SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

ZAVRŠNI ZADATAK br. 954

# PARALELIZACIJA EVOLUCIJSKIH ALGORITAMA

Maja Legac

Zagreb, lipanj 2009.

# Sadržaj

# 1. Uvod

U posljednjih petnaestak godina zabilježen je značajan razvoj genetskih algoritama. Genetski algoritam je recept koji kazuje što treba raditi s genetskim materijalom kako bi se s određenom vjerojatnošću nakon određenog vremena postiglo zadovoljavajuće rješenje zadanog optimizacijskog problema. Genetski materijal je skup svojstava koji opisuju neku jedinku[2]. Genetski algoritam je heuristička metoda optimiranja koja imitira prirodni evolucijski proces[1]. Analogija evolucije kao prirodnog procesa i genetskog algoritma kao metode optimiranja očituje se u procesu selekcije i genetskim operatorima.

Paralelno programiranje je razvoj paralelnog algoritma koji se izvodi na više računala ili na mreži. Evolucija u prirodi je paralelni proces. Genetski algoritam je svojevrsna apstrakcija prirodne evolucije i također ga je lako paralelizirati[2]. U zadnjih dvadesetak godina intenzivno se razmatraju mogućnosti paraleliziranja genetskih algoritama. Paralelni genetski algoritmi koriste se za rješavanje težih optimizacijskih problema.

Genetski algoritam je vremenski zahtjevan i troši puno procesorskog vremena za razliku od drugih metoda optimiranja. Cilj paralelizacije je skratiti trajanje izvođenja složenih primjenskih programa, a da se pritom ne naruše njihova svojstva. Osnovna ideja paraleliziranja programa jest raščlanjivanje sekvencijskog (slijednog) programa na nezavisne podzadatke koji se mogu izvoditi paralelno. Idealno, genetski algoritam bi trebao trajati onoliko puta kraće koliko računalo ima procesora. Naravno, u stvarnosti se postižu ubrzanja koja su manja od broja procesora.

U ovom radu baviti ćemo se raspodijeljenim genetskim algoritmima (*distributed genetic algorithm, DGA*). Osnovno obilježje raspodijeljenog genetskog algoritma je raspodijeljena populacija. Umjesto da djeluje kao sekvencijski genetski algoritam nad jednom velikom populacijom, raspodijeljeni genetski algoritam je raspodijeljen i djeluje nad više manjih populacija ili otoka.

Alat koji će se koristiti za paralelizaciju je *Message Passing Interface* (MPI).

# 2. Paralelni genetski algoritmi

## 2. 1. Podjela paralelnih genetskih algoritama

Evolucija u prirodi je paralelni proces. Genetski algoritam je svojevrsna apstrakcija prirodne evolucije i također ga je lako paralelizirati. U zadnjih dvadesetak godina intenzivno se razmatraju mogućnosti paraleliziranja genetskih algoritama. Ideja o izgradnji specifičnih paralelnih računala je još starija: još pedesetih godina John Holland predlaže specifičnu arhitekturu paralelnih računala koja bi bila pogodna za simulaciju evolucije prirodnih vrsta. Paralelni genetski algoritmi koriste se za rješavanje težih optimizacijskih problema. Teži problemi zahtijevaju velike populacije i velike duljine kromosoma, zbog čega postupak optimiranja duže traje. Osnovna motivacija paralelizacije genetskih algoritama je ubrzati njihovo izvođenje na višeprocesorskim računalima ili na više umreženih računala.

Cilj paralelizacije je skratiti trajanje izvođenja složenih primjenskih programa, a da se pritom ne naruše njihova svojstva. Osnovna ideja paraleliziranja programa jest raščlanjivanje sekvencijskog (serijskog) programa na nezavisne pod zadatke koji se mogu izvoditi paralelno. Kod genetskog algoritma treba odrediti što će se obavljati paralelno. Genetski algoritam ciklički ponavlja jedan te isti posao: iz iteracije u iteraciju izračunavaju se vrijednosti funkcije cilja i genetski operatori djeluju nad populacijom jedinki. Nameću se dva pristupa paraleliziranja genetskih algoritama:

- *Standardni pristup* – paralelizirati genetske operatore i izračunavati vrijednosti funkcije cilja paralelno
- *Dekompozicijski pristup* – podijeliti populaciju na manje dijelove – subpopulacije i izvršavati cijeli genetski algoritam nad subpopulacijama.

U prvom slučaju se paralelizira samo posao evaluacije ili genetskih operatora. *Evaluacija* ili *vrednovanje* je postupak izračunavanja vrijednosti funkcije cilja. Jednako kao i kod sekvencijalnog genetskog algoritma, genetski operatori djeluju

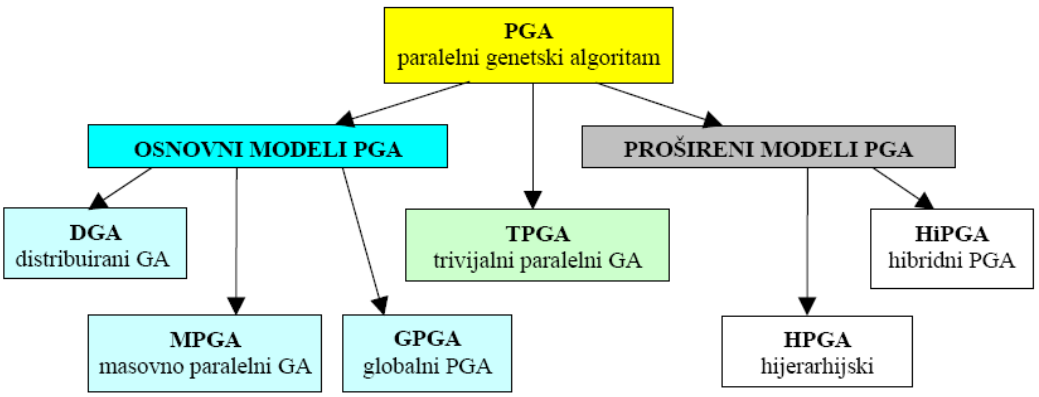
samo nad jednom, zajedničkom populacijom pa se takav model naziva jednopopulacijski model. U drugom slučaju se populacija dijeli na nekoliko subpopulacija pa se takav model naziva višepopulacijski model, odnosno genetski algoritam se naziva multipopulacijskim paralelnim genetskim algoritmom [2]. Očekuje se skraćenje trajanja izvođenja multipopulacijskog paralelnog genetskog algoritma (PGA) jer subpopulacije broje manje jedinki od inicijalne populacije pri jednopopulacijskom modelu.

Postoje nekoliko mogućih razina paraleliziranja genetskih algoritama: paraleliziranje na razini populacije, na razini jedinki te na razini evaluacije. Prema razini paralelizacije, postoje tri osnovna načina podjele sekvencijskog genetskog algoritma na podzadatke:

- *Krupnozrnata podjela* - podjela velike populacije na manje dijelove subpopulacije. U ovom se slučaju radi o dekompozicijskom pristupu ili o već spomenutom višepopulacijskom paralelnom genetskom algoritmu koji je raspodijeljen tako da se paralelno izvodi nekoliko genetskih algoritama nad manjim populacijama.
- *Sitnozrnata podjela* - ekstremni oblik podjele velike populacije na subpopulacije veličine jedne jedinke. Svaki procesor obavlja genetske operatore nad njemu dodijeljenom jedinkom i nad susjednim jedinkama. Ovakva podjela je također predstavnik višepopulacijskog modela.
- *Gospodar - sluga* – evaluacija tj. izračunavanje vrijednosti funkcije cilja se obavlja paralelno dok se genetski operatori izvršavaju sekvencijski. Gospodar obavlja genetski algoritam nad zajedničkom populacijom, stoga se to naziva jednopopulacijski model. U svakoj iteraciji sluge paralelno izračunavaju vrijednost funkcije cilja, nakon što gospodar obavi svoj dio posla.

