Paralelizacija metaheuristika

Seminarski rad u okviru kursa
Metodologija stručnog i naučnog rada
Matematički fakultet

Jovan Mirkov
jvnmirkov@gmail.com

Lazar Vasović
lazar1997@mts.rs

Dušica Golubović
dusica.golubovic@yahoo.com

Teodora Heldrih
heldrich013@gmail.com

20. april 2020.

Sažetak

Ukratko su opisani zadaci matematičke optimizacije i njihova moguća rešenja na primeru problema trgovačkog putnika, kao i problemi koji se susreću pri svakom od pristupa. Uz razvoj novih tehnologija rastu i mogućnosti današnjih računara, ali i potreba za bržim usavršavanjem. Kao rešenje nedovoljne brzine izvršavanja metaheuristika, predloženo je njihovo unapređenje paralelizacijom. Uz opširnu motivaciju, izloženo je nekoliko različitih modela u zavisnosti od potreba i tipa približnog metoda koji se paralelizuje. Uz svaki tip izložen je po jedan ilustrativan primer problema.

Ključne reči — optimizacija, metaheuristike, paralelizam

Sadržaj

1 Uvod

Veliki udeo realnih problema u računarstvu i drugim sferama života koji se rešavaju tehnikama optimizacije jeste NP-težak (eksponencijalan prostor pretrage), ili na neki drugi način komplikovan za tačno rešavanje – npr. prostor pretrage je polinomijalne kardinalnosti, ali bi provera svake mogućnosti zahtevala neprihvatljivo dugo vreme izvršavanja. Kao dobro rešenje pokazale su se metaheuristike, a pogotovu njihove paralelne verzije. One se danas koriste u raznim oblastima: optimizaciji, bioinformatici, razvoju softvera, pa sve do neračunarskih disciplina poput telekomunikacija i ekonomije [1, 2]. Više o optimizacionim problemima, metaheuristikama i tome zašto i kako ih je moguće ubrzati paralelizacijom, opisano je u poglavlju 2. U poglavlju 3 opisana je podela metaheuristika na metaheuristike zasnovane na populaciji jedinki i metaheuristike zasnovane na unapređenju jednog rešenja. Prvo je opisan algoritam populacijske metaheuristike (poglavlje 3.1) i njegova primena na paralelizaciju algoritama evolucije (poglavlje 3.1.1), kao i konkretna primena na paralelizaciju genetskog algoritma u okviru istog podnaslova. Paralelizacija metaheuristike zasnovane na unapređenju jednog rešenja je opisana u poglavlju 3.2.

2 Optimizacioni problemi i rešenja

Matematička optimizacija, poznata i kao **matematičko programiranje**, predstavlja odabir najboljeg elementa iz nekog skupa potencijalnih rešenja proučavanog problema prema određenom kriterijumu. Kako se odabir vrši pretraživanjem skupa rešenja, optimizacioni algoritmi spadaju u algoritme pretrage. Kriterijum pretrage najčešće je iskazan ciljnom funkcijom koju treba minimizovati ili maksimizovati (ova dva problema su ekvivalentna, pošto se maksimizacija funkcije f svodi na minimizaciju $-f$), a opciono i skupom ograničenja koji mora biti zadovoljen kako bi rešenje bilo prihvatljivo, odnosno dopustivo, pošto su kod izrazito teških problema ponekad prihvatljiva i relativno dobra nedopustiva rešenja, koja samo u maloj meri ne ispunjavaju neko od nametnutih ograničenja.

2.1 Pristupi rešavanju

Za potrebe daljeg teksta, uvodi se sledeći motivacioni primer:

Primer 2.1 *Neka je zadat skup G od n gradova i funkcija udaljenosti $d(g_i, g_j)$ za svaki par gradova. Pronaći permutaciju $p : [1..n] \mapsto [1..n]$ takvu da je ukupna suma udaljenosti grana koje se prolaze obilaskom minimalna.*

Naveden je **problem trgovačkog putnika** (eng. *travelling salesman problem*, TSP), jedan od najpoznatijih problema kombinatorne optimizacije. Verzija sa odlučivošću – postoji li put (permutacija) dužine manje od zadatog L – poznata je u teoriji složenosti kao jedan od prototipnih NP-kompletnih problema, što znači da je poznat način rešavanja isključivo u eksponencijalnom vremenu. To nadalje znači da najverovatnije nije moguće sa sigurnošću odrediti traženu permutaciju kod osnovnog problema bez provere svih $n!$ mogućnosti [3].

