SVEUČILIŠTE U ZAGREBU

**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

**ZAVRŠNI RAD br. 215**

# Rješavanje problema N kraljica uz pomoć genetskog algoritma

Rene Huić

Zagreb, lipanj, 2008.

## Sažetak:

# 1. Uvod

Još iz davnih vremena Al-Khwārizmī-a se koriste i algoritmi za pomoć pri rješavanju raznih problema, ne zato jer se mora, već zato jer se može. Tijekom prošlih stoljeća su se razvijali algoritmi, a razvojem računala se samo još više poticalo na intenzivnije razvijanje algoritama, jer se odmah uvidio potencijal računala da zamijeni čovjekovo sporo ručno računanje, podliježno greškama. Početkom 60-tih godina došlo se na ideju iskoristiti princip evolucije kod izrade algoritma. Jedno od ostvarenja te ideje je i genetski algoritam, sa temeljima još iz 60-tih. Genetski algoritam se tek zadnjih 20-tak godina počeo intenzivnije proučavati i koristiti u praktične svrhe, a sve to zahvaljujući razvoju stolnih računala, te naglim povećanjem kapaciteta memorije i procesorske snage računala, koji dozvoljavaju da se genetski algoritam koristi i kod složenijih problema. Najveći utjecaj na učinkovitost genetskog algoritma su njegovi operatori, poput funkcije dobrote, broja generacija ili vjerojatnosti mutacije. Uz to, sama implementacija operatora ovisi o prikazu jedinke unutar populacije, tj. rješenja. No, tu dolazi do izražaja jedna loša strana genetskog algoritma, a to je da se skoro svakom problemu operatori moraju posebno prilagođavati kako bi mogao raditi na efikasan i učinkovit način.

Jedan od glavnih zadataka današnjih znanstvenika koji se bave proučavanjem genetskih algoritama je pronaći univerzalne operatore koji bi se mogli koristiti za rješavanje svih problema. Proučavaju se različiti operatori te njihov utjecaj na rješenje dobiveno genetskim algoritmom kod određenog problema. U ovom će radu biti opisani različiti operatori za dva različita prikaza rješenja kod tipičnog kombinatoričkog problema, problem N kraljica, u kojem je cilj postaviti N kraljica na šahovsku ploču veličine NxN, a da se pri tome nijedan par kraljica ne napada.

# 2. Genetski algoritam

## *2.1 Osnove genetskog algoritma*

Genetski algoritam je heuristička metoda koja služi za rješavanje optimizacijskih problema. Kombinira slučajno i usmjereno pretraživanje prostora rješenja. Princip rada genetskog algoritma analogan je evolucijskom procesu u prirodi. Prirodnom selekcijom se biraju jedinke koje će preživjeti i stvoriti potomstvo, tj. preživljavaju bolje, jače i sposobnije jedinke, i veća je vjerojatnost da će se te jedinke pariti kako bi nastali potomci koji će tvoriti slijedeću generaciju. Isto tako, potomci bi trebali biti bolji od roditelja baš zbog toga jer su nastali od sposobnijih jedinki. Na taj način populacija napreduje i sve se bolje prilagođava okolini. Preživljavanjem najboljih jedinki, genetski algoritam se usmjerava prema optimumu i pretražuje samo one dijelove prostora rješenja koji sadržavaju dobra rješenja. [1]

