uzmemo u razmatranje koliko je to rešenje gore od trenutnog rešenja.
Ovo će nam omoguíti Bolcmanova (Gibsova) raspodela verovatnoća[4].Funkcija verovatnoće izbora novog ciklusa sada izgleda ovako.

$$h(\Delta E) = \frac{1}{1 + \exp \frac{\Delta E}{T}}$$
$$\approx \exp \frac{-\Delta E}{T}$$

[3]en. cost function, fitness function
[4]Ludwig Boltzmann, Josiah Willard Gibbs

Gde h predstavlja funkciju verovatnoće izbora novog ciklusa a ΔE prestavlja razliku između prethodne i trenutne energije, odnosno kod nas razliku dužina trenutnog i prethodnog ciklusa $\Delta E = E_k - E_{k-1}$. Simulirano kaljenje sa ovakvim izborom (pogrešnih) narednih stanja se često naziva Bolcmanovo kaljenje[5], ali se nekad navodi i kao osnovna verzija algoritma.
Teorijske osnove Bolcmanove raspodele izlaze iz okvira ovog rada, te se njome nećemo baviti.
Sada moramo eksplicitno uvesti 'hlađjenje' u naš algoritam. U svakom k iteraciji algoritma, temperatura iznosi $T = \frac{T0}{\ln k}$, gde je $T0$ početna temperatura.
Sada naš algoritram izgleda ovako.

Slika 2 Pseudokod unapređenog algoritma

```
1:  procedure BolcmanovoSimuliranoKaljenje
2:      T0 ← Početna temperatura
3:      MaxIters ← Maksimalni broj iteracija algoritma
4:      Ciklus ← Inicijalni ciklus
5:      NajboljiCiklus ← Ciklus
6:      while k < MaxIters do
7:          NoviCiklus ← GenerišiNoviCiklus(Ciklus)
8:          r ← Nasumični broj iz intervala (0, 1)
9:          if Dužina(NoviCiklus) < Dužina(Ciklus) then
10:             Ciklus ← NoviCiklus
11:             if Dužina(Ciklus) < Dužina(NajboljiCiklus) then
12:                 NajboljiCiklus ← Ciklus
13:         else if r < exp(−(Dužina(NoviCiklus) − Dužina(Ciklus))/T)
    then
14:             Ciklus ← NoviCiklus
15:         k ← k + 1
16:         T ← T0/ln(k)
17:     return Ciklus
```

4 Adaptivno simulirano kaljenje

U prethodnom poglavlju smo videli kako da u trenutcima kada biramo lošiji ciklus od trenutnog, to ne radimo potpuno nasumično nego uzmemo u obzir to koliko je ciklus lošiji od trenutnog.
Simlulirano kaljenje i dalje ostaje dosta jednostavan algoritam što nije loše ako nam je brzo potrebno nekakvo rešenje, ali nam neće puno pomoći ako naiđemo na težak problem. Problem, u našem slučaju, brzo postaje težak sa povećanjem dimenzije problema. Sa različitom dimenzijom problema, potrebno je naći odgovarajuću temperaturu koja će dovesti do globalnog minimuma, kao i funkciju hlađenja. Adaptivno simulirano kaljenje pokušava da u ovom smislu poboljša

[5] Szu and Hartley(1987)

algoritam, kao i da olakša njegovu primenu na različite probleme, kao i različite dimenzije istog problema.

