SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6847

Evolucijska optimizacija upravljanja električnom energijom temeljena na primjeni pametnih mreža

Marko Ivić

Zagreb, lipanj 2020.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

Hvala svim bližnjima na bezuvjetnoj podršci

SADRŽAJ

1. Uvod

Evolucijsko računanje je metoda računarske znanosti kojom se vrše izračuni koji se ne mogu riješiti klasičnim matematičkim putem. Ovaj pristup računarstvu je inspiriran Darwinovom teorijom evolucije što je naposljetku dovelo do razvoja brojnih optimizacijskih algoritama inspiriranih prirodom. Pojedini biološki mehanizmi se mogu izravno preslikati u evolucijske algoritme, dok su neki evolucijski algoritmi potpuno zasnovani na matematičkoj pozadini.

Primjena evolucijskog računarstva je vrlo zastupljena u računarskoj znanosti upravo zbog svoje široke primjenjivosti. Problemi koje je potrebno rješavati često postaju preveliki ili nerješivi za uobičajene pristupe. Domene problema koji se mogu rješavati su razne, od vektora realnih brojeva do vektora cijelih brojeva u proizvoljnim bazama pa sve do optimizacije simboličkih izraza (genetsko programiranje). Konkretne primjene mogu biti razne, a ovaj rad se bavi optimizacijom problema u domeni upravljanja električnom energijom.

Izazov modernog svijeta je ispravno skladištenje i proizvodnja električne energije kako bi se zadovoljile sve potrebe. Osim toga dodatan problem stvara nepredvidljivost više aspekata koji utječu na proizvodnju i isplativost. Stoga je potrebno dobiti robusno rješenje. Tržište je raznovrsno pa se tako osim proizvođača i potrošača mogu naći i oni koji su i jedno i drugo. U zadatku je potrebno optimizirati ponašanje mreže tako da se uspješno pohrani električna energija za iduća 24 sata.

U drugom poglavlju se nabrajaju i opisuju korišteni algoritmi. Zatim se pobliže opisuje problem te njegove specifičnosti. Sljedeće poglavlje opisuje korištene postavke za svaki algoritam. Nakon toga se u posebnom poglavlju prikazuju rezultati te se daje osvrt na njih. Naposljetku se daje zaključak na temelju cjelokupnog rada

2. Algoritmi

U evolucijskom računarstvu postoji *No free lunch theorem* zbog kojeg ne postoji ispravan niti najbolji algoritam za rješavanje određenog optimizacijskog problema. Stoga se ovaj rad bavi usporedbom više različitih algoritama ispitanih na istom problemu. Neki algoritmi sadrže varijacije u određenim dijelovima kako bi se proširili načini pretraživanja.

2.1. Genetski algoritam

Prvi algoritam koji će se razmatrati je genetski algoritam. On se sastoji od jednostavnog procesa selekcije, križanja i mutiranja. Ovaj rad razmatra jednu vrstu selekcije te više vrsta križanja i mutiranja rješenja. Pseudokod za genetski algoritam može se vidjeti u algoritmu 1.

Algoritmu se predaje nasumično inicijalizirana populacija veličine n. Jedinke su generirane unutar zadanog raspona za specifičan problem. Korištena je k-turnirska selekcija koja odabire nasumično k roditelja i vraća najboljeg [4]. Zatim je korišteno više operatora križanja i mutacije. Operatori križanja djeluju na roditelje te stvaraju dijete. Operatori križanja Arithmetic Simple, BGA, BLX-α i Onepoint Crossover su implementirani po njihovoj definiciji [3]. Posljednji operator križanja je operator Composite Crossover koji pri svakom svom pozivu nasumično odabire jedan od prethodno definiranih algoritama te križanje radi po njemu. Implementirani operatori su prikazani u tablici 2.1. Operator mutacije djeluje na dijete dobiveno križanjem dvaju roditelja. Cilj mutacije je izmijeniti kromosom djeteta tako da dijete nije samo rekombinacija gena roditelja. Operatori Gaussian i Uniform mutation za svaki gen djeteta dodaju, s nekom vjerojatnošću, nasumičan broj generiran po normalnoj odnosno uniformnoj razdiobom. Operator Reset gene svaki gen djeteta nanovo inicijalizira, s određenom vjerojatnošću, unutar raspona vrijednosti definiranog za taj gen. Operator mutacije Composite mutation pri svakom svom pozivanju nasumično odabire jedan od prethodno navedenih operatora mutacije te djete mutira po njemu. Operatori mutacije se mogu vidjeti u