Prva generacija se slučajno odabire. Svaka slijedeća generacija se popunjava križanjem najboljih jedinki iz trenutne populacije. Svaka jedinka predstavlja jednu kombinaciju ulaznih parametara, kodiranih na primjereni način. Podaci koje sadrži jedinka predstavljaju njen genetski materijal. Jednako kao i u prirodnoj selekciji, selekcijom u genetskom algoritmu se biraju jedinke prema svom genetskom materijalu: one sa kvalitetnijim genima (tj., one koje daju bolje rezultate) će imati veću šansu za preživljavanje, te će dobiti priliku da prenesu svoj genetski materijal na potomstvo. Na taj način populacija u genetskom algoritmu napreduje, dajući sve bolja rješenja za problem koji se optimira. Proces selekcije, reprodukcije i manipulacije genetskim materijalom se ponavlja sve dok nije zadovoljen uvjet zaustavljanja genetskog algoritma, tj. sve dok se ne dođe do zadovoljavajućeg rezultata. Kako se ne bi dogodilo da se završi u nekom lokalnom optimumu, genetski algoritam koristi i operator mutacije koji simulira “nezgodu“ u stvarnoj evoluciji, jedinku koja se ne uklapa u populaciju. Razlog ovome je slučajan skok na neko još neistraženo područje prostora rješenja u kojem se možda nalazi bolje rješenje od onih trenutno sadržanih u populaciji. Ako se to ne dogodi, već se dođe do lošijeg rješenja, selekcija će se pobrinuti da se to rješenje brzo zamijeni s onim boljima.

## *2.2 Operatori genetskog algoritma*

Jedinka (ili kromosom) predstavlja jedno potencijalno rješenje problema koji se rješava, kodirano na određeni način i u njemu su sadržani svi podaci koji obilježavaju jednu jedinku. Budući da način kodiranja rješenja može znatno utjecati na učinkovitost genetskog algoritma, pravilan izbor strukture podataka koja će se koristiti predstavlja vrlo važan korak u implementaciji genetskog algoritma. Za odabrani način prikaza rješenja potrebno je definirati i genetske operatore, pri čemu je bitno da operatori ne stvaraju nove jedinke koje predstavljaju nemoguća rješenja, jer se time mnogo gubi na efikasnosti algoritma. Postoji mnogo različitih prikaza koji se koriste u genetskim algoritmima za rješavanje različitih problema. U optimizaciji funkcija se obično koriste binarni (kromosom je predstavljen nizom bitova) i prikaz realnim brojem, u raznim kombinatoričkim problemima se koriste nizovi brojeva, polja, binarna stabla, i sl.[2]

Rad genetskog algoritma može se najbolje opisati sa slijedećih pet koraka[5]:

1. kodiranje jedinki
2. evaluacija dobrote jedinki
3. selekcija
4. križanje (slika 1)
5. mutacija
6. dekodiranje rješenja

Kodiranje jedinki je početni korak u dizajnu genetskog algoritma. Svaka jedinka mora biti jedinstveno predstavljena u populaciji. Nakon što se odredi kako jedinke izgledaju, najčešće se slučajnom metodom generira početna populacija u kojoj se za svaku jedinku računa (evaluira) dobrota.

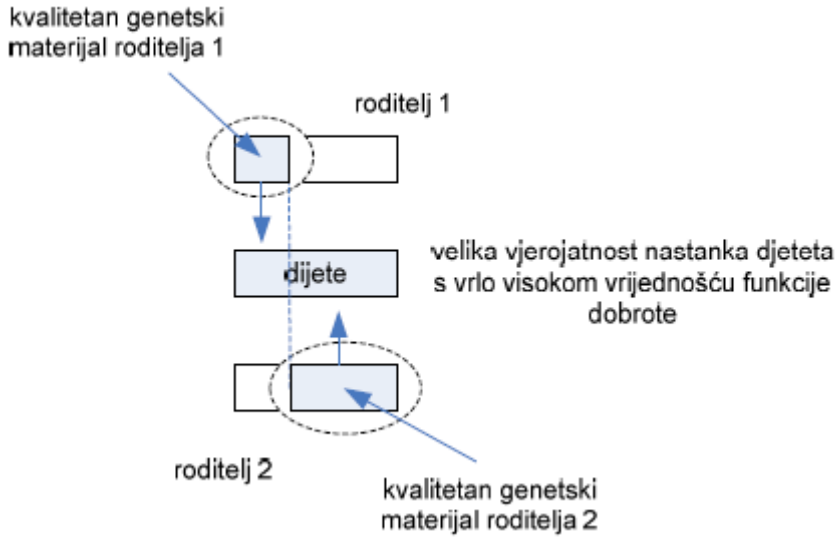
Dobrota je mjera koja genetskom algoritmu govori je li neko rješenje bliže ili dalje od optimuma u usporedbi s ostalim članovima populacije. Oni članovi koji imaju bolju dobrotu su bliži optimumu i za njih je veća vjerojatnost križanja. Genetski algoritmi ne