U ovisnosti o načinu podjele genetskog algoritma na podzadatke, tri su osnovna modela paralelnih genetskih algoritma (slika 2.1.) :

- *Raspodijeljeni genetski algoritam (DGA)* – sastoji se prema krupnozrnatoj podjeli od nekoliko subpopulacija, pa se naziva još višepopulacijski genetski algoritam. To je najpopularniji model paralelnih genetskih algoritama.
- *Masovno paralelni genetski algoritam (MPGA)* – se prema sitnozrnatoj podjeli sastoji od $N_p$ procesora koji predstavlja $N_p$ jedinki. Svaki procesor obavlja genetske operatore nad svojom i njoj susjednim jedinkama.
- *Globalni paralelni genetski algoritam (GPGA)* – je predstavnik podjele na gospodara i slugu. Paralelni dio posla obavljaju sluge. Sekvencijski dio posla obavlja gospodar. Sluge izračunavaju vrijednost funkcije cilja, a gospodar izvršava se ostale genetske operatore.




**Slika 2.1. Podjela PGA**

Važno je napomenuti da samo globalni paralelni genetski algoritam ima ista svojstva kao i sekvencijski genetski algoritam, pa su sva teorijska razmatranja vezana uz sekvencijski genetski algoritam primjenjiva i na globalni paralelni genetski algoritam. Ostali modeli značajno mijenjaju način izvođenja genetskog algoritma.

Spomenuta tri osnovna modela se mogu međusobno kombinirati ili nadograditi nekom drugom metodom optimiranja. Prema tome, postoje još dva proširena modela paralelnih genetskih algoritama:

- *Hijerarhijski paralelni genetski algoritam* (HPGA) je kombinacija prethodna tri modela. Primjerice, to može biti raspodijeljeni genetski algoritam na nekoliko međusobno povezanih računala, a na svakom od računala obavlja se globalni paralelni genetski algoritam nad subpopulacijama.
- *Hibridni paralelni genetski algoritam* (HyPGA) je kombinacija jednog od prethodna četiri modela s nekim drugim algoritmom za lokalno pretraživanje. Najčešće se radi o nekoj od gradijentnih metoda koje se primjenjuju nakon određenog broja iteracija i to samo nad nekim jedinkama.

Posljednji, ali i najjednostavniji model paralelnih genetskih algoritama je:

- *Trivijalni paralelni genetski algoritam* (TPGA). Radi se o više genetskih algoritama koji se paralelno obavljaju na nekoliko potpuno nezavisnih računala (računala ne moraju biti povezana) kako bi se, primjerice, statistički obradili eksperimentalno dobiveni rezultati ili kako bi se odredio optimalan skup parametara [2]. Ova ekstremno jednostavna paralelna metoda je, u stvari, izuzetno korisna upravo za statističku analizu algoritma. Primjerice, isti algoritam pokrene se na nekoliko odvojenih računala s različitim početnim uvjetima i promatra se kvaliteta dobivenog rješenja. Potom se rezultati statistički obrađuju. S obzirom da je genetski algoritam stohastički proces, prikupljanje statističkih podataka je od izuzetne važnosti [2].

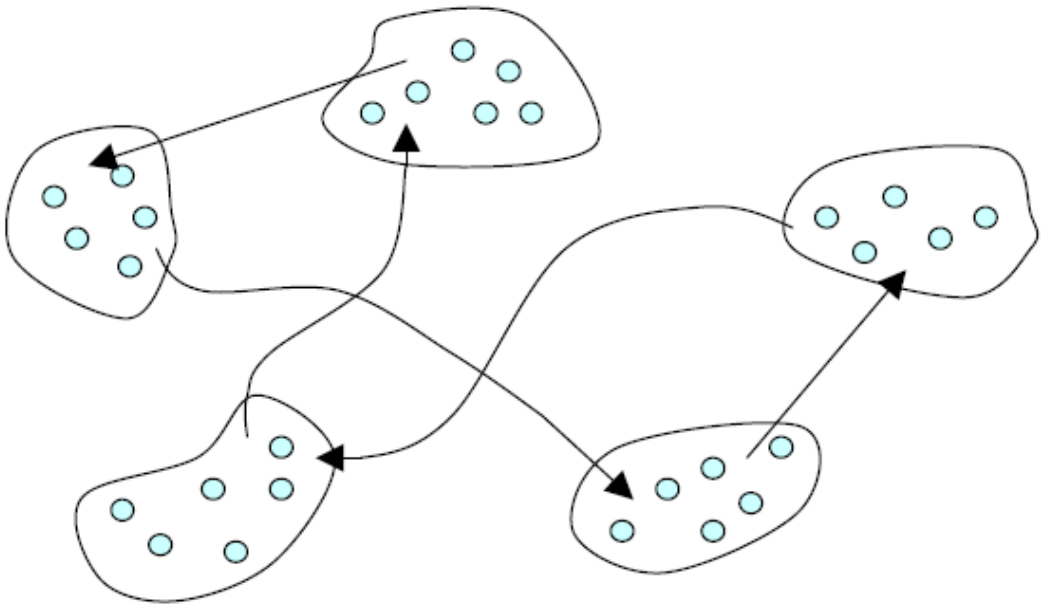
## 2. 2. Raspodijeljeni genetski algoritmi

Osnovno obilježje DGA je raspodijeljena populacija. Umjesto da djeluje kao sekvencijski genetski algoritam nad jednom velikom populacijom, raspodijeljeni genetski algoritam je raspodijeljen i djeluje nad više manjih populacija ili otoka. Model DGA je nastao uslijed potrebe da se umrežena računala iskoriste za paralelno obavljanje genetskih algoritama. Genetski algoritmi se obavljaju u *čvorovima*. Čvorovi su najčešće umrežena računala, ali mogu biti i procesori u višeprocesorskom sustavu koji međusobno komuniciraju. Genetski algoritmi u čvorovima mogu se međusobno razlikovati. Zajednički im je optimizacijski problem koji svi paralelno rješavaju. Čvorovi međusobno razmjenjuju jedinke u nadi da će novo pristigla jedinka u novoj okolini potaknuti pretraživanje još neistraženog područja prostora rješenja i na taj način postići još bolje rješenje. Više genetskih algoritama paralelno djeluju nad subpopulacijama i svakih $M_i$ iteracija, gdje je $M_i$ migracijski interval, izmjenjuju dobivena rješenja (slika 2.2). Izmjenjivanje jedinki naziva se migracijom. Subpopulacije komuniciraju (izmjenjuju podatke, tj. jedinke ili migrante) preko komunikacijskog kanala. Subpopulacije su relativno izolirane kako bi genetski algoritam pretraživao različite dijelove prostora rješenja. Takav model se može implementirati na nekoliko umreženih računala ili na jednom višeprocesorskom računalu. Komunikacijski kanal može biti lokalna mreža, globalna mreža ili Internet, zajednički radni spremnik, cjevovod, red poruka, varijable okoline, itd. Ime raspodijeljeni ili raspodijeljeni genetski algoritam dobio je po tome što se može implementirati i na računala MIMD (Multiple Instruction, Multiple Data Stream) arhitekture s raspodijeljenim radnim spremnikom.

```
Raspodijeljeni_genetski_algoritam {
    genN = 0;
    dok (nije_zadovoljen_uvijet_zavrsetka_evolucijskog_algoritma) {
          za svaku subpopulaciju obavljaj paralelno {
              odaberi mS jedinki prema dobroti;
              ako (genN % mI == 0) {
                   posalji mS jedinki na susjedni otok;
                   primi mS jedniki od susjednog otoka;
                   zamjeni mS jedinki na otoku;
                }
          mutiraj;
          krizaj;
          }
     genN++;
}
```

**Slika 2.2. Raspodijeljeni genetski algoritam**

Multipopulacijski model paralelnih genetskih algoritama je najjednostavniji za implementaciju na umreženim računalima pa se najčešće koristi u praksi. Stoga se pod pojmom paralelnih genetskih algoritama često podrazumijeva raspodijeljeni ili raspodijeljeni genetski algoritam. U literaturi se još naziva i krupnozrnatim genetskim algoritmom ili otočnim paralelnim genetskim algoritmom.



