Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

Seminar

Identifikacija parametara prijelazne toplinske impedancije poluvodiča metodom optimizacije.

Autor
Dana Dodigović

Voditelj
dr. sc. Marinko Kovačić

10. svibnja 2018.

Sadržaj

1 Uvod

Elektronički uređaji se tipično sastoje od vrlo velikog broja tranzistora u kojima tijekom rada dolazi do gubitaka. Ti gubitci predstavljaju pretvorbu električne u toplinsku energiju. U mnogo slučajeva nije dovoljno da toplina odlazi u okolinu jer se iz uređaja oslobađa prevelika snaga i temperatura prelazi maksimalnu vrijednost pri kojoj uređaj još može raditi. Zbog toga je potrebno ugraditi hladnjak koji smanjuje unutarnju temperaturu uređaja tako da odvodi toplinu iz uređaja u okolinu.

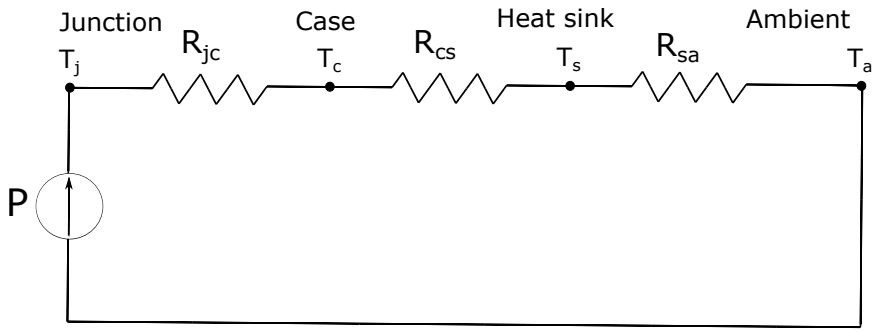
Da bi se optimirala temperatura uređaja, potrebno je odrediti parametre prijelazne toplinske impedancije na temelju kataloške krivulje koju daje proizvođač tranzistora, no kako postoji relativno velik prostor rješenja, u ovom radu se implementira vlastita inačica genetskog algoritma kojom se određuju ti parametri. Rezultati i detaljniji opis uz definicije potrebne za shvaćanje ovog problema bit će prikazani u nastavku rada. Genetski algoritam je pogodno koristiti zato što je implementacija jednostavna te se kroz relativno mali broj generacija dobivaju rezultati koji s malom greškom opisuju krivulju danu u katalogu.

2 Prijelazna toplinska impedancija

Toplinski otpor definira se kao razlika temperatura između dvije točke podijeljena s toplinskom snagom (1).

$$R_{th} = \frac{T_1 - T_2}{P} \quad [K/W] \tag{1}$$

Protok topline između poluvodiča s hladnjakom ugrađenim na kućište i okoline može se prikazati pojednostavljenim modelom kao serija toplinskih otpora. Takav model je analogija električnog kruga gdje je snaga analogna sa strujnim izvorom, toplinski otpor s električnim otporom, a razlika u temperaturi s električnim potencijalom [1]. Otpornici sa slike 1 predstavljaju: toplinski otpor između unutrašnjosti i kućišta poluvodiča (R_{jc}), toplinski otpor između kućišta i hladnjaka koji je ugrađen na njega (R_{cs}), te toplinski otpor između hladnjaka i okoline (R_{sa}).



Slika 1: Ekvivalentni električni krug

2.1 Vremenski promjenjiva snaga

Slika 1 prikazuje model u kojem je izvor toplinske snage konstantan. Ako se na izvor dovede step pobuda, kondenzatori se uključuju u paralelu s pripadnim toplinskim otporima i događa se prijelazna pojava nabijanja kondenzatora. Kondenzatori predstavljaju pohranjenu toplinsku energiju. Električni krug sa slike 2 se koristi za analizu temperature čiji je eksponencijalni izgled krivulje (slika 3) rezultat vremenski promjenjive snage.