Algorithm 1 Genetski algoritam

Ulaz: pop – Početna populacija
Ulaz: $mate$ – Operator križanja
Ulaz: $mutate$ – Operator mutacije
Ulaz: $maxiter$ – Maksimalni broj iteracija
Ulaz: $lowerB$ – Donja granica rješenja
Ulaz: $upperB$ – Gornja granica rješenja
Izlaz: $bestIndividual$ – Najbolja jedinka
gen := 1
while $gen < maxiter$ **do**
  $limitInRange(pop_{gen}, lowerB, upperB)$
  $evaluate(pop_{gen})$
  $add(pop_{gen+1}, best(pop_{gen}))$
  $pop_{gen} = scale(pop_{gen}, 0, 1)$
  for $(i := 0; i < |pop_{gen}| - 1; inc(i))$ **do**
    $parent1 = kTournament(pop_{gen}, k)$
    $parent2 = kTournament(pop_{gen} \setminus \{parent1\}, k)$
    $child = mate(parent1, parent2)$
    $child = mutate(child)$
    $add(pop_{gen+1}, child)$
  end for
  $pop_{gen+1} = scale(pop_{gen+1}, lowerB, upperB)$
  $gen = gen + 1$
end while
$bestIndividual = best(pop)$
return bestIndividual

tablici 2.2.

Tablica 2.1: Operatori križanja

Operatori križanja
Arithmetic Simple
BGA
BLX-α
Onepoint Crossover
Composite Crossover

2.2. Simulirano kaljenje

Sljedeći razmatrani algoritam je algoritam simuliranog kaljenja. Ovo je također još jedan prirodom inspiriran optimizacijski algoritam. Njegova fizikalna pozadina je proces hlađenja željeza pri čemu željezo dobiva neka poželjna svojstva. Algoritmu se predaje početna populacija inicijalizirana unutar zadanih granica za određeni problem.

Tablica 2.2: Operatori mutacije

Operatori mutacije
Gaussian Mutation
Uniform Mutation
Reset Gene
Composite Mutation

Glavna ideja algoritma simuliranog kaljenja jest prihvaćanje lošijih rješenja s većom vjerojatnošću na početku iteracija te opadanjem te vjerojatnosti kroz iteracije ovisno o planu hlađenja. Ovaj rad koristi geometrijski plan hlađenja. Cilj ovakvog načina odabiranja rješenja je smanjiti vjerojatnost da algoritam zapne u lokalnom optimumu. Algoritam radi na principu stvaranja populacije susjedstva trenutne generacije tako da se postojećim rješenjima dodaje vektor nasumičnih brojeva generiranih po normalnoj razdiobi. Vektor nasumičnih brojeva se iznova generira za svaku jedinku te postoji vjerojatnost prihvaćanja svake generirane vrijednosti tog vektora s vjerojatnosti $changeChance$ inače ta vrijednost se postavlja na 0. Lošije rješenje se prihvaća s vjerojatnošću $e^{(-diff/currTemp)}$ gdje je $diff$ razlika dobrota rješenja susjeda i rješenja u populaciji. Pseudo kod je dan u algoritmu 2.

2.3. Modificirani Frankenstein PSO

Algoritam roja čestica je još jedan poznati algoritam za optimizaciju vektora realnih brojeva. U ovom radu ćemo razmatrati modificirani *Frankenstein PSO*, originalni algoritam je dan u [2]. Ideja *Frankenstein PSO* algoritma je kombiniranje mehanizama više vrsta algoritama roja čestica kako bi se prednosti pojedinačnih algoritama spojile u jedan algoritam. Značajke *Frankenstein PSO* algoritma su

- ažuriranje topologije susjedstva kroz iteracije od maksimalne do minimalne povezanosti
- ažuriranje utjecaja inercije na trenutnu brzinu ovisno o iteraciji
- za svaku pojedinu česticu se radi izračun nove brzine na temelju inercije i sume utjecaja čestica iz susjedstva
- novo rješenje se dobiva zbrajanjem starog rješenja s novom brzinom

. Modificirana verzija *Frankenstein PSO*-a dijeli gen u odjeljke koji se slijedno optimiziraju. Algoritmu je također moguće predati proizvoljnu funkciju koja upravlja

Algorithm 2 Simulirano kaljenje

Ulaz: pop – Početna populacija
Ulaz: α – Geometrijski koeficijent hlađenja
Ulaz: $initTemp$ – Početna temperatura
Ulaz: $tempTries$ – Broj evaluacija za temperaturu $currTemp$
Ulaz: $lowerB$ – Donja granica rješenja
Ulaz: $upperB$ – Gornja granica rješenja
Ulaz: $changeChance$ – Vjerojatnost s kojom će svaki gen generirati susjedstvo za sebe
Izlaz: $bestIndividual$ – Najbolja jedinka

```
gen := 1
bestInIter = []
evaluate(pop_gen)
while gen < maxiter do
    currTemp = α^gen * initTemp
    for (try := 0; try < tempTries; inc(try)) do
        pop_gen = scale(pop_gen, 0, 1)
        neighborhoodPop = generateNeighborhood(pop_gen, |pop_gen.individual|, μ, σ, changeChance)
        pop_gen = scale(pop_gen, lowerB, upperB)
        neighborhoodPop = scale(neighborhoodPop, lowerB, upperB)
        limitInRange(neighborhoodPop, lowerB, upperB)
        evaluate(neighborhoodPop)
        for (i := 0; i < |pop_gen|; inc(i)) do
            diff = neighborhoodPop_i.fitness − pop_gen,i.fitness
            if diff <= 0 then
                pop_gen,i = neighborhoodPop_i
            else if random(0, 1) < exp(−diff/currTemp) then
                pop_gen,i = neighborhoodPop_i
            end if
        end for
    end for
    gen = gen + 1
    add(bestInIter, best(pop))
end while
bestIndividual = best(bestInIter)
return bestIndividual
```

utjecajem inercije na brzinu. Algoritmu se predaje početna populacija nasumično generirana unutar zadanih granica. Pseudo kod je dan u algoritmu 3.