Ovakav raspored činilaca navodi na zaključak da je **egzaktno** rešavanje ovog i sličnih problema neisplativo. Već za dvadesetak gradova, metod grube sile poput totalne enumeracije morao bi da proveri preko dva triliona ($20! > 2 \cdot 10^{18}$) potencijalnih rešenja, pri čemu bi evaluacija svakog sabirala

desetine grana. Iz tog razloga se, u slučaju da nije od prevelike važnosti dobijanje tačnog rešenja, pribegava upotrebi **heuristika**. One daju približno rešenje, ali se izvršavaju u polinomskom vremenu. Uprkos efikasnosti dobijanja, heuristička rešenja mogu biti problematična zbog lošeg kvaliteta, tako da se često primenjuju heuristike sa određenom garancijom kvaliteta. Primera radi, za slučaj TSP u euklidskoj ravni (važi i za druge mere sa svojstvom nejednakosti trougla) osmišljena je 2-aproksimativna heuristika [3] koja dokazano pronalazi rešenje koje je u najgorem slučaju dvaput lošije (duže) od optimalnog, i to u kvadratnom vremenu [4].

Problem mnogih heuristika jeste njihova specifičnost. Jednom osmišljen algoritam, poput 2-aproksimativnog za TSP, često nije ponovo upotrebljiv, osim za neke veoma slične probleme. Stoga se za potrebe naglašavanja upotrebljivosti na širokom spektru optimizacionih problema uvodi pojam **metaheuristike**. Metaheurističke metode opisuju opšte strategije pretrage za rešavanje optimizacionih problema [5]. Formulisane su nezavisno od konkretnog problema, ali najčešće su prilično parametrizovane, pa ih je pre pokretanja moguće prilagoditi posebnom problemu koji se rešava. Poput drugih približnih metoda, metaheuristike razmatraju samo mali deo skupa mogućih rešenja, ali s druge strane uglavnom ne daju nikakve garancije o kvalitetu dobijenog rešenja. Ipak, rezultati su često dovoljno dobri i prihvatljivi, posebno u situacijama kada bi jedina alternativa bila egzaktna metoda koja bi se izvršavala neprihvatljivo dugo.

2.2 Može li bolje?

Uprkos tome što se približnim rešavanjem znatno poboljšava vremenska složenost pretrage, često ni dužina izvršavanja metaheuristike nije zadovoljavajuća. Uobičajene metaheuristike uglavnom su sekvencijalne – svaki deo algoritma izvršava se jedan za drugim. Kako to često nije nužno, kao dobar način unapređenja javlja se **paralelizacija**. Uopšteno se pod ovim pojmom podrazumeva pisanje ili prilagođavanje programa tako da koriste mogućnosti paralelnog izvršavanja [6]. Konkretno, ukoliko je raspoloživo više izvršnih jedinica u vidu procesora ili jezgara, moguće je istovremeno izvršavati pojedine delove metaheurističkih algoritama. Prema tipu podele posla između jedinica, moguće je:

- **paralelizovati zadatke** – npr. jedna nit pokreće glavni program dok druge računaju neke međukorake (konkurentnost na nivou jedinice),
- **paralelizovati podatke** – npr. svako rešenje se obrađuje i ažurira na posebnoj niti (*paralelizacija petlji*, konkurentnost na nivou naredbe).

Iz prethodne podele nazire se i drugi razlog paralelizacije, pored dobijanja na performansama. U pitanju je podela posla, odnosno razdvajanje odgovornosti, odnosno podrška logičkoj strukturi programa [7], što je možda čak i važniji motiv iz ugla razvoja softvera [6]. Treći razlog jeste rad na fizički nezavisnim uređajima – farmama (klasterima) računara, ili prosto u različitim adresnim prostorima. Na ovaj način, moguće je ne samo ubrzati postojeće metaheurističke algoritme i bolje ih modelovati, već i konstruisati nove koji maksimalno iskorišćavaju distribuiranu arhitekturu sistema na kojem se izvršavaju. U tome pomažu radni okviri kao što su MALLBA, ParadisEO, pALS, ECJ, OPT4J, DGPF i PMF, a ovako specijalizovani algoritmi često mogu biti robusniji i proizvesti bolja rešenja od prosto paralelizovanih, ali suštinski neizmenjenih sekvencijalnih [1, 2, 8].

Dve važne osobine razlikuju paralelne metaheuristike od sekvencijalnih – **dizajn** odnosno *„arhitektura“ algoritma* i **implementacija** odnosno *arhitektura sistema*. Prema prvoj karakteristici, izdvajaju se tri glavna paralelna modela koji se međusobno razlikuju po granularnosti, tj. nivou na kojem se sprovodi paralelizacija [8]. U tabeli 1 date su sheme modela i neke njihove osobine.