Vremenska konstanta je definirana kao umnožak toplinskog otpora i kapaciteta ($\tau = R_{th}C$), a prijelazna toplinska impedancija paralelnog spoja jednaka je:

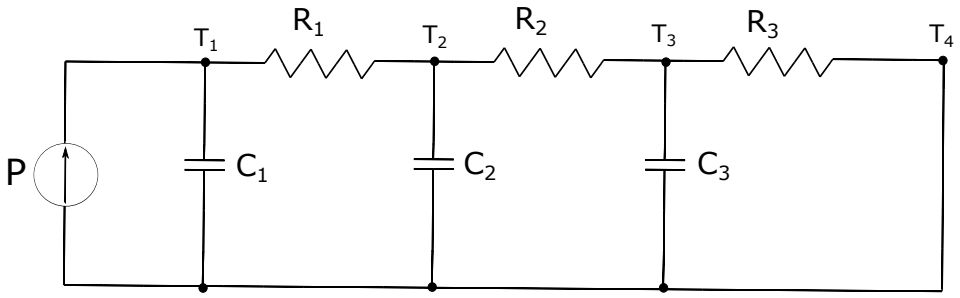
$$Z_{th} = \frac{R_{th}}{1 + s\tau} \tag{2}$$

Paralelni spojevi otpornika i kondenzatora su u seriji, pa za cijeli krug vrijedi:

$$Z_{th} = Z_{th1} + Z_{th2} + Z_{th3} \tag{3}$$

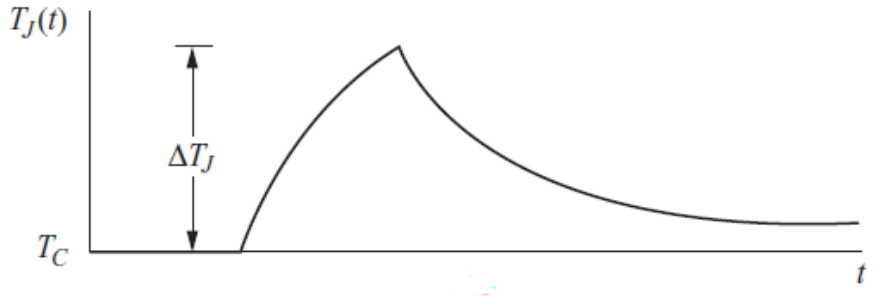
Djelovanjem step pobude na cijeli sustav dobiva se krivulja koja se uspoređuje s kataloškom krivuljom koju daje proizvođač uređaja.

Prijelazna toplinska impedancija od unutrašnjosti uređaja do kućišta $Z_{\Theta,th}$ se koristi kako bi se odredila promjena temperature unutrašnjosti uređaja koja je rezultat trenutnih promjena snage.



Slika 2: Ekvivalentni električni krug za vremenski promjenjivu snagu.

RC model (slika 2) može predstavljati cijeli sustav koji se sastoji od uređaja, kućišta, hladnjaka i okoline ili samo jednu od tih komponenata.



Slika 3: Odziv temperature na impuls snage.

3 Genetski algoritmi

Genetski algoritmi su stohastičke metode optimiranja osnovane na principu prirodnog evolucijskog procesa. Evolucijski proces se imitira metodama selekcije, križanja i mutacije. Uporaba genetskih algoritama primjerena je za optimizacijske probleme koje nije moguće riješiti egzaktnim matematičkim postupkom, no oni za razliku od egzaktnih postupaka, ne garantiraju pronalazak najboljeg mogućeg rješenja. Korisnik određuje uvjete zaustavljanja (vremenski period, broj iteracija, dobrota) i kada je bilo koji od tih uvjeta zadovoljen, rad se zaustavlja, a kao rješenje se vraća trenutno najbolja jedinka iz populacije.

Evolucijski proces se, kao i genetski algoritam, temelji na nekoliko pretpostavki. U svakom procesu postoji populacija jedinki koje se razlikuju po svojoj dobroti. Bolje jedinke imaju veću vjerojatnost preživljavanja i reprodukcije. Svaka jedinka ima svojstva zapisana u kromosomu te njezini potomci nasljeđuju dio njenih svojstava. Osim što nasljeđuju svojstva roditelja, nad potomcima može djelovati mutacija koja omogućava ulazak novog genetskog materijala u populaciju.