**Slika 2.3. Otočni model djelomično izoliranih subpopulacija**

Postoji mnoštvo primjera u prirodi gdje jedna te ista vrsta živi u potpuno razdvojenim subpopulacijama, primjerice, na više kontinenata. Te razdvojene vrste evoluiraju paralelno. Analogija s prirodom i dostupnost umreženih računala su razlozi velikoj popularnosti DGA. Veličina subpopulacije je obično manja od populacije koju koristi sekvencijski genetski algoritam pa se očekuje skraćenje trajanja izvođenja. Međutim, vrijeme trajanja izvođenja programa ne mora biti kraće onoliko puta koliko je manja subpopulacija od populacije. Naime, treba uzeti u obzir trajanje razmjene jedinki (vrijeme trajanja potrebno za komunikaciju među subpopulacijama). Ukupna veličina populacije je suma svih subpopulacija. Broj subpopulacija i veličina pojedine subpopulacije su parametri koji značajno utječu na ponašanje algoritma [2]. Broj subpopulacija je obično jednak broju procesora ($N_p$) ili broju računala na kojima se raspodijeljeni genetski algoritam izvodi. Početna populacija *P(0)* veličine *N* distribuira se na $N_p$ procesora. Obično se raspodjeljuje jednoliko tako da je veličina otočne populacije ($N_{otok}$) jednaka na svakom od otoka. Svaki procesor obavlja sekvencijski genetski algoritam nad svojim otokom i izmjenjuje $M_s$ jedinki svakih $M_i$ iteracija. $M_i$ se naziva migracijskim intervalom, a $M_s$ migracijskom stopom. Treba naglasiti da migriranje ovdje ne znači seljenje jedinki iz jedne subpopulacije u drugu, već odabir *boljih* jedinki iz jedne subpopulacije čije će kopije nadomjestiti *lošije* jedinke u drugoj subpopulaciji. Postupak migracije obavlja novi genetski operator: operator migracije. Način na koji će se obaviti migracija i kada se ona zbiva određuje pet dodatnih parametara:

- $M_i$ - *migracijski interval* ili period izmjene jedinki između procesora,
- $M_s$ - *migracijska stopa* ili broj jedinki koji se izmjenjuje,
- *strategija odabira boljih jedinki*,
- *strategija odabira jedinki za eliminaciju*
- *topologija razmjene jedinki*.

### 2.2.1. Migracijski interval

Migracijski interval $M_i$ je broj iteracija između dvije migracije. Operator migracije ne djeluje u svakoj iteraciji, kao što je to slučaj s ostalim genetskim operatorima, već svakih $M_i$ iteracija. Često se u literaturi umjesto migracijskog intervala koristi pojam frekvencija migracije $f_i$. Frekvencija migracije je učestalost izmjene jedinki i jednaka je recipročnoj vrijednosti migracijskog intervala. Raspodijeljeni genetski algoritam pretražuje prostor rješenja s pomoću nekoliko relativno izoliranih subpopulacija čime se nastoji izbjeći prerano pojavljivanje najbolje jedinke u subpopulacijama, tj. konvergencija k lokalnom optimumu. Migracijski interval je parametar koji značajno utječe na brzinu konvergencije genetskog algoritma [2]. Prevelika učestalost migracije dovodi do podjednakih subpopulacija i vjerojatno do suboptimalnog rješenja [2]. Povećava li se broj iteracija između migracija omogućava se subpopulacijama nezavisniji razvoj, a time se izbjegava prerana konvergencija.

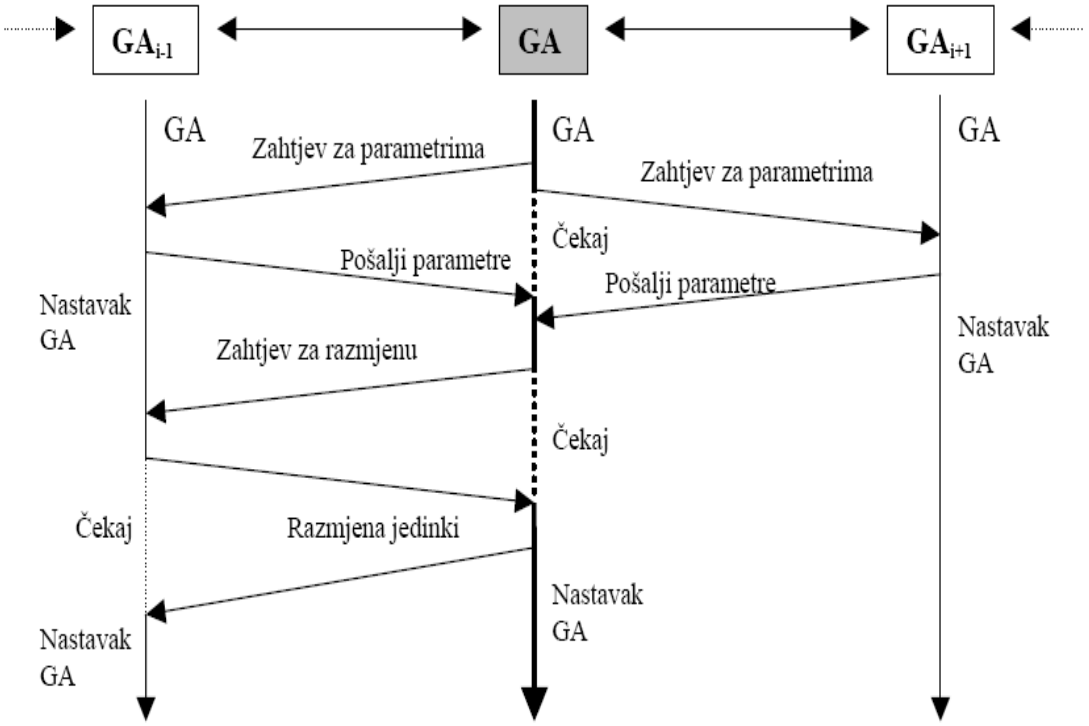
Migracijski interval može biti :

- Konstantan
- Slučajan : potpuno slučajan ili uvjetovan

U slučaju da je migracijski interval konstantan, subpopulacije razmjenjuju rješenja svakih unaprijed određenih $M_i$ iteracija. Promjenjivi migracijski interval može se odabrati potpuno slučajno ili ovisi o nekim drugim parametrima kao što je, primjerice, raspršenost populacije. Za potpuno slučajan migracijski interval važan je parametar $\overline{M}_i$ – prosječna vrijednost migracijskog intervala.