očekuju nikakve dopunske informacije o problemu optimiranja osim funkcije dobrote. Ona ne mora biti ni neprekidna, ni derivabilna, pa čak niti egzaktno definirana, isto tako ne mora biti jednaka funkciji cilja. Kvaliteta jedinki mjeri se pomoću funkcije cilja f. Neki postupci selekcije zahtijevaju da funkcija cilja ne može biti negativna te da ne poprima samo velike, približno jednake, vrijednosti. Zbog toga se obično u svakoj iteraciji obavlja translacija funkcije cilja, tj. vrijednosti funkcije cilja pojedine jedinke oduzima se najmanja vrijednost funkcije cilja u cijeloj populaciji. Rezultat je funkcija dobrote d koja se računa prema izrazu[6]:

D = f - fmin(t),

gdje je fmin(t) najmanja vrijednost funkcije cilja u generaciji t.

Križanje je postupak u kojem se odabiru dvije jedinke (roditelji) iz populacije te se njihovi dijelovi međusobno kombiniraju kako bi nastale nove jedinke (djeca). Novonastale jedinke se umeću u slijedeću generaciju umjesto onih lošijih.




**Slika 1** *Križanje*

Neke od novonastalih jedinki se još dodatno i mutiraju kako bi se proširio prostor rješenja koji se pretražuje, i da se izbjegne slučaj zaglavljenja u lokalnom optimumu. Na primjer, ako se uzme da je jedna jedinka kodirana kao niz znakova, mutacija može biti zamjena nekog slučajno odabranog znaka drugim slučajno odabranim znakom.

Ovaj postupak se ponavlja sve dok se ne dostigne neki unaprijed definirani kriterij zaustavljanja nakon kojeg se najbolje postignuto rješenje dekodira. To rješenje je najbliže optimumu ili je optimum (slika 2).


**Slika 2** *Prikaz rada genetskog algoritma*

Za svaku vrstu problema potrebno je dizajnirati posebni genetski algoritam ili je potrebno prilagoditi problem nekom već postojećem genetskom algoritmu. Evoluciju rješenja korištenjem genetskog algoritma moguće je usmjeravati podešavanjem parametara koje neki genetski algoritam može imati, a može se i dizajnirati algoritam tako da se tokom rada algoritma parametri mogu sami podešavati, bez uplitanja korisnika, kako bi se promijenilo ponašanje algoritma. O parametrima ovisi koliko će se vremena potrošiti na evoluciju rješenja, kojom će se brzinom algoritam usmjeravati prema potencijalnom optimumu, hoće li pretraživati veći ili manji dio prostora rješenja, itd. Skup parametara koji za jedan genetski algoritam i jedan problem daje kvalitetne rezultate, za neki drugi problem ne mora dati jednako kvalitetne rezultate.

Veličina populacije, broj generacija (iteracija) i vjerojatnost mutacije su tri standardna parametra koje koriste svi genetski algoritmi prilikom evolucije rješenja i o njima najviše ovisi način ponašanja genetskog algoritma. Postoji još različitih parametara koje koriste genetski algoritmi, ali tu o korištenju takvog pojedinog parametra, ovisi problem i način na koji se taj problem rješava genetskim algoritmom. Neki od tih «nestandardnih» operatora može biti duljina kromosoma ili vjerojatnost križanja[6].