Pseudokod 1: Rad genetskog algoritma

```
generiraj početnu populaciju P(0);
dok nije zadovoljen uvjet zaustavljanja radi
    evaluiraj fitnesse;
    odaberi P(i) iz P(i - 1);
    križaj P(i);
    mutiraj P(i);
kraj
vrati najbolju jedinku iz populacije;
```

3.1 Prikaz rješenja

Prikaz rješenja vrlo često može utjecati na učinkovitost rada cijelog algoritma. Nekoliko najčešćih prikaza su:

- binarni prikaz
- prikaz Grayevim kodom
- prikaz poljem ili vektorom brojeva
- prikaz stablima

Kada je to moguće, koristi se binarni prikaz zato što su operacije nad bitovima sklopovski implementirane i time se postiže velika brzina pri izvršavanju operacija definiranim nad operatorima, no nedostatak binarnog prikaza je to što se svaka jedinka na kraju rada treba ponovno pretvoriti u broj s pomičnim zarezom.

3.2 Funkcija dobrote

Funkcije dobrote (engl. *fitness*) je ocjena dobrote pojedine jedinke. Vjerojatnost preživljavanja svake jedinke je u manjoj ili većoj mjeri proporcionalna s njenom dobrotom, stoga je evaluacija jedinke ključan korak za daljnji tijek izvršavanja genetskog algoritma. Za svaki problem potrebno je nanovo definirati funkciju dobrote, no, ona se u nekom najjednostavnijem slučaju može zapisati kao ekvivalent funkciji f koju treba optimizirati, odnosno $dobrota(v) = f(x)$.

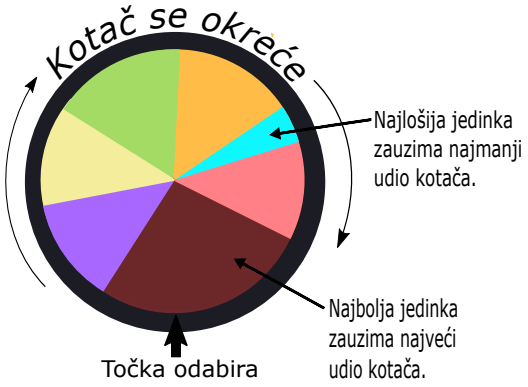
3.3 Selekcija

Selekcija je dio genetskog algoritma čija je namjena očuvanje i prijenos dobrih svojstava jedinki u sljedeću generaciju. Osnovna podjela:

- **generacijska** - nove jedinke nastaju na temelju kopija jedinka iz prethodne generacije
- **eliminacijska** - neke jedinke se eliminiraju te se na njihovom mjestu stvaraju nove

Unutar ove podjele razlikuje se nekoliko vrsta selekcija, od kojih su najčešće:

- **jednostavna generacijska selekcija** (engl. *roulette-wheel selection*) (slika 4) - vjerojatnost odabira jedinke je proporcionalna s dobrotom te se bolje jedinke u populaciji pojavljuju više puta
- **jednostavna eliminacijska selekcija** (engl. *elimination selection*) - određeni postotak populacije se eliminira i zamijenjuje novim jedinkama koje nastaju uporabom genetskih operatora
- **k-turnirska selekcija** - nasumičan odabir k jedinki od kojih se izabire najlošija i zamijenjuje novom koja nastaje uporabom genetskih operatora



Slika 4: Jednostavna generacijska selekcija.

Važno je napomenuti da determinizam (odabir samo najboljih jedinki iz populacije ili eliminacija samo najgorih) ne daje dobra rješenja zato što to stvara veliki selekcijski pritisak i algoritam vrlo brzo zastaje u nekom lokalnom optimumu.

3.4 Operatori

3.4.1 Križanje

Nakon što je odabrano koje jedinke će preživjeti, potrebno je stvoriti nove potomke te im osigurati genetski materijal koji je sastavljen od materijala njihovih roditelja. Upravo to što djeca nasljeđuju svojstva svojih roditelja je najvažnija karakteristika križanja zato što, ako su roditelji dobri, najvjerojatnije će biti dobra i njihova djeca, a možda čak i bolja.