Raspodijeljeni genetski algoritam može biti sinkroni ili asinkroni. Sinkroni raspodijeljeni genetski algoritam sinkrono razmjenjuje jedinke, tj. sve subpopulacije odjednom obavljaju razmjenu. U načelu, DGA je sinkroni jer se procesi sinkroniziraju za vrijeme faze razmjene podataka (komunikacije). S druge strane, spomenuti DGA s uvjetovanim migracijskim intervalom je primjer asinkronog algoritma, jer svaka populacija zasebno odlučuje kada će obaviti razmjenu jedinki. Na slici 2.4 prikazan je komunikacijski protokol između subpopulacija za raspodijeljeni genetski algoritam [2]. Najveći mogući migracijski

interval (ili najmanja moguća frekvencija migracije) jednak je ukupnom broju iteracija. To znači da tijekom optimiranja nema migracije i samo se na kraju evolucijskog procesa izmjenjuju dobivena rješenja. Pokazalo se da je rješenje dobiveno u tom slučaju lošije od rješenja koje se dobiva sa sekvencijskim genetskim algoritmom s jednom populacijom čija je veličina jednaka zbiru svih subpopulacija [2]. Naime, pri malim frekvencijama migracije, subpopulacije ostaju nezavisne i genetski algoritam istražuje različita područja prostora rješenja. Međutim, u tom slučaju se pretraživanje prostora rješenja odvija s manjom učinkovitošću, jer genetski algoritam djeluje nad manje jedinki. Za premale vrijednosti frekvencije migracije svojstva DGA se degradiraju na razinu svojstava sekvencijskog genetskog algoritma s veličinom populacije $N_{otok}$. Drugim riječima, takav DGA ima znatno lošija svojstva od odgovarajućeg sekvencijskog GA s veličinom populacije $N$.




**Slika 2.4. DGA**

Druga je krajnost najmanji migracijski interval (najveća frekvencija migracije) jedinične duljine što znači da jedinke migriraju u svakoj iteraciji. I u tom slučaju, dobivena rješenja su lošija od rješenja koja se dobivaju sa sekvencijskim genetskim algoritmom, osim ako se u svakoj iteraciji ne razmjenjuju najbolje jedinke. Dva su nedostatka DGA s prevelikom frekvencijom migracije: suboptimalna rješenja se u početku evolucijskog procesa prebrzo prošire po svim subpopulacijama pa svi čvorovi pretražuju ista područja prostora rješenja. Drugi nedostatak je da zbog učestale migracije komunikacijski kanal može postati usko grlo postupka optimiranja. Vrijednosti frekvencije migracije za koje subpopulacije postaju izolirane ili kada se suboptimalno rješenje počinje prebrzo širiti po subpopulacijama nazivaju se kritičnim vrijednostima frekvencije migracije. Negdje u intervalu između gornje i donje kritične vrijednosti frekvencije migracije nalazi se optimalna vrijednost tog parametra za koji DGA postiže brže rezultate od sekvencijskog genetskog algoritma.

### 2.2.2. Migracijska stopa

Migracijska stopa ($M_s$) je broj jedinki koje se razmjenjuju svakih $M_i$ iteracija. Slično kao i migracijski interval, ovaj parametar utječe na raznolikost populacije. Naime, ako je mala frekvencija migracije, subpopulacije će biti međusobno različitije i obrnuto. Jednako tako će se subpopulacije međusobno više razlikovati, ako se razmjenjuje mali broj jedinki. Nadalje, ako migracijska stopa teži k veličini populacije, tada će subpopulacije biti međusobno sličnije. To se i poklapa s principom populacijske genetike, koji kaže da se bolje osobine jedinki brže šire u malim populacijama nego u velikim [2]. Najčešće je migracijska stopa jednaka jedinici (razmjenjuje se samo jedna jedinka) ili vrijednosti blizu jedan, tj. razmjenjuje se relativno mali postotak subpopulacije.

### 2.2.3. Strategija odabira jedinki

Strategija odabira jedinki značajno utječe na selekcijski pritisak, a time i na brzinu konvergencije algoritma. Selekcijski pritisak je veći što bolje jedinke imaju veću vjerojatnost preživljavanja u odnosu na loše jedinke i obrnuto. Konvergencija k rješenju (bilo ono i lokalno) je spora, ako je selekcijski pritisak previše slab. S druge strane, preveliki selekcijski pritisak uzrokuje prebrzu konvergenciju, najčešće k lokalnom optimumu [2]. Pravilnim odabirom broja jedinki za migraciju i strategije odabira jedinki može se postići tzv. superlinearno ubrzanje. *Superlinearno ubrzanje* je ubrzanje koje je veće od broja procesora: $a>N_P$ [2]. Skraćenje trajanja izvođenja paralelnog genetskog algoritma uzrokuje dodatni selekcijski pritisak. Premali selekcijski pritisak uzrokuje sporu konvergenciju, pa sekvencijski algoritam sporo konvergira. Raspodijeli li se populacija na $N_P$ jednakih dijelova i primijeni li se raspodijeljeni genetski algoritam, migracija dodatno povećava selekcijski pritisak koji uzrokuje bržu konvergenciju, a time u kraćem vremenu dolazi do rješenja. Tako je, zapravo, superlinearno ubrzanje posljedica skraćenja broja iteracija. U stvarnosti, za jednak broj iteracija, ubrzanje je uvijek manje od broja procesora ($a<N_P$), jer se stanovito vrijeme troši za međusobnu komunikaciju.

Prilikom razmjene treba nekako odabrati jedinke za slanje susjedima kao i jedinke za eliminaciju. Za oba slučaja moguće su dvije strategije:

- *strategija slučajnog odabira*
- *strategija odabira najboljih*, odnosno *najlošijih jedinki*.

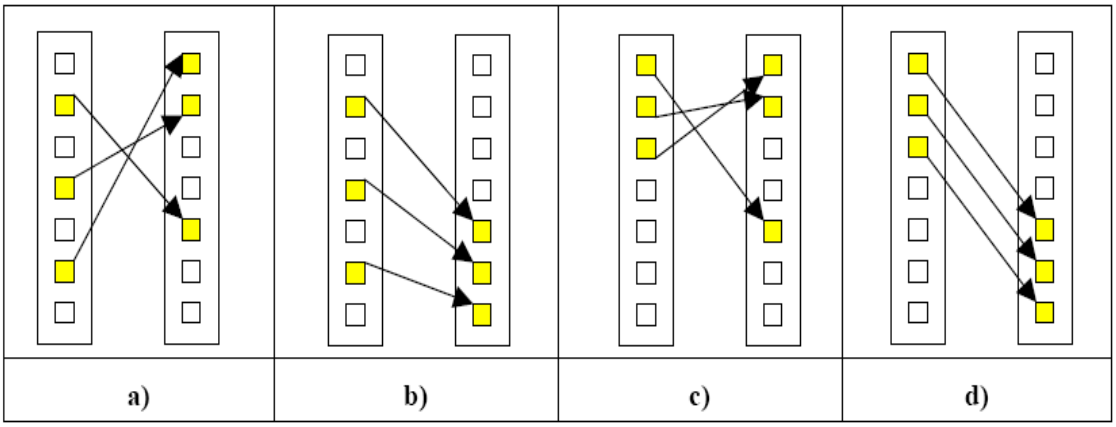
Strategije odabira jedinki se razlikuju prema načinu odabira jedinki za slanje i jedinki za eliminaciju. Budući da se jedinke mogu birati nasumice ili po kriteriju dobrote, postoje sljedeće četiri kombinacije:

- *strategija slučajnog odabira* - slučajno se odabiru jedinke za slanje i za eliminaciju (slika 2.5a);
- *miješane strategije*:
  - slučajno se odabiru jedinke za slanje, a nadomještaju se najgore jedinke (slika 2.5b);

- najbolje jedinke se odabiru za migraciju, a nadomještaju se slučajno odabrane jedinke (slika 2.5c)

- *strategija odabira najboljih i najgorih jedinki* - najbolje jedinke migriraju i nadomještaju najgore jedinke (slika 2.5d).