Algorithm 3 Modificirani Frankenstein PSO

Ulaz: pop – Početna populacija
Ulaz: k – Raspored ažuriranja topologije
Ulaz: φ – Zbroj koeficijenata ubrzanja
Ulaz: $lowerB$ – Donja granica rješenja
Ulaz: $upperB$ – Gornja granica rješenja
Ulaz: $inertiaFunction$ – Funkcija inercije
Ulaz: $subsections$ – Odjeljci
Ulaz: $maxEvaluations$ – Broj evaluacija
Ulaz: α – Faktor brzine prolaska kroz kromosom
Izlaz: $bestIndividual$ – Najbolja jedinka
for $(subsIndex := 0; i < |subsections|; inc(subsIndex))$ **do**
  $generateVelocities(pop, lowerB, upperB)$
  $initNeighborhood(pop)$ {Every particle is a neighbor to a every other}
  $t, esteps, count, offset = 0$
  $subsSize = subsections[subsIndex]$
  $evalsPerIter = (maxEvaluations/|pop|)/|subsections|$
  $repeats = evalsPerIter/(|pop.individual|/subsSize)/\alpha$
  while $(t < evalsPerIter)$ **do**
    if $count \geq repeats$ **then**
      $offset = (offset + subsectionSize \mod |pop.individual|$
      $count = 0$
    else
      $count = count + 1$
    end if
    $evaluate(pop)$
    $updatePersonalBest(pop)$
    if $(t > 0 \wedge t \leq k \wedge t \mod \lceil k/(n-3) \rceil = 0)$ **then**
      $updateTopology(pop, esteps)$
    end if
    $\omega^t = inertiaFunction(t)$
    for $(i := 0; i < |pop_{gen}|; inc(i))$ **do**
      Generate $\mathbf{U}_m^t \forall p_m \in N_i$ {Matrix with diagonal elements initialized with uniform distribution}
      $\varphi_m = \varphi/|N_i| \forall p_m \in N_i$
      $v_i^{t+1} = \omega^t v_i^t + \sum_{p_m \in N_i} \varphi_k \mathbf{U}_k^t (pb_k^t - x_i^t)$
      $x_i^{t+1}[offset : subsSize + offset] = x_i^t[offset : subsSize + offset] + v_i^{t+1}[offset : subsSize + offset]$
    end for
    $t = t + 1$
  end while
end for
$bestIndividual = best(pop)$
return bestIndividual

2.4. BOBYQA i NEWUOA

Ovi algoritmi se koriste za optimizacije nad vektorima realnih brojeva. Njihova implementacija je preuzeta sa stranice https://nlopt.readthedocs.io/en/latest/. BOBYQA i NEWUOA su algoritmi koji provode kvadratnu aproksimaciju funkcije cilja, a da pri tom funkcija ne mora biti derivabilna. Korištene implementacije poštuju maksimalan broj poziva evaluacijske funkcije i granice rješenja definirane u problemu.

3. Opis problema

Opis problema kojim se ovaj rad bavi službeno je zadan unutar *testbed*a 1 na sljedećoj poveznici http://www.gecad.isep.ipp.pt/ERM-competitions/2020-2/. Problem proizlazi iz potrebe za upravljanjem električnom energijom u iduća 24 sata s ciljem ostvarivanja prihoda ili barem smanjivanja rashoda. Brojni faktori utječu na količinu proizvedene i potrošene energije. Preveliko oslanjanje na točne prognoze utjecaja određenih faktora može imati katastrofalne posljedice. Pri izračunu evaluacijske funkcije uzimaju se u obzir četiri faktora: proizvodnja električne energije putem fotonaponskih ćelija, prognoza opterećenosti mreže, veleprodajna cijena električne energije te cijena električne energije na lokalnom tržištu. Vrijednosti tih faktora su nasumično odabrane za svaku evaluaciju. Cilj optimizacije je osigurati robusnost rješenja na različite faktore koji utječu na isplativost pametne mreže.