Sa porastom broja dimenzija (parametara) rešenjam, rizikujemo da neke dimenzije potpuno zapostavimo ili da druge previše često razmatramo. Da bismo predupredili ova loša ponašanja algoritma, pojam temperature sada posmatramo nešto drugačije.
Neka svaka dimenzija ima svoju temperaturu u k-toj iteraciji našeg algoritma $T_i(k)$. Svaka temperatura T_i ima svoju funkciju hlađenja, ona se izvršava svaki put kad izvršimo promenu u datoj dimenziji, odnosno kad generišemo novo rešenje.
U našem slučaju, neka svi gradovi imaju istu temperaturu na početku izvršavanja algoritma. Nakon zamene mesta dva grada, za vrednosti njihovih temperatura pozivamo funkciju hlađenja, te temperature dobijaju nove vrednosti. Što je manja vrednost temperature, to je manja verovatnoća da će u nekoj narednoj iteraciji algoritma ovaj grad biti izabran za zamenu.
Slično, imamo i 'glavnu' temperaturu, onu koja nam govori dal ćemo lošije rešenje prihvatiti ili odbaciti. Funkciju hlađenja za ovu temperaturu izvršavamo u svakoj iteraciji algoritma u kome smo prihvatili novo rešenje. Inberg navodi da se dobro ponašanje algoritma može očekivati ako odaberemo istu funkciju hlađenja za sve temperature, i nju definišemo na sledeći način.

$$T_i(k) = T_{0i} \exp\left(-c_i k_i^{\frac{1}{D}}\right)$$

Gde je k_i broj izmena parametra i. U našem slučaju, broj ciklusa koji su nastali kao zamena čvora na i-tom indeksu ciklusa. T_{0i} je početna temperatura za datu dimenziju, D broj dimenzija rešenja problema, a c je parametar kojim možemo preciznije kontrolisati rad algorima.

$$c_i = m_i \exp \frac{-n_i}{D}$$

gde su m_i i n_i slobodni parametri koji nam služe da poboljšamo performanse algoritma prilikom primene na konkretan problem.

Slično definišemo i funkciju hlađenja za glavnu temperaturu,

$$T(k) = T_0 \exp\left(-ck^{\frac{1}{D}}\right).$$

Gde je k broj prihvaćenih rešenja, bila ona bolja ili gora od trenutnog. Parametar kontrole algoritma c definišemo na praktično isti način, pomoću slobodnih parametara m i n,

$$c = m \exp \frac{-n}{D}$$

4.1 Ponovno kaljenje

Kako algoritam doseže dovoljno veliki broj iteracija, vrednosti temperatura se smanjuju. To najviše efekta ima na glavnu temperaturu što će nas sprečavati da 'pogledamo' nova rešenja, i time potencijalno ostaviti lokalni minimum kao najbolje rešenje. S toga je potrebno, u zavisnosti od nekog kriterijuma, restartovati[6] algoritam, odnosno vrednosti temperatura postaviti na početnu, te anulirati neke brojače. Ovim će se algoritam izbaciti iz lokalnog minimuma ako se u njemu našao, a ako nije onda svakako imamo zapamćeno najbolje rešenje.
Problemi kojima se bave optimizacioni algoritmi, pa i naš problem trgovačkog putnika, su takvi da promena na različitim dimenzijama problema ne dovodi do identičnih promena kvaliteta rešenja problema. Ako sve temperature samo postavimo na početnu, gubimo informaciju o tome koliko je koji parametar osetljiv na promene, odnosno prilikom promene na kojoj dimenziji imamo najveću šansu da dobijemo kvalitetnije rešenje problema.
U slučaju našeg problema, promene u redosledu obilaska gradova koji se nalaze dovoljno blizu jedan drugom neće bitnije uticati u ukupnoj dužini algoritma, te ovakve promene ne treba često razmatrati. Sa druge strane, gradovi koji su nemaju bliske susede, imaju veću šansu da smanje dužinu ciklusa.
Razlike u osetljivosti na promene parametara zato želimo da zapamtimo, i to radimo prilikom poslednjeg prihvatanja rešenja, nastalog kao posledica promene datog parametra. U našem slučaju, dovoljno je zapamtiti koliku je razliku u dužini rute napravila promena grada na određenom na mestu gde je izvršena.
Dakle, prilikom svakog restartovanja algoritma, temperature reskaliramo na sledeći način,

$$k_i \to k_i',$$
$$T'_{ik}{}' = T_{ik}\frac{s_{max}}{s_i},$$
$$k_i' = \left(\frac{\ln \frac{T_{i0}}{T_{ik}{}'}}{c}\right)^D.$$

Sve temperature, uključujući i glavnu, pri pokretanju algoritma postavljamo na vrednost dužine inicijalnog ciklusa. Glavnu temperaturu reskaliramo na sličan način. S obzirom da je prvobitna temperatura postavljena na vrednost dužine inicijalnog ciklusa, ona se bitno razlikuje od vrednosti trenutnog minimuma. Zato ovu temperaturu postavljamo na vrednost poslednjeg minimuma dužina ciklusa. Parametar k u ovom slučaju postavljamo tako da temperatura koja će biti dobijena prilikom prvog hlađenja bude jednaka bude jednaka dužini najboljeg ciklusa.