Na slici 2.5 su jedinke razvrstane od najbolje na vrhu do najgore na dnu.



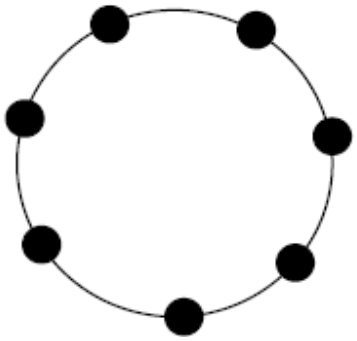

**Slika 2. 5. Strategije odabira jedinki**
**a) Slučajan odabir jedinki za slanje i eliminaciju**
**b) Slučajno odabrane jedinke nadomještaju najlošije**
**c) Najbolje jedinke nadomještaju slučajno odabrane**
**d) Najbolje jedinke nadomještaju najlošije**

U primjeni se najčešće koriste treći ili četvrti način: najbolje jedinke se odabiru za migraciju, a nadomještaju se slučajno odabrane ili najlošije jedinke. Strategija odabira najboljih jedinki za migraciju i najlošijih jedinki za eliminaciju u kombinaciji s prevelikom frekvencijom migracije dovodi do prerane konvergencije.

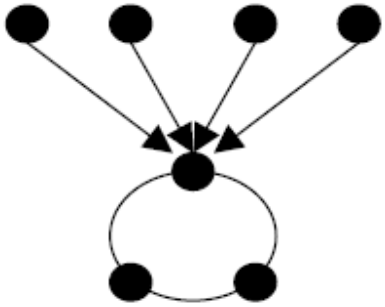
### 2.2.4. Topologija razmjene jedinki

Pokazalo se da na svojstva distribuiranog genetskog algoritma utječe i topologija razmjene jedinki. Topologija razmjene jedinki je plan po kojem čvorovi razmjenjuju jedinke.

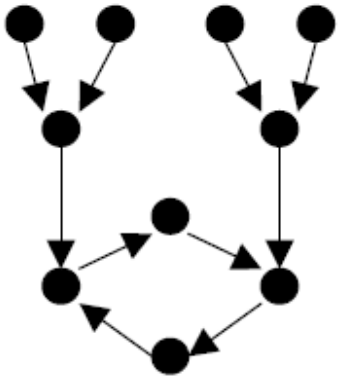
Slika 2.6 prikazuje najjednostavniju prstenastu topologiju koja se najviše koristi u primjeni. Primjer dvoslojne otočne topologije je prikazan na slici 2.7, a višeslojne topologije na slici 2.8. Pokazalo se da je topologija s više slojeva robusnija, jer je opasnost od zaglavljivanja u lokalnim optimumima manja. Dobivena rješenja nižeg sloja se prikupljaju u jednoj ili više subpopulacija višeg sloja. Ukoliko neki od genetskih algoritama nižeg sloja zaglavi u lokalnom optimumu, njegovo rješenje (lokalni optimum) će eliminirati genetski algoritam višeg sloja uporabom selekcije.



**Slika 2. 6. Prstenasta topologija**



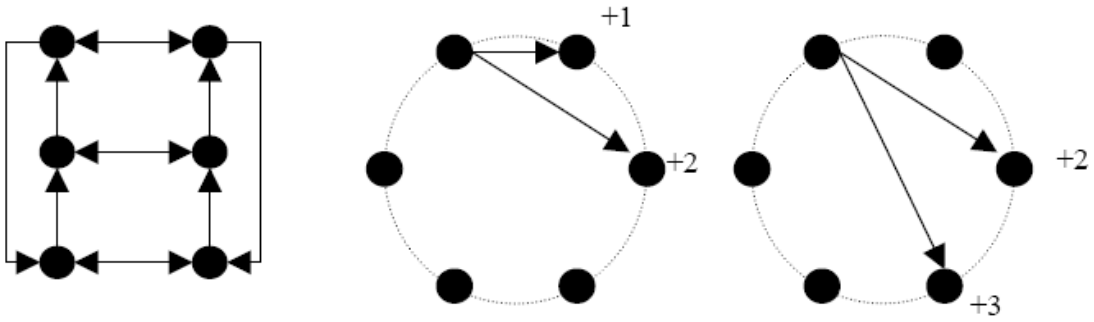
**Slika 2. 7. Dvoslojna otočna topologija**



**Slika 2. 8. Tipična višeslojna topologija s jednim susjedom**

Pored navedenih topologija, može se koristiti i slučajna topologija: genetski algoritam svaki puta prije migracije slučajno odabire kojoj će subpopulaciji poslati svoje jedinke [2].

Broj susjeda ili stupanj povezanosti je također važno obilježje raspodijeljenih genetskih algoritama. Slika 2.9 prikazuje nekoliko primjera topologija s dva susjeda. Svaki čvor šalje svoje najbolje jedinke dvama čvorovima. Jednako tako, svaki čvor dobiva najbolje jedinke od dva čvora. Ti čvorovi se nazivaju susjednim čvorovima. Susjedni čvorovi od kojih se dobivaju i kojima se šalju jedinke mogu biti različiti (slika 2.9b i slika 2.9c), tj. od jednih čvorova se primaju jedinke, a drugim čvorovima se šalju jedinke.




**Slika 2.9. Primjeri topologija s dva susjeda**
**a) Ljevičasta topologija**
**b) Kružna +1+2 topologija**
**c) Kružna +2+3 topologija**

Brzina širenja dobrih rješenja među subpopulacijama ovisi o broju susjeda kojima se šalju jedinke. Ako je topologija "gusta", tj. jedinke jedne subpopulacije migriraju k više subpopulacija, tada će se brzo proširiti dobro rješenje po svim subpopulacijama. Međutim, svaka komunikacija, odnosno razmjena jedinki traje određeno vrijeme i s povećanjem gustoće topologije, povećava se i cijena komunikacije. Topologija može biti statička ili dinamička, ovisno o tome da li se s vremenom mijenja ili ne. Obično se koristi statička topologija, koja se zadaje unaprijed i ostaje nepromijenjena do kraja izvođenja algoritma. Čvorovi u dinamičkom modelu topologije nasumice odabiru čvorove kojima će slati jedinke

(radi se o već spomenutoj slučajnoj topologiji) ili je shema po kojoj će se obavljati razmjena jedinki tijekom evolucije unaprijed zadana.

**Tablica 2.1. Prednosti i nedostaci DGA**

| Prednosti | Nedostatci |
|---|---|
| • jednostavan za implementaciju na višeprocesorskom sustavu s raspodijeljenim kao i zajedničkim radinim spremnikom<br>• raspodijeljena populacija sprječava preranu konvergenciju k lokalnom optimmu, iako DGA brže konvergira od sekvencijskog genetskog algoritma.<br>• Mogućnost superlinearnog ubrzanja | • Mnogo novih parametara, čak sedam<br>• Novi genetski operator, migracija<br>• Teško je odrediti optimalan skup dodatnih parametara, a naročito migracijski interval<br>• Često je cijena komunikacije prevelika |

Jednostavnost implementacije na umreženim računalima i manja vjerojatnost zaglavljivanja u lokalnom optimumu su najznačajnije prednosti, a ujedno i razlozi zbog čega se raspodijeljeni genetski algoritmi danas najpopularniji. Među nedostacima ističe se prevelik broj novih (dodatnih) parametara zbog čega je postupak podešavanja algoritma za rješavanje specifičnog optimizacijskog problema otežan.