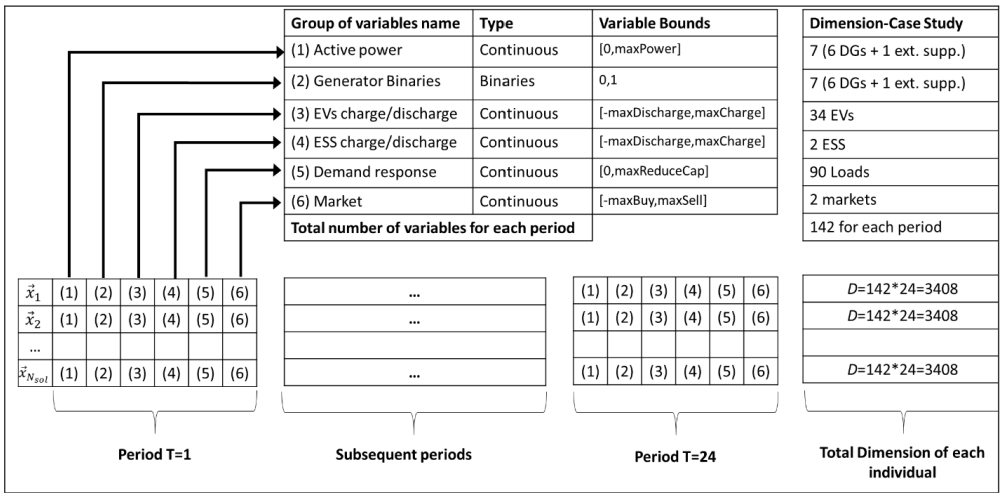
Evaluacijska funkcija pri svakoj evaluaciji odabire određen broj scenarija na kojima će se testirati predano rješenje. U ovom problemu ukupan broj scenarija je 500, a pri evaluaciji se nasumično odabire 10 scenarija. Ukupan broj scenarija i broj scenarija po evaluaciji su predefinirani u problemu te ih se ne može mijenjati.

Rješenje koje prima evaluacijska funkcija se sastoji od vektora realnih brojeva veličine $n = 142 * 24$. Odnosno optimiraju se 142 vrijednosti za svaki sat u danu. Navedene vrijednosti se mogu podijeliti u šest kategorija. Prvom kategorijom određuje se raspodijeljeno generiranje električne energije iz različitih izvora. Jedan od izvora električne energije su i fotonaponske ćelije te se na njih ne može utjecati koliko će električne energije proizvoditi. Stoga će evaluacijska funkcija zanemariti predane vrijednosti te će interno odabrati vrijednosti određene po nasumičnom scenariju. Druga kategorija sadrži binarne vrijednosti koje označavaju je li odgovarajući izvor energije iz prve kategorije upaljen ili ugašen. Ako se u evaluacijsku funkciju predaju realne vrijednosti one će se interpretirati kao binarne tako što će se zaokružiti na najbližu cijelu vrijednost. Treća kategorija su električna vozila, dok četvrtoj kategoriji pripadaju sustavi za spremanje električne energije. Kategorije tri i četiri su specifične po tome što mogu električnu energiju preuzimati ili predavati u mrežu. Kategorija pet predstavlja

odgovor mreže na potražnju. Interno će evaluacijska funkcija kažnjavati nezadovoljenu potražnju, odnosno prekomjerno generiranje energije. Posljednja, šesta, kategorija predstavlja tržište električne energije koje omogućuje kupovanje ili prodavanje električne energije. Prikaz dimenzija rješenja se može vidjeti na slici 3.1.

Specifičnost problema je veličina rješenja, vremenski skupa evaluacija i ograničen broj poziva evaluacijske funkcije. Evaluacijska funkcija se maksimalno može pozvati 5000 puta.

Problem predstavlja jednokriterijsku optimizaciju s ciljem minimizacije evaluacijske funkcije. Funkcija se može interpretirati kao $f = \sum$ troškovi - $\sum$ prihodi.



Slika 3.1: Dimenzije rješenja

4. Opisi rješenja

Ovaj rad razmatra pet algoritama za optimizaciju zadanog problema. Sljedeći algoritmi su implementirani iznova: simulirano kaljenje, genetski algoritam i modificirani *Frankenstein* PSO. Za algoritme BOBYQA i NEWUOA je preuzeta gotova implementacija. Svi algoritmi pozivaju evaluacijsku funkciju 5000 puta, što odgovara maksimalnom broju poziva definiranom u zadatku. Svi grafovi u ovom poglavlju su nacrtani s bibliotekom $matplotlib$ [1].

4.1. Genetski algoritam

Genetski algoritam ima ukupno četiri kombinacije varijablinih parametara. Razmatraju se dvije vrste križanja i dvije vrste mutiranja. Prva vrsta križanja je kompozitno križanje u kojem se nasumično odabire jedan tip križanja iz tablice 2.1. Druga vrsta križanja je križanje BLX-α. Po uzoru na prvu vrstu križanja, prva vrsta mutacije djece je kompozicija svih vrsta mutacije iz tablice 2.2. Gausolika mutacija koja se koristi u Composite mutation operatoru ima vrijednosti $\mu = 0$ i $\sigma = 1/7$. Druga vrsta mutacije je uniformno mutiranje određenih gena djece. Parametri korišteni u genetskom algoritmu mogu se vidjeti u tablici 4.1. Genetski algoritam radi nad vektorima realnih brojeva iz raspona [0, 1]. Stoga se prije pozivanja algoritma u svakoj iteraciji radi skaliranje iz raspona brojeva zadanih problemom u raspon [0, 1]. Zatim se nakon tog poziva, a neposredno prije evaluacije rješenja vektori realnih brojeva iz raspona [0, 1] skaliraju natrag na granice definirane u problemu.