Podjela križanja:

- **križanje s jednom točkom prekida** (engl. *single-point crossover*) - uzima jedan dio materijala iz prvog roditelja, a drugi dio materijala iz drugog i spaja u jedno ili dva novonastala djeteta
- **križanje s dvije ili više točka prekida** (engl. *multi-point crossover*) (analogno)
- **jednoliko (uniformno) križanje** - izvodi se kao logička operacija nad bitovima
- **aritmetičko križanje** - slučajni broj između vrijednosti dva kromosoma roditelja
- **segmentirano križanje** (engl. *segmented crossover*) - prepisuju se bitovi iz nasumično odabranog roditelja, a zatim mijenjaju s određenom vjerojatnošću

3.4.2 Mutacija

Mutacija je unarni operator kojim se s nekom vjerojatnošću na slučajan način mijenja jedan dio jedinke. Selekcijom boljih jedinki algoritam pokušava brže konvergirati k nekom rješenju, dok je uloga mutacije da algoritam vadi van iz lokalnih optimuma te time daje šansu pronalaska prostora u kojem se nalazi najbolje moguće rješenje.
Neki primjeri mutacije kod brojeva s pomičnim zarezom:

- **Gaussova mutacija** - trenutnom rješenju se dodaje slučajni broj koji se ravna po normalnoj distribuciji s parametrima μ (srednja vrijednost) i σ (standardno odstupanje)
- **jednolika mutacija** - trenutnom rješenju se dodaje slučajni broj iz nekog intervala
- **granična mutacija** - rješenje se pomiče na granicu područja

3.5 Uvjeti zaustavljanja

Najbolje moguće rješenje nije poznato pa je od strane korisnika potrebno definirati uvjete zaustavljanja. Ako je bilo koji od uvjeta zadovoljen, algoritam prestaje s radom.

Neki od uvjeta zaustavljanja su:

- broj generacija
- broj evaluacija funkcije cilja
- dobrota dobivenog rješenja
- vremensko ograničenje
- broj generacija s jednakom dobrotom rješenja

3.6 Parametri genetskog algoritma

Ovisno o odabranom tipu selekcije, mutacije i križanja definiraju se sljedeći parametri:

- veličina jedinke (binarni prikaz)
- veličina populacije
- broj jedinki za eliminaciju (eliminacijska selekcija)
- veličina turnira (k-turnirska selekcija)
- vjerojatnost križanja (generacijski genetski algoritam)
- vjerojatnost mutacije jedinke

Postavi li se vjerojatnost mutacije na vrijednost koja je bliska jedinici, algoritam se pretvara u slučajnu pretragu, a suprotno tome, ako vjerojatnost teži u nulu, cijela populacija vrlo brzo poprima vrijednost najbolje jedinke iz populacije (ostaje u lokalnom optimumu). Nadalje, parametri uvelike utječu na performanse genetskog algoritma stoga njihovo podešavanje iziskuje veliki broj eksperimenata. Jednostavnim ponavljanjem eksperimenta s istim parametrima se također može povećati kvaliteta dobivenog rješenja.

4 Identifikacija parametara prijelazne toplinske impedancije genetskim algoritmom

U ovom poglavlju je opisana implementacija, dana su dodatna objašnjena korištenih operatora te su priloženi dobiveni rezultati za određene parametre.

Kod 1: Rad genetskog algoritma

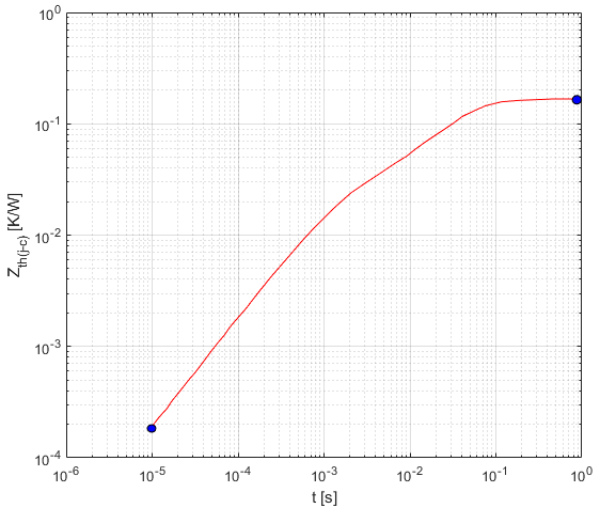
```
1  public Individual startGeneticAlgorithm() {
2     // Generates population with random individuals.
3     population = new Population(popSize, paramLoader);
4
5     // Connecting Java to running MATLAB session.
6     String[] engines = MatlabEngine.findMatlab();
7     MatlabEngine matEng = MatlabEngine.connectMatlab(engines[0]);
8
9     for (int iteration = 1; iteration < nOfIter; iteration++) {
10       evaluation.evaluateFitness(population, matEng);
11       population = selection.select(population, crossover, mutation);
12    }
13
14    // Evaluate fitness one more time because some individuals
15    // changed in last iteration without updating their fitness.
16    evaluation.evaluateFitness(population, matEng);
17
18    // End MATLAB session.
19    matEng.close();
20
21    // Return the best individual from population.
22    return population.getTheBestIndividual();
23 }
```