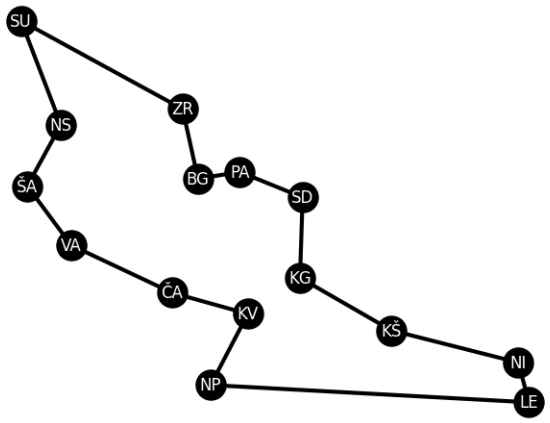
Prilikom svakog restartovanja algoritma potrebno je najbolji ciklus označiti kao trenutni koji obrađujemo. Samo restartovanje, povećanjem temperatura, omogućava algoritmu da sagleda mnogo veći skup rešenja u prvim iteracijama nakon

[6] en: Reannealing

restartovanja. Ako bismo za trenutno rešenje ostavili potencijalno ne-najbolje rešenje, rizikujemo da prilikom pokretanja algoritma sa novim temeraturama prihvatimo rešenje koje je previše loše, i time poništimo napredak koji smo ostvarili.

5 Rezultati

Algoritam prvo testiramo na podacima sa malim brojem dimenzija. Koristimo *rs5.tsp*, *rs10.tsp* i *rs15.tsp*, datoteke sa podacima o redom 5, 10 i 15 najvećih gradova u Srbiji[7]. Algoritmi su testirani tako što je prvo izmereno vreme potrebno algoritmu Grube sile da izračuna najkraću rutu, a potom je taj podatak iskorišcen kao kriterijum zaustavljanja u ostalim algoritmima i tom prilikom je izmereno vreme zaustavljanja svakog algoritma.



Slika 3: Najbolji ciklus za prvih 15 najvećih gradova u Srbiji

Kako je za 10 gradova bilo potrebno 21 sekundi da se ruta nađe iscrpnom pretragom, lako se može izračunati da bi za 15 gradova bilo potrebno skoro 3 meseca. Zato rešenje dobijamo pretragom naprednijim algoritmima Simuliranog kaljenja, a možemo ga i vizuelno verifikovati.
Parametar temperature za sve algoritme postavljen na vrednost dužine najboljeg ciklusa. Autor, prilikom implementacije[8], definiše parametre m i n na sledeći način,

$$m_i = -\log(TemperatureRatioScale)$$

,

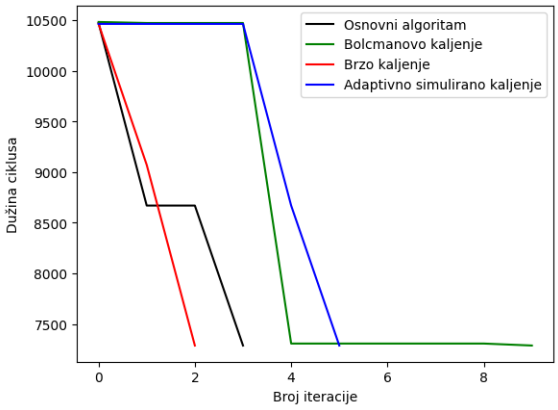
$$n_i = \log(TemperatureAnnealScale)$$

[7] Informacije o geografskom Centru grada u preuzete sa Vikipedije.
[8] ASA-README