Tabela 1: Paralelni modeli metaheuristika

Granularnost	Zavisnost od problema	Ponašanje	Cilj
Nivo algoritma	Nezavisan	Izmenjeno	Efektivnost
Nivo iteracije	Nezavisan	Neizmenjeno	Efikasnost
Nivo rešenja	Zavisan	Neizmenjeno	Efikasnost

U **paralelnim modelima na nivou algoritma** izdvajaju se nezavisni i kooperativni pristup, zasnovani na paralelizaciji zadataka. Pri prvom, različite metaheuristike se, bez saradnje, izvršavaju istovremeno, a rešenja se na kraju kombinuju. Mogu se koristiti iste, ili različite metaheuristike, pri čemu u slučaju istih mogu biti isti, ili različiti parametri. Čak i neki metaparametri, poput kodiranja rešenja, operatora pretrage, ciljne funkcije, skupa ograničenja i kriterijuma zaustavljanja, mogu biti različiti. Pri drugom, različite jedinice u toku istovremenog izvršavanja razmenjuju informacije, čime se međusobno usmeravaju ka boljim i robusnijim rešenjima. Iz tog razloga je ponašanje algoritma izmenjeno u odnosu na sekvencijalnu verziju, mada nema zavisnosti od samog problema koji se rešava. Glavni dobitak je efektivnost metaheuristike, u vidu boljih rešenja.

Kada se govori o **paralelnim modelima na nivou iteracije**, cilja se na to što se metaheuristike izvršavaju u više sličnih koraka, uglavnom u petlji. Većinski su zasnovani na raspodeli rešenja, npr. paralelno ispitivanje više jedinki iz susedstva, ili podela populacije na podskupove, što je već pomenuta paralelizacija podataka. Ni u ovom slučaju nema zavisnosti od samog problema. Kako paralelna izračunavanja ne interaguju, ponašanje algoritma je neizmenjeno u odnosu na sekvencijalnu verziju. Glavni dobitak je efikasnost, u vidu bržeg rada.

Paralelni modeli na nivou rešenja umnogome su slični onima na nivou iteracije. U cilju dobijanja na efikasnosti, paralelizuju se operacije koje se izvršavaju nad svakim potencijalnim rešenjem. Ponašanje je takođe neizmenjeno u odnosu na sekvencijalnu verziju i zasnovano na paralelizaciji podataka. Ipak, postoji razlika u tome što se paralelizuju koraci algoritma specifični za problem koji se rešava, kao što je izračunavanje zahtevne funkcije cilja za svako rešenje.

Poznata je i upotreba sva tri predstavljena modela u spoju, u okviru hijerarhijski strukturiranih hibridnih algoritama. U njima se na nivou algoritma paralelno izvršavaju metaheuristike zasnovane na paralelnom modelu na nivou iteracije. Svaka paralelizovana iteracija nadalje je modelovana tako da primenjuje paralelizaciju na nivou rešenja. Glavna osobina ovakvog hijerarhijskog modela jeste natprosečno visok nivo paralelizma, kao i vrlo visoka skalabilnost [8].

Problem implementacije, kao druga važna osobina koja razlikuje paralelne metaheuristike od sekvencijalnih, tesno je vezan za arhitekturu sistema na kojem se algoritam izvršava. Rešenja ovog problema odgovaraju

na pitanje kako efikasno preslikati paralelni model algoritma na dostupnu paralelnu arhitekturu [2, 8]. Kako konkretne arhitekture i realizacije nisu tema ovog mahom teorijskog rada, stvar će biti samo ukratko izložena. Savremeni pristup deli paralelne arhitekture na tri klase – **programabilna kola**, **višeprocesorske sisteme** i **distribuirane platforme**. Predstavnik prve klase su FPGA kola (eng. *field-programmable gate array*). Drugu čine klasični višejezgarni sistemi (višenitno programiranje), kao i arhitektura GPGPU (eng. *general-purpose computing on graphics processing units*), koja potencira na podeli rada između centralne (CPU) i grafičke procesorske jedinice (GPU). Treću čine već pomenute farme (klasteri) računara, P2P računarstvo (eng. *peer-to-peer* – čvor do čvora), mrežno računarstvo (eng. *grid computing*), kao i računarstvo u oblaku (eng. *cloud computing*). Svaka od nabrojanih arhitektura uspešno se koristi za realizaciju paralelnih metaheuristika, doduše u različitoj meri i sa različitim domenima primene [1, 8].

3 Podela paralelnih metaheuristika

Postoji veliki broj metaheurističkih pristupa, ali se po osnovnoj ideji izdvajaju dva – **metaheuristike zasnovane na populaciji jedinki** ili *P-metaheuristike* (od eng. *population*) i **metaheuristike zasnovane na unapređenju jednog rešenja** ili *S-metaheuristike* (od eng. *single [solution]*). Prvi pristup se sastoji iz generisanja početne populacije rešenja, bilo na slučajan način ili nekim drugim algoritmom, koja se zatim iterativno unapređuje. Na kraju svake iteracije, cela populacija ili njen deo zamenjuju se novim jedinkama nastalim na osnovu starih. Kao konačno rešenje problema odabira se najbolja jedinka iz populacije s kraja rada algoritma. Kako kroz vreme otkrivaju nove oblasti u prostoru pretrage, populacijske metode su mahom *eksplorativne*. Drugi pristup započinje jednim početnim rešenjem koje se u svakom koraku pretrage uglavnom poboljšava zamenom sa nekim boljim obližnjim rešenjem. Može se reći da se jedinka kreće kroz prostor, pa se ove tehnike nazivaju i **metaheuristike zasnovane na putanji**. Kao konačno rešenje uzima se tekuća jedinka s kraja rada algoritma. Kako se u ovom slučaju brzo nalazi lokalni optimum pretragom susedstva, putanjske metode su mahom *eksploatativne*. Uobičajeno je kombinovanje (hibridizacija) ova dva pristupa, pri čemu se prvo vrši istraživanje (eksploracija) celokupnog prostora pretrage, a zatim detaljnije ispitivanje (eksploatacija) onih oblasti koje obećavaju [9].