# 3. Message Passing Interface

## 3.1. Nastanak i svojstva standarda

U razvoju paralelnih aplikacija javlja se potreba za mehanizmima razmjene poruka (*message passing*) za uporabu na računalima s raspodijeljenom memorijom. 1993. godine osnovan je Message Passing Interface forum od strane četrdesetak industrijskih i istraživačkih organizacija. 1994. godine razvijen je MPI standard. MPI - 2 standard predstavljen je u svom završnom obliku 1997. godine. Standard definira komunikaciju porukama, odnosno razmjenu podataka među procesima. Proces je skup naredbi (program) koji se obavlja na nekom procesoru. Broj procesa u MPI programu je po definiciji konstantan tijekom izvođenja (u osnovnoj inačici standarda nisu predviđeni mehanizmi stvaranja tj. gašenja procesa). Svi procesi obično izvode isti algoritam, međutim mogu se pokrenuti različiti algoritmi za različite procese. MPI je postigao veliku popularnost na području paralelnog računarstva zato jer je neovisan o sklopovlju na kojem se izvodi program, ima velik skup funkcija koje ostvaruju većinu potrebnih funkcionalnosti i dobro iskorištava mogućnosti računala ili mreže na kojoj se program izvodi. Nedostatak MPI je što stavlja odgovornost na programera, jer programer mora sam eksplicitno implementirati distribuciju podataka i svu međuprocesnu komunikaciju i sinkronizaciju.

Načini komunikacije u MPI :

- *Point – to – point* - komunikacija se odvija između dva procesa korištenjem pošalji i primi (send and receive) operacija.
- *Collective* – komunikacija se odvija između grupe procesa tako da jedan čvor skuplja podatke od svih ostalih čvorova i izvodi operaciju
- *Probe* – funkcije za asinkornu komunikaciju
- *Communicator* – mehanizam za razvoj modularnih paralelnih programa (mogućnost korištenja topologija mreže).

Verzija MPI koja se koristila u izradi ovog rada je MPICH2 1.0.8p koja je objavljena u ožujku 2009[*].

## 3. 2. Osnovne funkcije

Svi procesi komuniciraju jedni s drugima razmjenom poruka, čak ako se izvršavaju na istom procesoru. Iako zbog učinkovitosti ugloavnom se dodjeljuje jedan procesor za jedan proces. Bilo koja tehnika paralelnog programiranja mora omogućiti za svaki proces sljedeće mehanizme [3] :

- Otkriti ukupan broj procesa
- Moći identificirati vlastiti proces u grupi procesa
- Poslati poruku određenom procesu
- Moći primiti poruku od nekog procesa

Sve funkcije pisane u ovom poglavlju pisane su C/C++ sintaksom. Svaki C/C++ program mora imati u zaglavlju «#include "mpi.h"».

Dvije funkcije koje se moraju nužno naći u svakom programu koji koristi MPI su :

```
int MPI_Init(int *argc, char ***argv)
```
i
```
int MPI_Finalize().
```

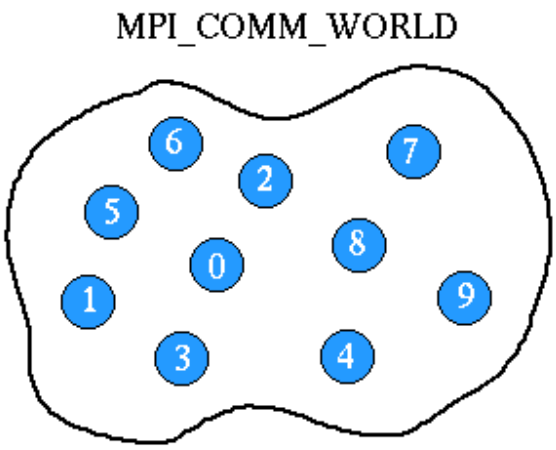
Funkciji MPI_Init prosljeđuju se odgovarajući parametri funkcije main. MPI_Finalize koristi se po završetku programa. Funkcije kojima se radi identifikacija grupe i procesa su :

```
int MPI_Comm_size ( MPI_Comm comm, int *size )
```
i
```
int MPI_Comm_rank ( MPI_Comm comm, int *rank ).
```

S MPI_Comm_size dohvaćamo broj procesa, tj. veličinu grupe, a s MPI_Comm_rank dohvaćamo indeks vlastitog procesa. Indeksi kreću od nule. Grupa procesa u MPI standardu se naziva communicator – određuje skup procesa na koje se odnosi konkretna akcija (većina funkcija zahtjeva komunikator kao jedan od argumenata). Globalna grupa koja uključuje sve uključene procese označava se sa MPI_COMM_WORLD (slika 3.1.). Korisnik može sam definirati grupe procesa (sve takve grupe su podskup globalne grupe). Dakle, jedan

[*] http://www.mcs.anl.gov/mpi/mpich/

proces može biti član više grupa, ali u svakoj grupi ima poseban indeks. Određivanje na koliko i kojim računalima će se program izvoditi obavlja se izvan programa (koristeći opcije dotične MPI implementacije).




**Slika 3.1. Globalna grupa procesa**

## 3. 3. Razmjena poruka

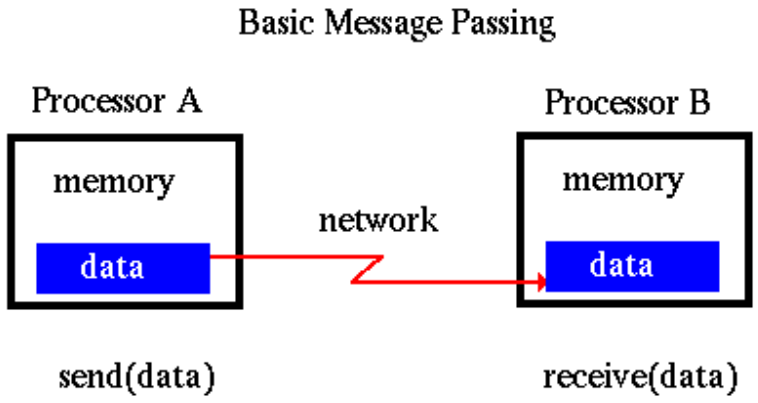
Poruka se sastoji od oznaka poruke i podataka. Podaci su niz jednog od MPI podatkovnih tipova. Osnovni MPI tipovi podataka odgovaraju osnovnim tipovima u C-u. U pozivu funkcije korisnik mora napisati odgovarajuću MPI oznaku za tip podataka koji je prosljedio, npr. ako korisnik prosljeđuje neki integer podatak, odgovarajuća MPI oznaka je MPI_Int. Korisnik također može definirati vlastite tipove podataka u porukama, ali najbolje je kao tip podataka je koristiti MPI_Byte. Osnovne funkcije slanja i primanja poruka :

```
int MPI_Send( void *buf, int count, MPI_Datatype datatype,
int dest, int tag, MPI_Comm comm )
```

i

```
int MPI_Recv( void *buf, int count, MPI_Datatype datatype,
int source, int tag, MPI_Comm comm, MPI_Status *status ).
```

Proces pošiljatelj zove Send, a primatelj Recv.