**Veličina populacije** – parametar koji direktno utječe na kvalitetu dobivenih rješenja, ali isto tako i ovisno o selekciji može imati velik utjecaj na duljinu izvođenja genetskog algoritma. Npr. kod selekcija koje će ovdje biti kasnije objašnjene, veličina populacije uvelike utječe na duljinu izvođenja, dok kod npr. turnirske selekcije, veličina populacije ne utječe direktno na vrijeme izvođenja genetskog algoritma. Pokazalo se da manja veličina populacije je učinkovita ako se uzme manji broj generacija, dok bi se za veću veličinu populacije trebalo povećati broj generacija.

**Broj generacija (iteracija)** – parametar koji direktno utječe i na kvalitetu dobivenih rješenja i na duljinu izvođenja genetskog algoritma. Logički je da veći broj generacija, daje više vremena genetskog algoritmu da pronađe optimum zadanog problema, no kod dizajniranja genetskog algoritma za teže probleme, vrijeme se uzima kao jedan od važnih faktora. Naravno da se uvijek pokušava naći optimum rješenja za problem, ali je nekada cijena za vrijeme koje bi trebalo genetskom algoritmu jednostavno prevelika, pa se zato pokušava pronaći zadovoljavajuće rješenje u što kraćem vremenu. Zadovoljavajuće rješenje je ono rješenje koje je dovoljno blizu optimumu, a za koje je potrebno mnogo manje vremena kako bi se do njega došlo.

**Vjerojatnost mutacije** – jedan od najvažnijih parametara kod genetskih algoritama, jer direktno utječe na to da li će doći kod određene jedinke ili kod određenog bita unutar jedinke do mutacije, a sama mutacija, kao što je već spomenuto, radi slučajne skokove algoritma po prostoru rješenja, što omogućuje izbjegavanje zaglavljivanja algoritma u lokalnim optimumima i proširivanje područja pretrage na još neistražene dijelove prostora rješenja. Moguće je da će dobivena rješenja biti lošija od originalnih, ali zbog toga ta lošija rješenja imaju manju vjerojatnost ulaska u daljnje reprodukcije čime se ostavlja prostor za napredovanje u pravom smjeru. Genetski algoritam je

dosta osjetljiv na promjene ovog parametra, pa je dosta bitno pažljivo odabrati njegove vrijednosti.

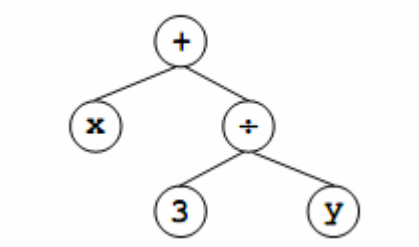
Parametri genetskog algoritma predstavljaju važnu komponentu rada algoritma. Mnogi su znanstvenici pokušavali odrediti skupove parametara koji bi bili optimalni za velik broj problema. No, nažalost se pokazalo da svaki problem ima svoj skup optimalnih parametara te da nije moguće odrediti parametre koji bi bili dobri za širok spektar problema. Ali se uspjelo doći do zaključaka o tome koji su parametri dobri za početak pretraživanja genetskog algoritma i od kuda treba krenuti kako bi se odredili optimalni parametri za pojedini genetski algoritam. Postojali su i pokušaji da se odrede relacije pomoću kojih bi se mogla izračunati koja je poželjna vrijednost trećeg parametra ako su poznata prva dva. Nemoguće je odrediti skupove parametara koji bi bili optimalni za rješavanje velikog skupa problema, ali postoje određene međuzavisnosti i pravilnosti između pojedinih parametara:

1) za veće populacije je dobro povećati broj generacija i smanjiti vjerojatnost mutacije
2) veći broj generacija daje najčešće bolja rješenja
3) jednako kvalitetna rješenja se mogu dobiti sa različitim skupovima parametara

Osim definiranja fiksnih parametara na početku genetskog algoritma, tj. parametara koji se tijekom rada genetskog algoritma ne mijenjaju, moguće je dizajnirati genetski algoritam s dinamičkim parametrima, tj. s parametrima koji se sami podešavaju sa svrhom da promijene ponašanje genetskog algoritma. Npr. ako se najbolja jedinka ne promijeni kroz određen broj generacija, tada sam algoritam mijenja parametre kao npr. veličinu populacije i vjerojatnost mutacije kako bi se izbjegla mogućnost zaglavljivanja u nekom lokalnom optimumu.