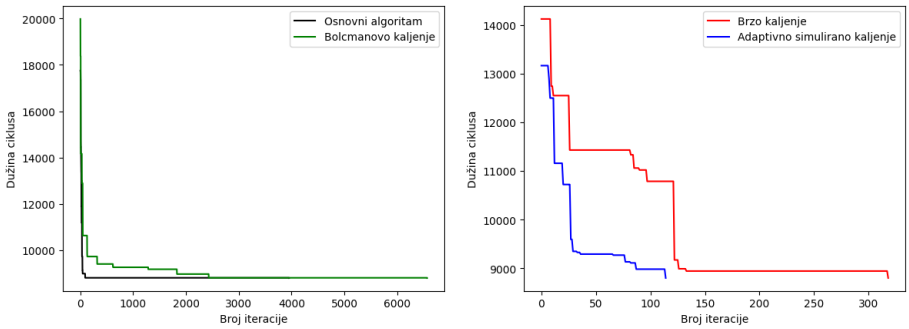
, gde su podrazumevane vrednosti za TemmperatureRatioScale $1.0 * 10^{-5}$ a za TemperatureAnnealScale 100.0. Sa tim vrednostima i mi testiramo algoritam.

Naziv algoritma	rs5.tsp	rs10.tsp	rs15.tsp
Gruba sila	$1.703*10^{-4}$	21.927	$7.567 * 10^{6**}$
Osnovni algoritam	$1.734*10^{-4}$	0.084	-
Bolcmanovo kaljenje	$3.08*10^{-4}$	0.152	23.934
Brzo kaljenje	$2.535*10^{-4}$	0.01	0.016
Adaptivno simulirano kaljenje	$3.366*10^{-4}$	0.003	0.021

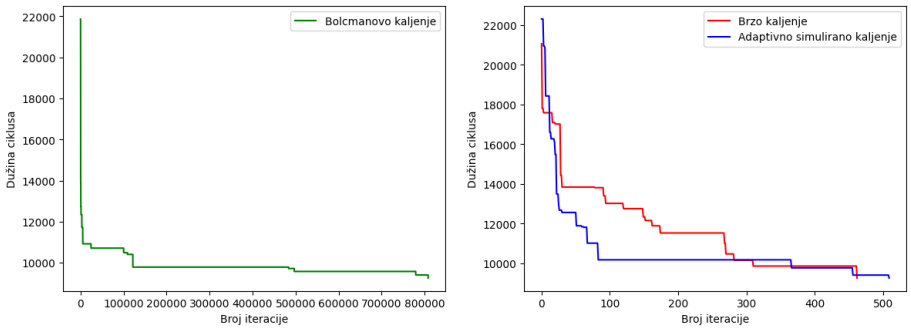
Tabela 1: Vreme zaustavljanja algoritama izraženo u sekundama.
**procenjeno vreme zaustavljanja(oko 87 dana)



Slika 4: Grafički prikaz pronalaženja minimuma na skupu rs5



Slika 5: Grafički prikaz pronalaženja minimuma na skupu rs10



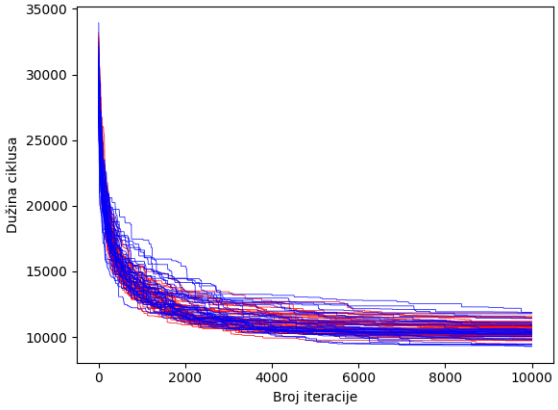
Slika 6: Grafički prikaz pronalaženja minimuma na skupu rs15