4.1 Prikaz rješenja

Jedinka je zapisana kao polje brojeva s pomičnim zarezom uz jednu vrijednost koja određuje njezinu dobrotu. U prvoj polovici polja se nalaze toplinski otpori, a u drugoj pripadne vremenske konstante (slika 5).

Slika 5: Strukutura jedinke.

Za jedinku se također definiraju gornje i donje granice vrijednosti. One se određuju kao maksimalne i minimalne vrijednosti koje kataloška krivulja može poprimiti. Vrijednosti koje kataloška krivulja poprima očitane su programskim alatom "Web plot digitizer" u četrdesetak točaka.



Slika 6: Granice otpora i vremenske konstante očitane iz kataloške krivulje (oznaka plavom bojom).

4.2 Evaluacija

Evaluacija se vrši na temelju funkcije "z_th.m" (oznaka za prijelaznu toplinsku impedanciju) napisane u MATLAB-u i ona je dana već gotova napisana od strane voditelja seminara. Povezivanje MATLAB-a s Javom odvija se na početku rada genetskog algoritma, a sesija se zatvara pri završetku.

Izračun u MATLAB-u se za cijelu populaciju odvija samo u prvoj iteraciji, a kasnije isključivo za novododane jedinke. Iako se povezivanjem s MATLAB-om svjesno usporava rad algoritma, u sklopu ovog rada nije toliko stavljen naglasak na to zato što je i sam cilj rada bio istražiti i naučiti ponašanje genetskih algoritama.

Funkcija pogreške se računa kao suma kvadrata razlike izračunatih i izmjerenih vrijednosti toplinskih otpora iz kataloške krivulje. Funkcija dobrote se može računati kao negativna vrijednost funkcije pogreške (4), zato što će u najgorem slučaju vrijednost ove razlike biti jednaka nuli (točke se podudaraju), a za sve veće pogreške dobit ćemo vrijednost koja je manja što i odgovara definiciji funkcije dobrote.

$$fitness(v) = -\sum(r_{calc} - r_{ds})^2 \tag{4}$$

Kod 2: Izračun funkcije dobrote

```
1 private double evaluateFitness(Individual i, MatlabEngine matEng) {
2    // Saving result from MATLAB in new array.
3    // Result is negative fitness value.
4    Object[] result = matEng.feval(2, "z_th", i.getData());
5
6    double fitness = - (double)result[0];
7    return fitness;
8 }
```

4.3 Selekcija

Turnirska selekcija s k turnira je opisana u poglavlju 3.3. Za 3-turnirsku selekciju vrijedi da se na temelju tri nasumično odabrane jedinke izbacuje najlošija i zamjenjuje s novom koja nastaje križanjem dvije preostale jedinke. Taj postupak se ponavlja za cijelu populaciju. Može se primijetiti da je za ovakvu vrstu selekcije inherentno ugrađen elitizam (očuvanje najbolje jedinke).