3.1 Pristupi zasnovani na populaciji

Kao što je već rečeno, P-metaheuristike počinju od inicijalne populacije rešenja. Dalje, izdvajaju se dve glavne faze ovog algoritma koje se iterativno primenjuju: [8]

- generisanje nove populacije,
- zamena trenutne populacije.

U fazi generisanja kreira se nova populacija rešenja, dok se u fazi zamene trenutne populacije vrši odabir jedinki iz trenutne i novokreirane populacije. Funkcija evaluacije dodeljuje vrednost prilagođenosti, kojom se za svaku jedinku u populaciji određuje kvalitet dodeljenog rešenja. Ovaj proces se ponavlja sve dok nije ispunjen kriterijum zaustavljanja. U nastavku je dat pseudokod ovog algoritma [8].

Većina P-metaheuristika su algoritmi koji se zasnivaju na prirodnim procesima. Svi oni se međusobno razlikuju po izvođenju faza generisanja i selekcije kao i po postojanju memorije u njihovoj pretrazi [8].

Algoritam 1: Algoritam populacije

```
1 P := P_0;
2 t := 0;
3 while not kriterijumZaustavljanja(P_t) do
4 |   Generisati(P_t');
5 |   P_{t+1} := Selekcija(P_t ∪ P_t');
6 |   t := t + 1;
7 end
```

3.1.1 Paralelizacija algoritama evolucije

Algoritam 2: EA pseudokod

```
1 P := P_0;
2 t := 0;
3 while not kriterijumZaustavljanja(P_t) do
4 |   Evaluacija(P_t);
5 |   P_t' := Selekcija(P_t);
6 |   P_t' := operatorReprodukcije(P_t');
7 |   P_{t+1} := Zameni(P_t, P_t');
8 |   t := t + 1;
9 end
```

Algoritmi evolucije (eng. *evolutionary algorithms*), u daljem tekstu EA, jedinke u populaciji biraju koristeći operatore varijacije (mutacija, rekombinacija) koji se primenjuju *direktno* na jedinke. U gorenavedenom pseudokodu dat je opšti algoritam svakog EA. U zavisnosti od reprezentacije jedinke i koraka evolucije, postoji nekoliko potklasa EA, među kojima su evolutivno programiranje (EP), genetski algoritmi (GA) i evolutivne strategije (ES) [1].

Za netrivijalne probleme, izvršavanje reprodukcije jednostavnog EA nad velikom populacijom zahteva ogromne računarske resurse. Samo računanje funkcije prilagođavanja za svaku jedinku u populaciji je najskuplja operacija. Ovi problemi se rešavaju primenom paralelnog modela [1].

Kada je u pitanju bilo koji algoritam zasnovan na populaciji, paralelizacija se javlja prirodno kao posledica činjenice da je svaka jedinka koja joj pripada nezavisna jedinica. Kao rezultat toga, efikasnost algoritma se drastično poboljšava kada se on izvršava **paralelno**. Dve glavne strategije paralelizacije su:

- **paralelizacija populacije** – populacija se deli na disjunktne delove koji evoluiraju odvojeno i kasnije mogu da se spoje,
- **paralelizacija operacija** – operacije koje se najčešće primenuju na jedinke se izvršavaju paralelno [1].

U ranim fazama paralelizacije ovih algoritama model koji se najčešće koristio je **gospodar-sluga** (eng. *master-slave*) model, takođe poznat i kao **globalna paralelizacija**. Ona podrazumeva da jedan procesor, master procesor, preuzima izvršavanje operacija selekcije, dok sluge procesori izvršavaju rekombinaciju, mutaciju i evaluaciju funkcije prilagođenosti. Ovaj algoritam je isti kao i sekvencijalni, osim što je brži, posebno za one funkcije koje su vremenski zahtevne. Obično ostale delove algoritma nema potrebe paralelizovati, osim ako se radi o populaciji koja je **struktuirana**. Moguće je i korišćenje i više procesora, kako bi se ubrzalo sekvencijalno izvršavanje algoritma. Ako je ovo slučaj, nema interakcije između nezavisnih izvršavanja. Međutim, većina paralelnih evolucijskih algoritama koristi neku vrstu prostornog rasporeda za jedinke, *struktuiranu populaciju*, u formi skupova ostrva ili u vidu difuzne mreže. Kao posledica ovoga, većina operacija varijacije može lako da se paralelizuje. U nekim slučajevima, čak i bez upotrebe paralelnih mašina za pokretanje ovog algoritma, dobijaju se bolji rezultati nego kod primene tradicionalnih algoritama [1].