## *2.3 Prikaz rješenja*

Uz same parametre genetskog algoritma, prikaz same jedinke, tj. prikaz rješenja genetskog algoritma je isto vrlo važna komponenta kod izrade genetskog algoritma. Pošto je jedinka zapravo potencijalno rješenje problema, koje je ovisno o prikazu same jedinke kodirano na određeni način, vrlo je važno odrediti pravilnu strukturu prikaza rješenja, jer uvelike utječe na samu efikasnost genetskog algoritma. Jednako tako je potrebno odrediti pravilne genetske operatore za određeni prikaz rješenja, te ih prilagoditi da ne stvaraju jedinke koje bi predstavljale nemoguća rješenja, jer inače genetski algoritam uvelike gubi na efikasnosti.




**Slika 3** ***Prikaz rješenja binarnim stablom***

Postoji mnogo različitih načina prikaza rješenja, pri čemu prikaz rješenja uvelike ovisi o problemu koji se rješava genetskim algoritmom. U raznim kombinatoričkim problemima se koriste binarna stabla, nizovi brojeva ili polja kao prikaz rješenja, dok se kod optimiziranja funkcija najčešće koristi binarni prikaz (jedinka je prikazana nizom bitova).

$$(0\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 0)$$

**Slika 4** *Binarni prikaz rješenja*

# 3. Problem N kraljica

## *3.1 Opis problema N kraljica*

Problem n-kraljica je klasičan kombinatorički problem u području umjetne inteligencije. Budući problem ima jednostavnu, regularnu strukturu, i inherentne je kompleksnosti, korišten je za stvaranje mjernih programa za algoritme pretraživanja umjetne inteligencije. Problem je polinomne složenosti (P) ako koristi manje od O($n^t$) koraka, gdje je t neki konačni broj. Ako se pogođeno rješenje problema može provjeriti u polinomnom vremenu (tj. mogu se ponoviti rezultati testiranja) onda se kaže da je NP, baš kao što je i problem N kraljica. Skup problema koji leže u NP je jako velik (uključuje i problem faktorizacije brojeva), pri čemu složenosti mogu dosezati $O(2^n)$ ili $O(n!)$. Problem je NP-težak ako nema nijednog problema u NP kojeg je lakše riješiti.[7]

Za rješavanje problema je potreban algoritam, koji je efikasan u pretrazi i optimizacijskim okolinama, isto tako mora biti sposoban zadovoljiti ograničenja problema. Kod takvih problema je cilj pronaći sve dodijeljene vrijednosti takve da zadovoljavaju sva ograničenja. Ne postoji ni jedan algoritam polinomne složenosti koji može riješiti NP-težak problem, ali postoje neke nedeterminističke metode koje omogućuju rješavanje NP problema u polinomijalnom vremenu. Jedna od takvih metoda je genetski algoritam, no mora se upamtiti da je genetski algoritam samo aproksimativan, tj. ne jamči pronalaženje optimalnog rješenja zadanog problema.[4]

Problem 4-kraljica je najjednostavniji primjer problema n-kraljica za koji postoji rješenje. Potrebno je postaviti četiri kraljica na šahovsku ploču veličine 4x4, tako da se nijedan par kraljica ne može međusobno napasti. To znači da nijedan par kraljica se ne može nalaziti u istom redu ili stupcu ili na istoj dijagonali. U generalnom problemu se treba postaviti N kraljica na šahovsku ploču veličine NxN, tako da se nijedan par kraljica ne može napadati. Postoji $\binom{n^2}{n}$ različitih načina da se postavi N

kraljica na ploču veličine NxN. Za relativno malen problem od 10 kraljica, postoji više od $1.73 * 10^{13}$ načina postavljanja na ploču. To je ogroman prostor rješenja za pretraživanje, kod tako malog problema.