Implementirani genetski algoritam najbolje rješenje u trenutnoj generaciji prenosi u iduću.

4.2. Simulirano kaljenje

Algoritam simuliranog kaljenja je potrebno prilagoditi dimenzijama rješenja. Pretraživanjem susjedstva se mijenja samo 10% vrijednosti rješenja. Vektor nasumičnih vri-

Tablica 4.1: Parametri genetskog algoritma

Veličina populacije	50
Vjerojatnost mutacije	6.5%
Broj iteracija	100
σ	1/7
μ	0
k u k-turnirskoj selekciji	3

jednosti se generira po normalnoj razdiobi s parametrima $\mu = 0$ i $\sigma = 1/7$. Algoritam pamti najbolje rješenje svake iteracije te vraća najbolje viđeno rješenje. Svi parametri su prikazani u tablici 4.2.

Tablica 4.2: Parametri simuliranog kaljenja

Veličina populacije	10
Veličina pretrage susjedstva	10%
Broj iteracija	250
Početna temperatura	1000
σ	1/7
μ	0
α	0.99
Pokušaja po temperaturi	2

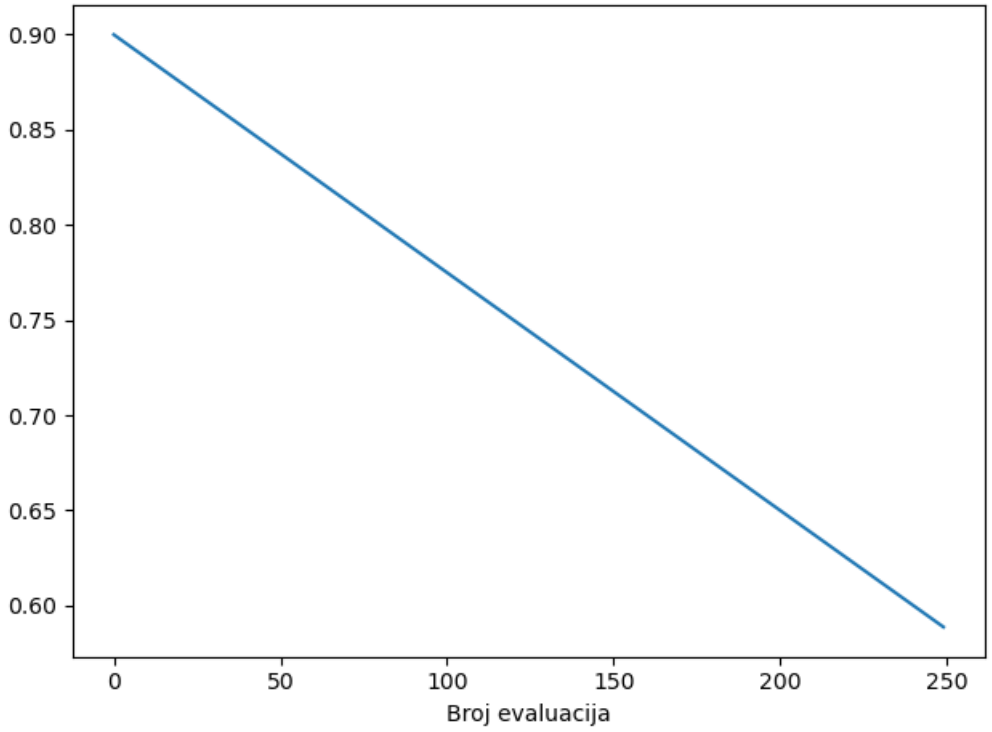
4.3. Modificirani Frankenstein PSO

Frankenstein PSO je modificiran i prilagođen za domenu problema. Osnovna ideja modificiranog algoritma je podijeliti rješenje na određen broj odjeljaka i zatim te odjeljke slijedno optimizirati jednog po jednog. Jednom kad algoritam prođe kroz sve odjeljke, optimizacija će krenuti ispočetka odnosno od prvog odjeljka. Ovo ponašanje će se ponavljati sve do isteka broja evaluacija predodređenog za taj odjeljak. Algoritmu se predaje lista s veličinama odjeljaka. Broj evaluacija za svaku veličinu odjeljka iz predane liste je $evalsPerIter = (maxEvaluations/|pop|)/|subsections|$. Odnosno algoritam će slijedno uzimati veličine odjeljaka iz liste, sa svakom veličinom obaviti prethodno opisan postupak $evalsPerIter$ puta, a zatim će uzeti sljedeću vrijednost i ponavljati ove korake sve dok se ne iskoriste sve vrijednosti iz liste. Broj evaluacija po