**Slika 3. 2. Osnovni način funkcioniranja Send – Recv**

Parametri su :

- *buf* je početna adresa podatka u memoriji koji se šalju (kod pošiljatelja), tj. primaju (kod primatelja)
- *count* je veličina podatka
- *datatype* je odgovarajući MPI tip podatka
- *dest* određuje indeks (rank) primatelja, a source indeks pošiljatelja
- *tag* je oznaka vrste poruke
- *comm* je oznaka komunikatora unutar kojeg se komunikacija odvija (MPI_COMM_WORLD ako se radi o globalnoj grupi)
- u strukturi status će nakon primitka biti zapisani podaci o poruci

Uvjeti uspjeha komunikacije :

- indeksi pošiljatelja i primatelja moraju odgovarati
- komunikacija se mora odvijati unutar iste grupe
- oznake poruke (*tag*) moraju biti iste
- memorijski prostor primatelja mora biti dovoljno velik

Poopćenje primanja poruke :

- od bilo kojeg pošiljatelja – navodi se MPI_ANY_SOURCE kao source parametar
- za bilo koju oznaku poruke – navodi se MPI_ANY_TAG kao tag parametar

Često se u praksi javlja potreba slanja poruka svim procesima ili sakupljanje podataka od svih procesa. MPI nudi funkcije za globalnu komunikaciju. Svojstva globalnih operacija :

- svi procesi u grupi moraju pozvati određenu funkciju (globalna operacija odnosi se na sve procese u nekoj gupi)
- sve funkcije su blokirajuće
- nema oznake poruke (parametar *tag*)
- memorijski prostori za primanje moraju biti jednake veličine

Prototip funkcije koja šalje isti podatak svim procesima u definiranom komunikatoru. Parametar root je indeks procesa pošiljatelja.

```
int MPI_Bcast ( void *buffer, int count, MPI_Datatype
datatype, int root, MPI_Comm comm ).
```

Iako MPI ima velik broj funkcija (129), sa ovih sedam navedenih može se ostvariti većina potrebnih funkcionalnosti.

# 4. Programska implementacija

## 4.1. Podsustav za slanje i primanje poruka

Podsustav za slanje i primanje poruka kao što mu samo ime kaže služi za slanje i primanje poruka. Podatci koji se šalju su jedinke. Podsustav je izveden na sljedeći način : virtualni razred Communicator (slika 4.1.) nasljeđuje konkretni razred MPICommunicator. U razredu MPICommunicator su implementirane funkcije za slanje i primanje jedinki.

```
class Communicator {
public :
        virtual bool initialize(StateP state) {
                return true;
        }
        virtual void finalize() {}

        virtual int getRank()= 0;

        virtual int getSize() = 0;

        virtual std::string getName() = 0;

        virtual bool sendInd(int, std::vector<IndividualP>) = 0;

        virtual std::vector<IndividualP> recvInd(int) = 0;

};
```

**Slika 4.1. Razred Communicator**

Svaka jedinak koja će biti poslana prvo se mora zapisati u XML čvor. U TOPLEVEL_NODE se sprema jedinka koja će biti zapisana u XML čvor. Prototip funkcije koja se koristi za slanje naveden je u poglavlju 3.3. Implemenatacija funkcije je vidljiva na slici 4.2.

```
bool MPICommunicator :: sendInd(int dest, std::vector<IndividualP> individuals) {

        XMLNode topNode = XMLNode::createXMLTopNode(TOPLEVEL_NODE);
        for (int i = 0; i < (int)individuals.size(); i++) {
                XMLNode node;

                individuals[i]->write(node);
                topNode.addChild(node);
        }

        char *buffer = topNode.createXMLString(true);

        MPI_Send(buffer, strlen(buffer) + 1, MPI_CHAR, dest, MPI_ANY_TAG, MPI_COMM_WORLD);


        delete [] buffer;

        return false;
}
```

**Slika 4.2. Funkcija za slanje jedinki**

Druga važna funkcija koja je implementirana je funkcija za primanje jedinki.
Prilikom primanje jedinki prvo jedinka mora biti pročitana iz XML čvora, a nakon toga se šalje. Prototip funkcije koja je korištena za primanje jedinki također je navedena u pogavlju 3.3. Funkcija za primanje ima dodatni atribut koji je tipa MPI_Status. U njega se zapisuju podatci o poruci. Implementcija funkcije je vidljiva na slici 4.3.

```
std::vector<IndividualP> MPICommunicator :: recvInd(int src) {
	std::vector<IndividualP> result;
	MPI_Status status;
	int size;

	MPI_Probe(src, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
	MPI_Get_count(&status, MPI_CHAR, &size);

	if (size == 0)
		return result;

	char *buffer = new char[size+1];
	MPI_Recv(buffer, size, MPI_CHAR, src, MPI_ANY_TAG, MPI_COMM_WORLD, &status);

	XMLNode mainNode = XMLNode::parseString(buffer, TOPLEVEL_NODE);
	for (int i = 0; i < mainNode.nChildNode(); ++i) {
		XMLNode child = mainNode.getChildNode(i);

		/*
			Metoda Individual::read() zahtjeva vec postojeci, inicijalizirani
			Individual objekt (zajedno sa svim genotipovima itd.).

		 */

		IndividualP ind = static_cast<IndividualP>(state->population_->at(0)->at(0)->copy());
		if (ind)
			result.push_back(ind);
		else {
			result.push_back(static_cast<IndividualP>(new Individual));
			result[i]->initialize(state);
		}

		result[i]->read(child);
	}

	delete[] buffer;

	return result;
}
```

**Slika 4.3. Funkcija za primanje jedinki**

Razred MPICommunicator implementira još neke funkcije (Slika 4.1.) koje neće biti objašnjavane jer iz samog imena jasna je njihova zadaća.

## 4.2. DistributedGeneticAlg algoritam

DistributedGeneticAlg je virtualni razred koji nasljeđuje razred Algorithm (Slika 4.4.). Konkretna implementacija se nalazi u razredu MigratonOperator koji nasljeđuje DistributedGeneticAlg. Razred DistributedGeneticAlg koristi podsustav za slanje i primanje poruka. Funkcije se dohvaćaju preko pokazivača na razred Communicator :

```
typedef boost::shared_ptr<Communicator> CommunicatorP;
```

U funkciji initialize dohvaćamo Communicator, a ako nije postavljen onda se ispisuje poruka o pogrešci. Sve funkcije dohvaćamo preko pokazivača na State :

```
typedef boost::shared_ptr<State> StateP;
```

```
class DistributedGeneticAlg : public Algorithm {
protected:
        CommunicatorP communicator;
        StateP state;

public:

        virtual bool initialize(StateP state) {

                this->state = state;

                if (! state->isCommunicatorSet()) {
                        state->getLogger()->log(Logger::LOGLEVEL_CRITICAL, "Communicator object not set!");

                        return false;
                }

                communicator = state->getCommunicator();
                return true;
        }

        CommunicatorP getCommunicator() {
                return communicator;
        }

        virtual bool advanceGeneration(StateP, DemeP) = 0;

        virtual bool run(StateP) = 0;

        std::string getName() { return "DistributedGeneticAlg"; }

};
```

**Slika 4.4. Razred DistributedGeneticAlg**

U funkciji advanceGeneration(StateP, DemeP) se nalazi implementacija algoritma, ali u razredu MigrationOperator.