## *3.2 Prikaz jedinke*

Od mnogih mogućih prikaza rješenja, uzeta su dva tipična za ovakve probleme: permutiranim nizom i bit matricom. Zbog velike kompleksnosti problema N kraljica, prikazi rješenja su se još prilagodili problemu kako bi se smanjila kompleksnost i ubrzao rad genetskog algoritma. Tako je i kod prikaza s permutiranim nizom i kod bit matricom, samim odabirom prikaza jedinke riješen problem sa konfliktima u istom retku ili stupcu.

(1 2 3 4 5 6 7 8 9)

**Slika 5** *Prikaz rješenja permutiranim nizom*

Kako se može i vidjeti iz slike 5, micanjem duplikata u nizu se rješava problem pozicioniranja kraljica u istom redu, a predstavljanjem pozicije u stupcu samo jednim bitom permutiranog niza se rješava problem pozicioniranja više kraljica u istom stupcu.

$$\begin{matrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{matrix}$$

**Slika 6** *Prikaz rješenja bit matricom*

Prikaz rješenja bit matricom se sastoji od matrice dimenzija NxN, pri čemu se pozicija kraljica obilježava sa 1, dok se sve ostale pozicije obilježavaju sa 0. Limitiranjem broja jedinica po redu i stupcu na samo jednu, riješen je problem pozicioniranja više kraljica u istom stupcu ili redu.

Odabir genetskih operatora ovisi o prikazu rješenja, zbog toga su se koristili različiti operatori križanja i mutacija kod prikaza rješenja, dok su operatori selekcije i elitizma ostali jednaki, jer ne ovise o samom prikazu rješenja.

## *3.3 Ocjena dobrote*

Poteškoća s određivanjem funkcije dobrote kod mnogih problema je u stvaranju funkcije koja će što bolje odrediti koliko je neko rješenje dobro, tj. koliko je blizu optimumu. Kod problema N kraljica nije potpuno tako, jer ovdje je slučaj da ili se nađe rješenje ili se ne nađe, nema između. Zbog toga je cilj funkcije dobrote odrediti koliko je neko pogrešno rješenje blisko točnom. Prikazi rješenja su odabrani tako da se kod oba prikaza eliminiraju konflikti po stupcima i recima. U tom slučaju je jedini problem koji se javlja kod problema N kraljica zapravo taj da dolazi do konflikata kraljica po dijagonalama. Uzet će se da je dobrota najbolje jedinke zapravo ukupan broj kraljica, tj. N. Za svaku kraljicu koja je u konfliktu sa rješenjem, dobrota će se smanjivati za 1.

Za određivanje konflikata na dijagonalama unutar nekog rješenja kod permutiranog prikaza $(k_1, k_2, ..., k_n)$, koriste se dvije formule:

1) $k_i = k_j + (i - j)$

2) $k_i = k_j - (i - j)$

Glavni algoritam pronalaska konflikata je takav, da na početku počne od trenutne pozicije kraljice u dva smjera, budući postoje dvije dijagonale, gornja i donja. Za svaku sljedeću poziciju ispituje se je li u konfliktu na temelju dvije prethodno napisane

formule. S prvom se provjerava da li se trenutno ispitivana pozicija kraljice nalazi na gornjoj dijagonali početne kraljice, dok se s drugom formulom ispituje da li se trenutno ispitivana pozicija kraljice nalazi na donjoj dijagonali početne kraljice. Kada se nađe konflikt na nekoj od dijagonala, tada se prekida potraga na toj dijagonali. Na taj način se izbjegava pogrešno računanje konflikata u slučaju više kraljica na istoj dijagonali. Složenost algoritma je u najgorem slučaju, tj. kada nema konflikata, jednaka $O(n^2)$, što je zapravo relativno sporo, dok u najboljem slučaju, tj. kada je svaka kraljica u konfliktu sa onom sljedećom, se složenost smanjuje, jer se ne mora kod svakog bita jedinke prolaziti do kraja niza, već se pronađu traženi konflikti nakon prolaska dva susjedna bita.