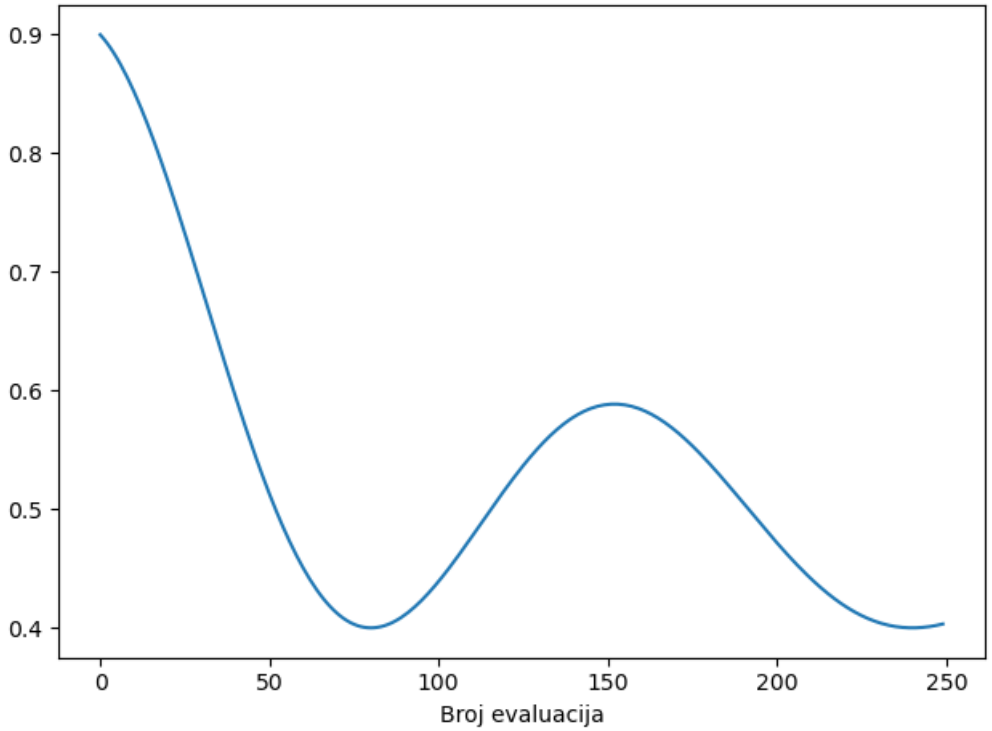
odjeljku je obrnuto proporcionalan parametru α.

Veličine odjeljaka mogu biti proizvoljne no semantički je bolje uzimati odjeljke veličine $brojSati * brojParametaraPoSatu$. Ovaj algoritam će se testirati za dvije liste koje sadrže različite veličine odjeljaka. Prva lista predstavlja optimiziranje cijelog rješenja odjednom odnosno veličinu odjeljka 24 * 142. Druga lista će najprije raditi optimizaciju u odjeljcima po osam sati, a zatim optimizirati cijelo rješenje odjednom. Pri odabiru nove veličine odjeljka nanovo će se generirati brzine čestica.

Sljedeći parametar koji će se razmatrati je jedna linearna i jedna nelinearna funkcija utjecaja inercije. Funkcija korištena u izvornom algoritmu linearno smanjuje utjecaj inercije te je zadana formulom $f(t) = \frac{\omega t_{max} - t}{\omega t_{max}} * (\omega_{max} - \omega_{min}) + \omega_{min}$. Prikaz ove funkcije je na slici 4.1. Ovaj rad razmatra još jednu funkciju utjecaja inercije $f(t) = e^{\frac{-t}{k}} * (cos(\frac{2*\pi}{k} * t)/4) + 0.4$. Prikaz ove nelinearne funkcije je na slici 4.2. Svi parametri korišteni u ovim funkcijama se mogu vidjeti u tablici 4.3.



Slika 4.1: Linearna funkcija utjecaja intercije



Slika 4.2: Nelinearna funkcija utjecaja intercije

Tablica 4.3: Parametri modificiranog *Frankenstein PSO*

Veličina populacije	20
Maksimalni broj iteracija	250
k	80
α	4
ωt_{max}	400
ω_{max}	0.9
ω_{min}	0.4
φ	4

5. Rezultati

Svi algoritmi navedeni u nastavku, za sve svoje kombinacije parametara, su pokretani deset puta. Pri svakom pokretanju nekog algoritma, funkcija za generiranje nasumičnih brojeva je inicijalizirana jedinstvenim *seed*om.

5.1. Genetski algoritam

Genetski algoritam je pokrenut s četiri kombinacije parametara. Korištena su dva operatora križanja i dva operatora mutacije. Isprobane su sve međusobne kombinacije prethodno navedenih operatora. Minimalne, maksimalne i medijalne vrijednosti za svaku kombinaciju parametara prikazane su u tablici 5.1. Rezultate genetskog algoritma moguće je vidjeti i na *boxplot* grafu 5.1 pod odgovarajućim indeksima označenima u tablici 5.4.