Kod 3: Odabir populacije za novu generaciju

```
1  public Population select(Population population, Crossover crossover,
       Mutation mutation) {
2     // Shuffle the population and iterate over groups of three.
3     // This simulates random index generation without duplicates.
4     population.shuffle();
5
6     for (int i = 0; i < population.size(); i += 3) {
7        // If last group has less than three elements, end selection.
8        if (i + 1 == population.size() || i + 2 == population.size()) {
9           break;
10       }
11
12       Individual i1 = population.getIndividual(i);
13       Individual i2 = population.getIndividual(i + 1);
14       Individual i3 = population.getIndividual(i + 2);
15
16       // Replace the worst individual with the new one.
17       if (i1.getFitness() < i2.getFitness() && i1.getFitness() <
             i3.getFitness()) {
18          Individual newIndividual = crossover.doCrossover(i2, i3);
19          mutation.mutate(newIndividual);
```

```
20          population.set(i, newIndividual);
21        } else if (i2.getFitness() < i1.getFitness() && i2.getFitness()
              < i3.getFitness()) {
22          Individual newIndividual = crossover.doCrossover(i1, i3);
23          mutation.mutate(newIndividual);
24          population.set(i + 1, newIndividual);
25        } else if(i3.getFitness() < i1.getFitness() && i3.getFitness()
              < i2.getFitness()){
26          Individual newIndividual = crossover.doCrossover(i1, i2);
27          mutation.mutate(newIndividual);
28          population.set(i + 2, newIndividual);
29        }
30      }
31      return population;
32  }
```

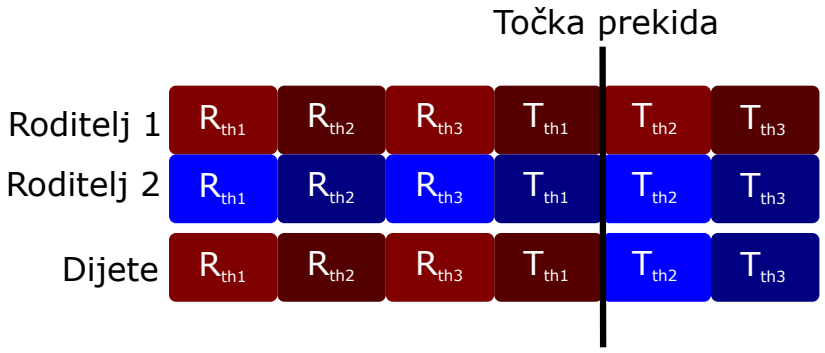
4.4 Operatori

Implementirani genetski algoritam koristi križanje s jednom točkom prekida te Gaussovu mutaciju.

4.4.1 Križanje

Križanje se obavlja tako da se nasumično odabere jedna točka u kojoj se polja roditelja dijele na dva dijela. Takvim načinom mogu nastati jedno ili dva djeteta. Dijete od početka do prve točke prekida uzima genetski materijal od jednog, a preostali dio od drugog roditelja.

U ponuđenoj implementaciji križanjem nastaje jedno novo dijete (slika 7) zato što je točno toliko potrebno da bi se izbacivanjem po jedne jedinke iz svake trojke u novoj generaciji dobila populacija veličine jednake kao u prošloj generaciji.



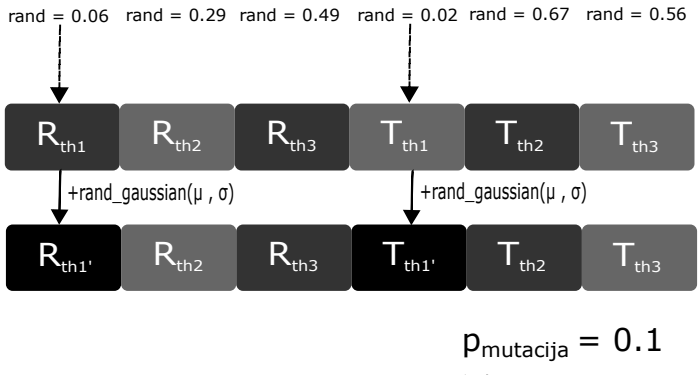
Slika 7: Križanje s jednom točkom prekida.

Kod 4: Križanje s jednom točkom prekida

```
1  public Individual doCrossover(Individual parent1, Individual parent2){
2     int size = parent1.getDataLength();
3     double data[] = new double[size];
4     int point = ThreadLocalRandom.current().nextInt(0, size);
5
6     for (int i = 0; i < point; i++) {
7        data[i] = parent1.getData(i);
8     }
9
10    for (int i = point; i < size; i++) {
11       data[i] = parent2.getData(i);
12    }
13
14    return new Individual(data);
15 }
```