Od svih mogućih tipova struktuiranih populacija, dve najpoznatije optimizacione procedure su svakako:

- **distribuirani** (eng. *distributed*) ili grubo granulirani,
- **ćelijski** (eng. *cellular*) ili fino granulirani.

Kod **distribuiranih** algoritama, populacija se deli na skupove *ostrva* nad kojima se nezavisno izvršava algoritam evolucije. Ova nezavisnost omogućava odvojeno izračunavanje funkcije prilagođenosti, ukrštanje i slično na svakom ostrvu. Razmene jedinki između ostrva nisu česte, i to sve sa ciljem uvođenja različitosti u subpopulaciju, a samim tim i izbegavanja upadanja u lokalni optimum. Dodatni parametri koji su potrebni se odnose na to koliko su česte migracije jedinki, kao i kako se biraju jedinke koji migriraju i ka kom ostrvu (potpopulaciji) [1].

Kada su u pitanju **ćelijski** algoritmi, uvodi se koncept *susedstva*, tako da jedinke mogu da imaju interakciju samo sa njima najbližim jedinkama. U isto vreme jedna jedinka ne pripada samo jednom susedstvu, stvarajući mrežu dobrih rešenja. Ovaj koncept omogućava bolje istraživanje prostora rešenja zbog njegove spore difuzije (širenja), dok unutar svakog susedstva dolazi do eksploatacije. Ovaj alogitam se može implementirati koristeći MIMD[1] računar sa distribuiranom memorijom, mada je na SIMD[2] računarima ta implementacija direktnija [1].

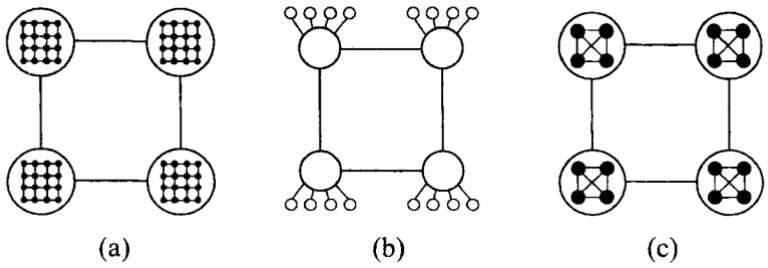
Pored ćelijskih i distribuiranih, rasprostranjeni su i **hibridni** modeli (slika 1). Oni podrazumevaju upotrebu paralelizacije u dva nivoa. Na višim nivoima se upotrebljava distribuirani algoritam, dok se na svakom ostrvu upotrebljava ćelijski algoritam, gospodar-sluga model, pa čak i još jedna distribuirana metoda [1].

U nastavku teksta će detaljnije biti opisani paralelni modeli algoritama evolucije.

1. **Model sa nezavisnim pokretanjima**:
   Ovaj model podrazumeva paralelno pokretanje istog sekvencijalnog algoritma bez interakcija između nezavisnih pokretanja. Ovaj krajnje jednostavan simultani metod je veoma koristan. Na primer, kada

[1] MIMD (eng. *Multiple Instruction, Multiple Data Stream*) arhitektura se sastoji od nekoliko stotina ili hiljada procesora sa malom memorijom. Različiti podati mogu biti učitani u memoriju svakog od procesora i oni mogu izvršavati različite instrukcije

[2] SIMD (eng. *Single Instruction, Multiple Data Stream*) arhitektura je sastavljena od stotinu ili pak hiljadu jednostavnih procesora, svaki sa malom memorijom. Za ovu arhitekturu je karakteristično izvršavanje iste instrukcije na svakom procesoru



Slika 1: Primer nekih hibridnih modela: na baznom nivou su (a) ćelijski, (b) gospodar-sluga, (c) distribuirani model [1]

se želi više pokretanja algoritma za isti problem sa različitim inicijalnim vrednostima, prikupljajući na taj način statistike o problemu. Pošto je sama priroda genetskih algoritama stohastička, postojanje ovakvih statistika se smatra značajnim [1].
Može se smatrati da je ovaj model specijalni slučaj distribuiranog modela gde ne postoje migracije. Ovako, rezultat distribuiranog izračunavanja je najbolje moguće rešenje iz nezavisnih izvršavanja [1].