Iz dobivenih rezultata možemo vidjeti da su medijalne i maksimalne vrijednosti vrlo slične za sve kombinacije parametara. Ono gdje se algoritmi razlikuju su minimalne vrijednosti, a tu najbolje vrijednosti ima algoritam koji koristi blx-α kao operator križanja i kompozitni operator mutacije. Zanimljivo je primijetiti da su rješenja za algoritam koji koristi kompozitni operator križanja i uniformnu mutaciju međusobno vrlo slična. Iz toga se može naslutiti da algoritam zapada u lokalni optimum iz kojega se ne može pomaknuti.

Tablica 5.1: Rezultati genetskog algoritma

parametri	max	min	med
x: blx-α m: composite	311.456411	212.239678	273.191120
x: blx-α m: uniform	309.377124	258.985895	283.186733
x: composite m: composite	314.442302	242.274661	285.198446
x: composite m: uniform	315.031831	266.966262	285.997027

5.2. Simulirano kaljenje

Rezultati simuliranog kaljenja su prikazani u tablici 5.2. Iz *boxplot* grafa 5.1 možemo primijetiti kako su rješenja dobivena s ovim algoritmom relativno ravnomjerno raspoređena između minimalne i maksimalne vrijednosti. Za indeks simuliranog kaljenja u *boxplot* grafu 5.1 pogledati tablicu 5.4.

Tablica 5.2: Rezultati simuliranog kaljenja

max	min	med
346.01424	240.545736	311.213259

5.3. Modificirani Frankenstein PSO

U modificiranom *Frankenstein PSO* algoritmu postoje dva parametra koja mogu varirati. Prvi parametar je funkcija utjecaja inercije za koju su implementirane dvije funkcije, jedna linearna i jedna nelinearna. Drugi parametar je lista s veličinom odjeljaka po kojima će se rješenje optimizirati. To su ukupno četiri kombinacije parametara.

Iz tablice 5.3 je jasno vidljivo da algoritmi koji koriste linearnu funkciju inercije daju bolja rješenja za minimalnu, maksimalnu i medijalnu vrijednost. Gledajući parametar koji označava listu veličine odjeljaka možemo primijetiti, iz *boxplot* grafa 5.1, da su rješenja međusobno sličnija za podjelu na 8 pa 24 sata nego što su za samo jedan odjeljak od 24 sata. Iz navedenih rezultata ne možemo zaključiti da će algoritam uvijek raditi bolje za podjelu na 8 pa 24 sata. Najbolje rješenje se ostvaruje za kombinaciju parametara linearne funkcije i podjele rješenja na odjeljke od 8 i 24 sata.

Tablica 5.3: Rezultati modificiranog *Frankenstein PSO* algoritma

parametri	max	min	med
f: exp*cos subs: 24 h	221.588307	130.999441	164.332214
f: exp*cos subs: 8/24 h	232.244580	152.849052	170.441131
f: lin subs: 24 h	192.861608	101.237845	132.849782
f: lin subs: 8/24 h	151.996574	96.510622	126.012170

5.4. Međusobna usporedba algoritama

Algoritmi BOBYQA i NEWUOA daju najlošija rješenja to možemo prepisati tome da funkcija vjerojatno nije dvostruko derivabilna. Ovi algoritmi, iako ne traže da je funkcija derivabilna, ne rade dobro za takve funkcije.

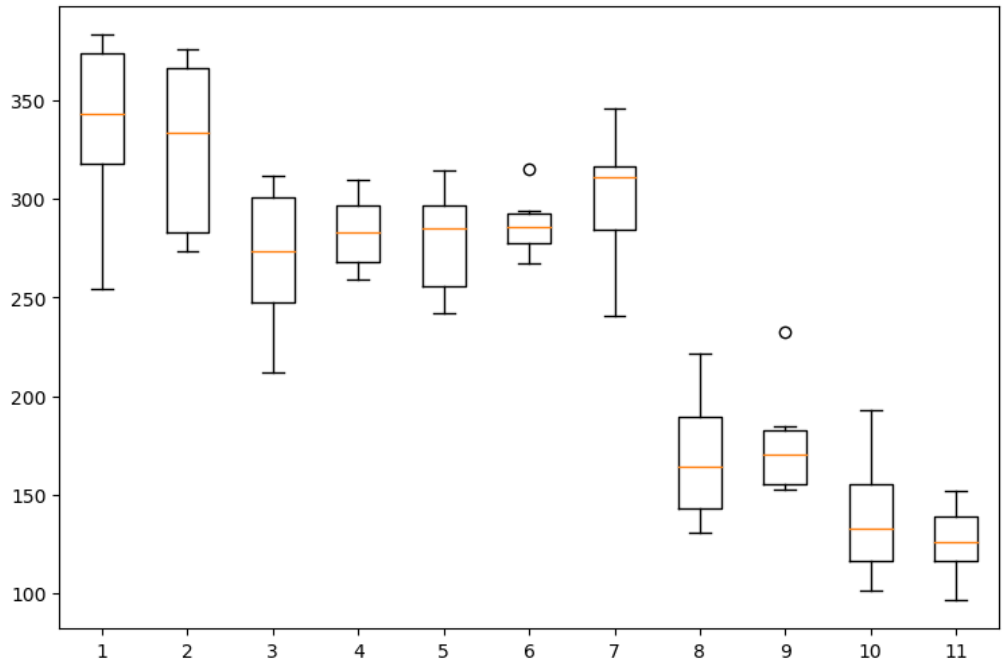
Najbolji genetski algoritam koristi parametre blx-α za operator križanja i uniformni operator mutacije. Ova rješenja su u pravilu bolja od BOBYQA i NEWUOA.