4.4.2 Mutacija

Gaussova mutacija (slika 8) modificira podatak unutar jedinke s nekom vjerojatnošću $p_{mutacija}$, odnosno, početno se generira slučajan broj u intervalu $[0, 1]$ te tek ako je generirani broj manji od $p_{mutacija}$, ona se odvija tako da se trenutnoj vrijednosti dodaje slučajan broj koji se ravna po normalnoj razdiobi s parametrima srednja vrijednost μ i standardna devijacija σ. Prednost dodavanja takvog broja ispred broja koji se ravna po jednolikoj razdiobi je to što će se u većini slučajeva dodati mali pomak od srednje vrijednost, ali još uvijek će se u manjem broju slučaja dodati broj s većim odstupanjem. Na taj način se umanjuje faktor slučajnosti kod dodavanja vrijednosti, ali i omogućava to da je nova vrijednost podatka u većem broju slučaja i dalje unutar zadane domene [*donja granica*, *gornja granica*].



Slika 8: Gaussova mutacija

Kod 5: Gaussova mutacija

```
1  public void mutate(Individual i) {
2     Random r = new Random();
3     double newValue;
4
5     // Mutation for thermal resistance.
6     for (int k = 0; k < i.getDataLength() / 2; k++) {
7        double rnd = ThreadLocalRandom.current().nextDouble(0, 1);
8        if (rnd <= mutationProbability) {
9           do {
10             // Add Gaussian random value.
11             newValue = i.getData(k) + r.nextGaussian() * rStdDev;
12             i.setData(newValue, k);
13          } while (!(newValue > rLowerBound && newValue < rUpperBound));
14       }
15    }
16
17    // Mutation for time constant.
18    for (int k = i.getDataLength() / 2; k < i.getDataLength(); k++) {
19       double rnd = ThreadLocalRandom.current().nextDouble(0, 1);
20       if (rnd <= mutationProbability) {
21          do {
22             // Add Gaussian random value.
23             newValue = i.getData(k) + r.nextGaussian() * tStdDev;
24             ind.setData(newValue, k);
25          } while (!(newValue > tLowerBound && newValue < tUpperBound));
26       }
27    }
28 }
```

4.5 Parametri genetskog algoritma

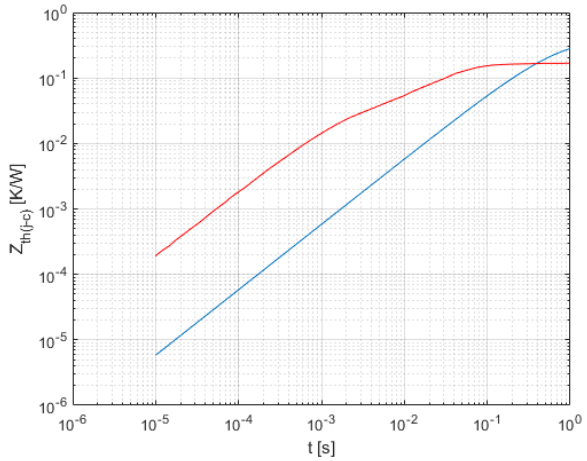
Uz implementaciju napisana je i tekstualna datoteka iz koje se čitaju parametri genetskog algoritma. Parametri su pohranjeni u mapi gdje je ključ ime parametra (u obliku enumeracije), a vrijednost je znakovni niz koji se, ovisno o tipu podatka, parsira kao cjelobrojna vrijednost ili broj s pomičnim zarezom.

Od parametara navedenih u poglavlju 3.6, ovdje se koristi veličina populacije, vjerojatnost mutacije, broj iteracija, veličina polja u kojoj je pohranjena jedinka, te se uz to definira i standardna devijacija toplinskog otpora i vremenske konstante. Unutar iste datoteke su definirane i gornje i donje vrijednosti otpora i vremenskih konstanta zato što se one također mijenjaju u ovisnosti o tome koja se kataloška krivulja optimira.

4.6 Rezultati

U rezultatima su dani grafovi koji prikazuju razlike između krivulja dobivenih genetskim algoritmom (označene plavom bojom) te krivulje očitane iz kataloga (označena crvenom bojom). Uz to su dani korišteni parametri te vrijednosti funkcije dobrote.