2. **Gospodar-sluga model**:
Kod ovog modela master procesor izvršava glavnu petlju genetskog algoritma i šalje potrebne parametre slugama, koji koristeći te parametre izračunavaju funkciju evaluacije. Vrednosti ovih funkcija se potom vraćaju master procesoru [1]. Pokazalo se da je ovaj model veoma efikasan kada je izračunavanje fitnesa veoma skupa operacija (u poređenju sa cenom komunikacije između master i sluga procesora).

3. **Distribuirani model**:
Nad svakom potpopulacijom se nezavisno izvršava genetski algoritam. Iako povremeno dolazi do migracija jedinki iz jedne subpopulacije u drugu, one su veći deo vremena izvršavanja međusobno izolovane. Distribuirani model zahteva postojanje *planova migracija* koji uključuju postojanje sledećih parametara [1]:

   - *razmak između migracija* – broj koraka u evoluciji u svakoj potpopulaciji između dve razmene, Migracije mogu da se događaju ili periodično ili sa određenom, unapred zadatom verovatnoćom P_M, za odluku u svakom koraku da li dolazi do migracije, ili ne,
   - *stopa migracije* – koji broj jedinki će učestvovati u migraciji; može se izraziti u procentima celokupne populacije, ili kao apsolutna vrednost,
   - *odabir/zamena jedinki* – kojim se biraju pojedinačne jedinke koje će biti zamenjene, kao i jedinke kojim se menjaju,
   - *topologija* – definiše suseda za svaku subpopulaciju. Subpopulacija može svojim susedima da šalje, kao i da preuzima pojedinačne jedinke u procesu migracije.

Svakodnevni razvoj mrežnih infrastruktura, softvera, internet servisa, kao i neprestano širenje mreža dovelo je do potrebe za novim algoritmima koji će zameniti tradicionalne koji su neefikasni kad su u pitanju problemi u industriji. Ta potreba za bržim rešenjima zadovoljena je upotrebnom

metaheuristika u oblasti telekomunikacija, kao i paralelizacije, kojom bi te metaheuristike dodatno bile ubrzane. Jedan od glavnih problema u telekomunikacijama je problem Štajnerovog drveta (eng. *Steiner tree problem*, STP), koji podrazumeva pronalaženje podstabla minimalne težine datog grafa uspostavljenih terminalnih čvorova. U radu [10] predložen je gospodar-sluga paralelni GA kao konkretno rešenje ovog problema. Nakon toga je u radu [11] ovaj predlog proširen predstavljanjem hibridnog paralelnog modela koji kombinuje distribuirani GA i lokalnu pretragu koristeći specifične STP heuristike. Analiza efikasnosti pokazuje da distribuirani model postiže bolje vrednosti ubrzanja od gospodar-sluga modela jer koristi nekoliko tačaka sinhronizacije i na taj način može da se izvršava u okruženju računarskih mreža širokog pojasa [1].

3.2 Pristupi zasnovani na unapređenju rešenja

Osnovna ideja metaheuristika zasnovanih na unapređenju je započinjanje od jednog inicijalnog rešenja, koje se u većini koraka pretrage zamenjuje drugim rešenjem iz okruženja. Ovakve tehnike omogućavaju da se lokalna optimalna rešenja nađu brzo, ali stvaraju problem ako se zahteva globalno optimalno rešenje. Važno je napomenuti da tokom razmatranja kvaliteta u cilju zamene rešenja, na izbor može uticati i konkretna metoda koja je odabrana. Na primeru metode simuliranog kaljenja, ako su određeni kriterijumi ove konkretne metode zadovoljeni, može se izabrati novo rešenje čija vrednost funkcije cilja nije bolja od vrednosti trenutne. Ovo takođe znači da nije izabrano rešenje koje ima najbolju vrednost funkcije cilja do koje se može doći iz trenutnog rešenja, čime se smanjuje verovatnoća konvergencije rešenja ka lokalnom optimumu koji nije i globalni [8].

Ovakve metode se nazivaju pretraživačko-orijentisane metode (eng. *exploration-based*), jer se zalažu za pretraživanje prostora rešenja. Metaheuristike za rešavanje optimizacionih problema se mogu posmatrati kao kretanje kroz prostor rešenja, prateći puteve nekog metoda pretrage.

Većina metaheuristika su sekvencijalne. Same po sebi imaju cilj da smanje vremensku složenost, ali uvek je moguće pribeći paralelizmu za još brža rešenja, mada paralelizam više doprinosi kvalitetu rešenja u ovom slučaju [2]. U najpoznatije metode koje se zasnivaju na unapređenju jednog resenja, uz neke manje poznate poput pohlepne slučajne adaptivne pretrage (eng. *greedy randomized adaptive search*, GRASP), spadaju:

- simulirano kaljenje,
- tabu pretraga,
- iterativna lokalna pretraga,
- metod promenljivih okolina.