## 4.3. MigrationOperator

Razred MigrationOperator nasljeđuje DistributedGeneticAlg. Podatci koji se koriste :

```
int mI;
int mS;
int velOtoka;
int brOtoka;
int nGenerations;
std::vector<PopulationP> otok;
```

mI je migraciski interval, mS je migracijska stopa (vidi poglavlje 2.). brOtoka je jednak broju procesa, nGenerations je podatak kojim se određuje koliko će se puta algoritam izvoditi. nGenerations se koristi u funkciji run(StateP) u kojoj se nGenerations puta poziva advanceGeneration(StateP, DemeP). Na svakom otoku se nalazi jedna subpopulacija. Veličina subpopulacije određena je parametrom velOtoka, dakle velOtoka je broj deme – ova na jednom otoku. U funkciji advanceGeneration(StateP state, DemeP deme) se nalazi implementacija algoritma (slika 2.2.). Za početak odabiremo mS najboljih jedinki sa svakog otoka koje će biti poslane na susjedni otok, pritom se koristi operator selekcije koji se dohvaća preko pokazivača :

```
SelBestOpP selBest;
```

Jedinke koje će biti zamjenjene se odabiru metodom slučajnog odabira. Na kraju ako je prošlo mI iteracija izvodi se migracija jedinki (slika 4.5.). Podatak genN je jednak broju generacija koje su se izmjenile do tada. Topologija razmjene jedinki je prstenasta. Proces nula prima od procesa kojem je indeks jednak broju procesora umanjen za jedan. Ostali procesi primaju od procesa kojem je indeks jednak vlastitom indeksu umanjen za jedan. Nakon što su jedinke primljene onda se izvršava zamjena subpopulacije s novim (primljenim) jedinkama. Nova subpopulacija se dodaje na otok na mjesto stare subpopulacije.

```
if(genN % mI == 0) {
    communicator->sendInd((communicator->getRank() + 1)%communicator->getSize(), ind);

    if(communicator->getRank() == 0)
        communicator->recvInd(communicator->getSize() - 1);
    else {
        communicator->recvInd(communicator->getRank() - 1);
        sub->at(i)->replace(i, ind.at(i));
        }
    }
    otok[i] = sub;
}
```

**Slika 4.5. Odsječak koda u kojem se izvodi migracija**

Nakon migracije jedinke mutiraju i križaju se odvojeno svaka na svom otoku.

## 4.4. Brzina izvođenja programa

Program je testiran na primjeru gdje je jedinka definirana preko niza nula i jedinica. Duljina jedinke u ovom primjeru je dvjesto, veličina populacije je također dvjesto. Parametri koji se odnose na migraciju su migracijski interval koji iznosi deset i migracijska stopa koja iznosi jedan. Dakle, svakih deset iteracija izvodi se migracija i pritom se šalje jedna jedinka na susjedni otok. Mjerenje je izvedeno na jednom, dva, četiri i osam procesora. Svako mjerenje je izvedeno pet puta. Vrijeme je izraženo u milisekundama.

**Tablica 4.1. Rezultati mjerenja**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Jedan otok | 8031, 4 | | | | | | | |
| Dva otoka | 4325,2 | | | | 4403,2 | | | |
| Četiri otoka | 2793,25 | | 3543,5 | | 3144,5 | | 3199,5 | |
| Osam otoka | 1172,2 | 1341,0 | 1328,2 | 1568,8 | 1415,6 | 1706,2 | 1656,0 | 1281,2 |

Broj otoka jednak je broju procesora. Na svakom otoku nalazi se jedna subpopulacija, tj. ako imamo dva otoka onda će postojati dvije subpopulacije gdje je svaka za pola manja od početne populacije. Izmjereno je vrijeme brzine izvođenja na svakom otoku, tj. procesoru. Ako imamo osam otoka onda će postojati osam različitih vremena. Prvo vrijeme predstavlja aritmetičku sredinu brzina izvođenja na prvom otoku, drugo vrijeme predstavlja aritmetičku sredinu brzina izvođenja na drugom otoku, itd. Vidi se da su vremena dobivena

mjerenjem na više procesora manja od vremena dobivenog mjerenjem na samo jednom procesoru zato jer je populacija manja.

# 5. Zaključak

Genetski algoritam ne određuje na koji način je genetski materijal pohranjen u radni spremnik, niti kako treba manipulirati genetskim materijalom, već samo kaže da se genetski materijal treba razmjenjivati, slučajno mijenjati (mutirati) i da bolje jedinke trebaju preživljavati selekciju. Kako će se te operacije obavljati određuje se na temelju iskustva. Stoga postoji velika sloboda u izradi GA. Cijena te slobode je loša učinkovitost, a to je ono čime smo se bavili u ovom radu – jednim vidom optimizacije. Za metodu optimizacije odabran je DGA zbog analogije s prirodom i mogućnosti umrežavanja računala. Upravo ta dva svojstva su razlog velike popularnosti DGA. DGA je najpopularniji oblik PGA zato jer je multipopulacijski model paralelnih algoritama najjednostavniji za implemantaciju, pa se stoga i najčešće koristi u praksi te je predmet mnogih znanstvenih istraživanja. U prirodi ima mnogo primjera gdje neka vrsta živi na više izoliranih mjesta, tj. jedna populacija je podijeljena na više subpopulacija i svaka se razvija posebno. Budući da subpopulacije evoluiraju paralelno, možemo zaključiti da se u prirodi evolucija odvija paralelno. Kako je dokazano da se u manjim populacijama bolje karakteristike brže prenose, morali smo se zapitati zašto ne bismo to imitirali na računalima. Razvojem računalne moći, povećanjem broja procesora ili samo umrežavanjem računala možemo jednostavno simulirati procese iz prirode. Tu se MPI pokazuje kao dobar alat za ostvarenje modela višeprocesorskog računala. Dijeljenjem populacije na manje dijelove uglavnom dobivamo smanjenje trajanja izvođenja programa. Upravo zbog više manjih populacija brže se pronalazi optimalno rješenje, iako i ovdje postoje zamke koje mogu smanjiti učinkovitost algoritma naspram sekvencijskog, ako je na primjer migracijski interval premalen ili broj jedinki koji se prenosi u susjednu subpopulaciju prevelik. Područje istraživanja GA pa tako i PGA je veliko i još uvijek nisu otkrivene sve mogućnosti.

# 6. Literatura


1. Golub, M. Skripta 1.dio : Genetski algoritmi, 27. rujan 2004. *Genetski algoritmi, prvi dio,* http://www.zemris.fer.hr/~golub/ga/ga_skripta1.pdf, 1. lipanj 2009.
2. Golub, M. Skripta 2.dio : Paraleleni Genetski algoritmi, 4. listopad 2004. *Genetski alogritmi, drugi dio,* http://www.zemris.fer.hr/~golub/ga/ga_skripta2.pdf, 1. lipanj 2009.
3. Jakobović, D. Predavanja, 5. 10. 2007. *Paralelno programiranje u raspodijeljenim sustavima,* http://www.fer.hr/_download/repository/Paralelno_programiranje_predavanja%5B7%5D.pdf, 1. lipanj 2009.
4. Žderić, I. Seminar, svibanj 2007. *Message Passing Interface,* http://www.zemris.fer.hr/~yeti/studenti/radovi/Zderic_Ivan_seminar.pdf, 1. lipanj 2009.
5. http://www.mhpcc.edu/training/workshop/mpi/MAIN.html#Message_Passing, 1. lipanj 2009.


# 7. Sažetak

U uvodu se navodi motivacija za uvođenje raspodijeljenih genetskih algoritam kao jednog vida optimizacije. U drugom poglavlju se ukratko opisuju različite vrste paralelnih genetskih algoritama i opisuju se novi parametri koji se koriste kao npr. migracijski interval, migracijska stopa i način razmjene jedinki, tj. topologija razmjene. Treće poglavlje je posvećeno MPI - u. Opisuju se glavne funkcije koje su korištene. Četvrto poglavlje je opis programskog ostvarenja s rezulatima mjerenja. Posljednje peto poglavlje je zaključak.