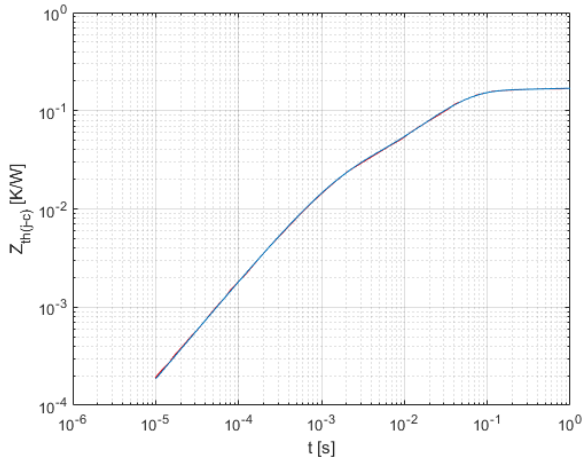
Slika 9 prikazuje kako izgleda aproksimacija kataloške krivulje nasumično odabranim vrijednostima toplinskih otpora i vremenskih konstanta. Funkcija dobrote za odabrane vrijednosti iznosi -0.0891. Ovako velika greška i udaljenost od kataloške krivulje je karakteristična za sve jedinke početne populacije.



Slika 9: Rezultat prije početka rada genetskog algoritma.

Dobrote jedinke sa slike 10 iznosi $-1.151 \cdot 10^{-5}$, a zadani parametri su postavljeni na sljedeće vrijednosti:

Parametar	Vrijednost
POPULATION_SIZE	200
MUTATION_PROBABILITY	0.1
R_STANDARD_DEV	0.1
T_STANDARD_DEV	0.5
NUMBER_OF_ITERATIONS	200000
DIMENSIONALITY	6



Slika 10: Rezultat nakon završetka rada genetskog algoritma.

Zanimljivo je primjetiti da je za deset puta manji broj iteracija, a ostale nepromjenjene parametre, u nekim izvršavanjima dobivena dobrota iznosila oko $-2 \cdot 10^{-5}$, ali je jednako tako za ove identične vrijednosti parametara rezultat nekada bio znatno lošiji.

5 Zaključak

Genetski algoritmi u današnje vrijeme više nisu inovacija u računarstvu te postoji mnogo komercijalne programske podrške koja nudi već gotove implementacije koje su vrlo efikasne i jednostavne za korištenje. Sve što je potrebno za njihovo pokretanje je pisanje funkcije koja će evaluirati dobrotu jedinke i postavljanje uvjeta zaustavljanja. Takav jednostavan način korištenja ima smisla zato što se genetski algoritmi primjenjuju za širok spektar problema koji nisu nužno vezani uz domenu računarstva, no, cilj ovog rada nije bio napisati najefikasniji genetski algoritam nego upoznavanje s njima, te pokazati da ih je moguće primjeniti i na ovu vrstu problema, što je u većoj mjeri pokazano u rezultatima rada.

Izvori

[1] Daniel W. Hart; ***Power electronics***; Heat sinks and thermal managment, The McGraw-Hill Companies, 2011.

[2] Wikipedia; **Heat sink**, `https://en.wikipedia.org/wiki/Heat_sink`

[3] Marko Čupić; ***Prirodom inspirirani optimizacijski algoritmi. Metaheuristike.***, 2012.

[4] Marin Golub; ***Genetski algoritam prvi dio***, `www.zemris.fer.hr/~golub/ga/ga_skripta1.pdf`, 1997.

[5] Marin Golub; ***Genetski algoritam drugi dio***; `http://www.zemris.fer.hr/~golub/ga/ga_skripta2.pdf`, 2004.

[6] Domagoj Jakobović; ***Genetski algoritmi - predavanje***; `http://www.fer.unizg.hr/_download/repository/GApredavanje.pdf`, 2015.

6 Sažetak

Svrha ovog rada je identificirati parametre prijelazne toplinske impedancije (toplinski otpor, vremenska konstanta) metodom optimizacije. Rad sažeto opisuje definciju prijelazne toplinske impedancije te osnovne pojmove potrebne za razumijevanje teme. Ti pojmovi se primarno odnose na elemente genetskog algoritma zato što je to ujedno i korištena metoda optimizacije. Uz opis genetskog algoritma je implementirana i vlastita inačica. Implementacija je zamišljena kao upoznavanje s genetskim algoritmima na nekom jednostavnijem primjeru koji opisuje problem koji se može pronaći u stvarnom životu. Uz samu implementaciju su priloženi i rezultati.