U daljem tekstu je dat opšti algoritam za ove metode. Promenljiva 's' predstavlja niz izabranih rešenja određenih pretragom. Pre petlje se inicijalizuje početno rešenje metodom 'Generisi' i promenljiva 't', koja broji korake kretanja. U petlji, koja se izvršava sve dok nije ispunjen kriterijum zaustavljanja pretrage, funkcijama 'IzaberiKorak', 'PrihvatiKorak' i 'PrimeniKorak' istražuje se prostor rešenja i prihvataju ona koja zadovoljavaju kriterijum vezan za konkretnu metodu koja se koristi.

Pretraga je iterativna i omogućava kretanje od jednog do drugog rešenja u prostoru rešenja. Pretraga počinje od jednog nasumičnog rešenja ili nekog dobijenog uz pomoć nekog drugog optimizacionog algoritma. U svakoj iteraciji trenutno rešenje je zamenjeno nekim drugim iz skupa susednih kandidata. Pretraga prestaje kad je dati uslov zadovoljen, kao npr.

Algoritam 3: Sekvencijalni algoritam metaheuristike zasnovane na unapređenju jednog rešenja

```
1 Generisi(s(0));
2 t := 0;
3 while not kriterijumZaustavljanja(s(t)) do
4 |   s'(t) := IzaberiKorak(s(t));
5 |   if PrihvatiKorak(s'(t)) then
6 |   |   s(t) := PrimeniKorak(s'(t));
7 |   end
8 |   t := t + 1;
9 end
```

maksimalni broj pomeranja, nađeno rešenje je zadovoljavajućeg kvaliteta, algoritam ne pravi znatna poboljšanja tokom nekog određenog vremena [2].

Paralelizacija ovih algoritama se prvi put spominje u naučnim krugovima 1980. godine. Jačine procesora u to vreme su bile slabe tako da su gorenavedeni algoritmi u serijskom izvršavanju bili vremenski zahtevni čak i za manje probleme. Vremenom su procesori postajali sve jači, što je uticalo na rast zainteresovanosti i istraživanja ove oblasti. Razni nezavisni autori su davali slične predloge klasifikacije modela paralelizacije ovih algoritama, između kojih postoji mnogo preklapanja u ključnim idejama modela. Na slici 2 je dat vizuelni prikaz klasifikacije modela koje je predložio Enrike Alba, koji se najviše istakao u istraživanju ove teme [1].

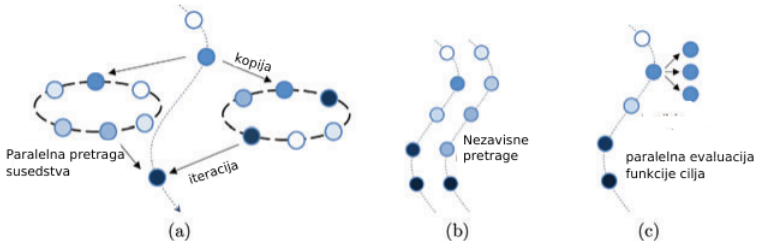
Najčešće predlagani modeli paralelizacije su:

1. **Model paralelizma podataka**:
   Ideja ovog modela je podela podataka na delove između procesorskih jedinica. Samim tim je i prostor rešenja podeljen. Preduslov je da su podaci (prostor rešenja) razdvojivi.
2. **Model paralelizma kretanja**:
   Na početku svake iteracije, program kopira trenutno rešenje između procesorskih jedinica. Informacije o vrednostima iz okoline se prikupljaju i prema njima se bira sledeći kandidat/rešenje. Ovaj model ne utiče na rešenje jer bi sekvencijalno izvršavanje dalo iste rezultate samo u dužem vremenskom periodu.
3. **Model paralelizma višestrukog pokretanja**:
   Najpopularnija metoda zbog lakoće korišćenja. Algoritam se pokreće više puta istovremeno na različitim procesorskim jedinicama, sa različitim inicijalnim rešenjima, ili sa istim inicijalnim rešenjima, ali sa drugim parametrima. Ova metoda doprinosi boljem pokrivanju prostora rešenja. Jedan od benefita ove metode je to što se može obezbediti međusobna komunikacija između pokrenutih algoritama tokom njihovog izvršavanja, korišćenjem memorije koje će deliti, i time pomoći drugim pokrenutim jedinicama ako su pre njih istražili neki deo prostora.
4. **Model ubrzanog kretanja**:
   Na početku svake iteracije, program koristi paralelizaciju za izračunavanje funkcija cilja. Ovo je korisno ako funkcija može takođe biti paralelizovana pa se funkcija kvaliteta rešenja koji se posmatra u trenutnoj iteraciji može posmatrati kao agregacija parcijalnih funkcija.

5. **Hibridni model**:
   Kombinovanje datih algoritama i modela paralelizma sa evolutivnim algoritmima koji su inherentno paralelni. Najčešće se koristi u kombinaciji sa modelom paralelizma višestrukog pokretanja gde se nad svakom jedinkom iz evolutivnog algoritma pokrene algoritam zasnovan na unapređenju jednog rešenja na odvojenim procesorskim jedinicama.