Primetimo da algoritmi mogu imati sreće u brzom pronalaženju rešenja, a mogu i upasti u lokalni minimum iz koga se ne mogu izbaviti. Osnovni algoritam je u većini slučajeva završio izvršavanje bez pronalaženja globalnog minimuma, u slučaju sa 15 gradova, dok je svaki naredni ostvarivao nešto bolji rezultat. Kako su druga dva algotirma uspešnija na malim skupovima podataka, njih testiramo na većem skupu podataka *berlin52.tsp*[9] i *qa194.tsp* [10].Brzo kaljenje testiramo sa vrednostima temperature od 1 do $2 * GlobalniMinimum$. Adaptivno simulirano kaljenje sa parametrima,

$$TemperatureRatioScale \times TemperatureAnnealScale \times ReannealRate$$

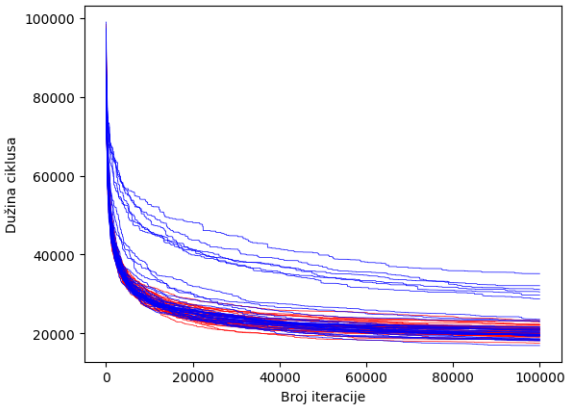
$$= [0.00001, 0.0001, 0.001, 0.01] \times [10, 100, 1000] \times [10, 30, 50],$$

gde je ReannealRate broj prihvaćenih ciklusa, nakon kojeg radimo resetovanje algoritma.



Slika 7: Grafički prikaz pronalaženja minimuma na skupu *berlin52*

[9]Preuzeto sa https://www.kaggle.com
[10]Preuzeto sa https://www.math.uwaterloo.ca/tsp/world/countries.html



Slika 8: Grafički prikaz pronalaženja minimuma na skupu *qa194*

6 Zaključak

Prvobitni cilj algoritma Adaptivno simulirano kaljenje je bio elimisanje unošenja parametra temperature. Kako samo ime kaže, algoritam treba i da se adaptira za različite tipove i dimenzije problema.
Primetimo da algoritmi mogu imati sreće u brzom pronalaženju rešenja, a mogu i upasti u lokalni minimum iz koga se ne mogu izbaviti. Osnovni algoritam je u većini slučajeva završio izvršavanje bez pronalaženja globalnog minimuma, u slučaju sa 15 gradova, dok je svaki naredni ostvarivao nešto bolji rezultat. Adaptivno simulirano kaljenje je ostvarilo svoj cilj globalne konvergencije bez unošenja parametara, i to je radilo u većini slučajeva bolje od standardnog algoritma.
Kako raste broj gradova, prednost našeg algoritma postaje sve izraženija. To se može videti na slikama 7, 8 i 9. Brzo kaljenje i ako brže konvergira od Adaptivnog, ima malu moguńost da se izbavi iz lokalnog minimuma. Adaptivno simulirano kaljenje i sa lošim izborom parametara, u većini slučajeva, ponaša se isto kao i Brzo kaljenje. Sa dobrim izborom parametara sigurno daje bolji rezultat od Brzog kaljenja, posle dovoljnog broja iteracija.

Literatura

[1] S. Kirkpatrick; C. D. Gelatt; M. P. Vecchi (1983) *Optimization by Simulated Annealing*, Science, New Series, Vol. 220, No. 4598.

[2] Lester Ingber (1989) *Very Fast Simulated Re-Annealing*, Mathl. Comput. Modelling.

[3] Lester Ingber (1995) *Adaptive simulated annealing (ASA): Lessons learned*, Control and Cybernetics.

[4] Lester Ingber (1993) *ASA-README, ASA-NOTES*, Adaptive Simulated Annealing (ASA).