Algoritam simuliranog kaljenja daje vrijednosti rješenja u širokom rasponu te u pravilu ima lošije vrijednosti od svih genetskih algoritama te malo bolje od BOBYQA i NEWUOA.

Modificirani *Frankenstein PSO* algoritam daje najbolje rezultate za lineranu funkciju utjecaja inercije i podjelu odjeljaka na 8 pa 24 sata. Generalno govoreći ovaj algoritam daje najbolje rezultate.

Tablica 5.4: Boxplot vrijednosti

n	algoritam	postavke
1	BOBYQA	
2	NEWUOA	
3	gen alg	x: blx-α m: composite
4	gen alg	x: blx-α m: uniform
5	gen alg	x: composite m: composite
6	gen alg	x: composite m: uniform
7	sim an	
8	modded fpso	f: exp*cos subs: 24 h
9	modded fpso	f: exp*cos subs: 8/24 h
10	modded fpso	f: lin subs: 24 h
11	modded fpso	f: lin subs: 8/24 h



Slika 5.1: Boxplot rezultata algoritama

6. Zaključak

Upravljanje električnom energijom je jedan od problema s kojim se susreće moderni svijet. Prelazak proizvođača električne energije na obnovljive izvore energije skupa s povećanom potražnjom stvara izazov pri opskrbljivanju tržišta. Za rješavanje tog problema prigodno je poslužiti se modernim tehnologijama pa je tako zahvaljujući povezanosti koju pruža internet moguće upravljati velikim brojem korisnika mreže.

Autori problema ovog rada upravo razmatraju primjenu računarstva na njegovo rješavanje. Evolucijsko računarstvo se od svog začetka neprestano razvija, a zahvaljujući brojnim znanstvenim radovima ono se može razvijati i dalje.

Za optimizaciju definiranog problema korišteno je ukupno pet algoritama. Uspješnost algoritama varira međusobno, a postoje i značajne varijacije unutar samih algoritma. Najuspješniji se pokazao *Frankenstein PSO* algoritam s par preinaka u odnosu na originalni pseudokod.

U evolucijskom računarstvu nije jednostavno odabrati algoritam za rješavanje nekog problema. Prikladnost algoritma se, ovisno o problemu, može pretpostaviti na temelju prijašnjih iskustava ili na temelju poznavanja domene problema, no na kraju njegove stvarne rezultate nećemo znati sve dok ga ne ispitamo i usporedimo s drugim algoritmima.

LITERATURA

[1] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

[2] Marco Montes de Oca, Thomas Stützle, Mauro Birattari, i Marco Dorigo. Frankenstein's pso: A composite particle swarm optimization algorithm. *Evolutionary Computation, IEEE Transactions on*, 13:1120 – 1132, 11 2009. doi: 10.1109/TEVC.2009.2021465.

[3] Stjepan Picek, Domagoj Jakobovic, i Marin Golub. On the recombination operator in the real-coded genetic algorithms. U *2013 IEEE Congress on Evolutionary Computation*, stranice 3103–3110. IEEE, 2013.

[4] Marko Čupić. *Prirodom inspirirani optimizacijski algoritmi*. 2013.

Evolucijska optimizacija upravljanja električnom energijom temeljena na primjeni pametnih mreža

Sažetak

Učinkovito upravljanje električnom energijom postaje sve veći problem za moderni svijet. Pametne mreže se koriste kako bi se isplativije upravljalo električnom energijom. Pronalazak optimalnih parametara za pametnu mrežu predstavlja težak problem za klasičan matematički pristup. Ovaj rad se bavi korištenjem evolucijskih algoritama za optimizaciju parametara pametne mreže. U radu se razmatra više vrsta algoritama te se uspoređuju njihovi rezultati.

Ključne riječi: evolucijsko računarstvo, evolucijski algoritam, genetski algoritam, simulirano kaljenje, algoritam roja čestica, frankenstein PSO, upravljanje električnom energijom, pametne mreže

Evolutionary Computation in the Energy Domain: Smart Grid Applications

Abstract

It is getting increasingly difficult to efficiently manage electrical grids in a modern world. Smart grids are used to better manage generation and use of electrical energy. Finding optimal parameters for the smart grid has proved to be a difficult task for mathematical approach. This paper uses evolutionary algorithms to optimize parameters of the smart grid. Multiple algorithms are tested and compared to each other using the calculated results.

Keywords: evolutionary computing, evolutionary algorithm, genetic algorithm, simulated annealing, particle swarm optimization, frankenstein PSO, smart grid applications, smart grids