Pokazalo se da je paralelizacija ovih metaheuristika veoma korisna kada je u pitanju rešavanje problema u oblasti telekomunikacija. Pouzdanost u telekomunikacijama odnosi se na mogućnost mreže da nesmetano funkcioniše kada dođe do kvara na nekom čvoru ili vezi. Pristup za povećavanje pouzdanosti u telekomunikacijskoj mreži sastoji se u ograničavanju broja grana na putanji između dva čvora – postoji unapred uspostavljen skup čvorova i putevi između njih imaju najviše k grana.

Autori Riberio i Rozeti [12] razvili su paralelni GRASP algoritam za ovaj problem kada je k = 2. Faza konstrukcije GRASP-a koristi iterativni algoritam za određivanje najkraćeg puta u grafu, i to nasumičnim odabirom početnih i krajnjih čvorova sve dok svaka kombinacija parova nije razmotrena. Faza lokalne pretrage pokušava da poboljša rešenja eliminišući neke puteve i preračunavajući puteve koristeći modifikovane težine grana. Paralelni pristup koristi strategiju višestruke nezavisne pretrage dok se iteracije distribuirano izvršavaju na procesorima. Paralelizam omogućava linearno ubrzanje i dovodi do dostizanja većeg kvaliteta rešenja [1].



Slika 2: Modeli paralelizma prema Enrikeu Albi: (a) kretanje, (b) višestruko pokretanje, (c) ubrzano kretanje [2]

4 Zaključak

Kako je i izloženo, paralelizacija je jedan od načina da se ubrza izvršavanje metaheuristika, a nekad i da se poboljša kvalitet rešenja. Paralelizacija ima i potencijal da se prilagodi problemima sa većim dimenzijama, kao i brojnim ograničenjima. Ipak, nije uvek sigurno da će paralelizacija, kao ni metaheuristika dati najbolje ili najbrže rešenje. Uz to, stalni razvoj tehnologije čini da izgleda kao da je potrebno mnogo znanja za pribegavanje parelelizaciji. Zato je, još uvek, ova oblast veliki izazov.

Literatura

[1] Enrique Alba. *Parallel Metaheuristics: A New Class of Algorithms*. Wiley Publishing, 2005.

[2] Enrique Alba, Gabriel Luque, and Sergio Nesmachnow. Parallel metaheuristics: Recent advances and new trends. *International Transactions in Operational Research*, 20:1–48, 01 2012. dostupno na: https://www.researchgate.net/publication/236189741_Parallel_Metaheuristics_Recent_Advances_and_New_Trends.

[3] Aleksandar Kartelj. Računarska inteligencija – optimizacija, 2019. dostupno na: http://poincare.matf.bg.ac.rs/~kartelj/nastava/RI2019/04.Optimizacija.ukratko.pdf.

[4] Thomas Sauerwald. Advanced Algorithms – VI. Approximation Algorithms: Travelling Salesman Problem, 2016. dostupno na: https://www.cl.cam.ac.uk/teaching/1516/AdvAlgo/tsp.pdf.

[5] Predrag Janičić and Mladen Nikolić. *Veštačka inteligencija*. Matematički fakultet, Beograd, 2020. dostupno na: http://poincare.matf.bg.ac.rs/~janicic/courses/vi.pdf.

[6] Saša Malkov. Razvoj softvera – Konkurentnost, 2019. dostupno na: http://poincare.matf.bg.ac.rs/~smalkov/files/rs.r290.2019/public/Predavanja/Razvojsoftvera.10.2019-Konkurentnost.pdf.

[7] Milena Vujošević Janičić. Programske paragidme – konkurentno programiranje, 2019. dostupno na: http://www.programskijezici.matf.bg.ac.rs/ppI/2019/predavanja/kp/konkurentno_programiranje_slajdovi.pdf.

[8] El-Ghazali Talbi. *Metaheuristics: From Design to Implementation*. Wiley Publishing, 2009. dostupno na: http://ie.sharif.edu/~so/Metaheuristics.pdf.

[9] Enrique Alba, Christian Blum, Pedro Isasi, Coromoto Leon, and Juan Gomez. *Optimization Techniques for Solving Complex Problems*. Wiley Publishing, 2009.

[10] Giuseppe Fatta, Giuseppe Presto, and Giuseppe Lo Re. A parallel genetic algorithm for the steiner problem in networks. 10 2011.

[11] Simone Martins, Mauricio Resende, Caio Ribeiro, and P. Pardalos. A parallel grasp for the steiner tree problem in graphs using a hybrid local search strategy." journal of global optimization 17, 267-283. *Journal of Global Optimization*, 17:267–283, 01 2000.

[12] Celso Ribeiro and Isabel Rosseti. A parallel grasp heuristic for the 2-path network design problem. pages 922–926, 01 2002.