## *3.4 Genetski operatori*

Zbog korištenja dva različita načina prikaza rješenja, operatori mutacija i križanja se razlikuju za određeni prikaz, konkretno, operatori križanja PMX, OX i Custom, i operatori mutacije obični i uvjetni se koriste kod prikaza rješenja permutiranim nizom, kako se može vidjeti iz primjera, dok se operator križanja matrično križanje i operator mutacije matrično mutiranje, koriste kod prikaza rješenja bit matricom.

### 3.4.1 Križanja

***PMX*** *(Partially matched crossover)* – Goldberg i Lingle predložili [8] – gradi djecu tako da odabire podniz od jednog roditelja i očuva poziciju i red što je više moguće kraljica drugog roditelja. Podniz roditelja se selektira nasumičnim odabirom dva mjesta rezanja, koja služe kao granica kod zamjene. Npr. dva roditelja (pri čemu su mjesta rezanja označena sa « | »)

r1 = ( 1 2 3 | 4 5 6 7 | 8 9 )
r2 = ( 4 5 2 | 1 8 7 6 | 9 3 )

bi proizvela dvoje djece na sljedeći način. Prvo, dijelovi između mjesta rezanja bi se zamijenili (simbol « x » će se u sljedećih par primjera smatrati kao «trenutačno nepoznato»):

d1 = ( x x x | 1 8 7 6 | x x )
d2 = ( x x x | 4 5 6 7 | x x )

Ova zamjena isto tako definira i seriju mapiranja:

1 <--> 4, 8 <--> 5, 7 <--> 6, 6 <--> 7

Tada možemo popuniti nepoznata mjesta (od originalnih roditelja), ali ona za koja ne postoji konflikt (ne postoji već kraljica u tom redu):

d1 = ( x 2 3 | 1 8 7 6 | x 9 )
d2 = ( x x 2 | 4 5 6 7 | 9 3 )

Na kraju, prvi « x » kod djeteta d1 (koji bi trebao biti 1, ali postoji konflikt, jer već postoji kraljica u tom redu) se zamjenjuje sa 4, jer postoji mapiranje 1 <--> 4. Jednako tako, drugi « x » kod djeteta d1 se zamjenjuje sa 5, te dva « x » kod drugog djeteta sa 1 i 8. Nakon toga djeca izgledaju kao:

d1 = ( 4 2 3 | 1 8 7 6 | 5 9 )
d2 = ( 1 8 2 | 4 5 6 7 | 9 3 )

***OX*** *(poredano križanje)* – Davis predložio [8] – gradi djecu tako da odabire podniz od jednog roditelja i očuva poziciju i red što je više moguće kraljica drugog roditelja. Npr. dva roditelja (pri čemu su mjesta rezanja označena sa « | »)

r1 = ( 1 2 3 | 4 5 6 7 | 8 9 )
r2 = ( 4 5 2 | 1 8 7 6 | 9 3 )

bi proizvela dvoje djece na sljedeći način. Prvo, dijelovi između mjesta rezanja bi se samo prepisali u djecu:

d1 = ( x x x | 4 5 6 7 | x x )
d2 = ( x x x | 1 8 7 6 | x x )

Dalje, počevši od drugog mjesta rezanja jednog roditelja, pozicije kraljica iz drugog roditelja su kopirana u istom poretku, pri tome izostavljajući ona mjesta kraljica koja su već zauzeta. Pri dolasku do kraja niza, krećemo od početne pozicije u nizu. Poredak mjesta kraljica kod drugog roditelja (s početkom od drugog mjesta rezanja) je:

9 – 3 – 4 – 5 – 2 – 1 – 8 – 7 – 6

nakon micanja pozicija 4, 5, 6 i 7, koja su već zauzele kraljice u prvom djetetu, dobijamo

9 – 3 – 2 – 1 – 8

Ovaj poredak mjesta kraljica se stavlja u prvo dijete (s početkom od drugog mjesta rezanja):

d1 = ( 2 1 8 | 4 5 6 7 | 9 3 )

Jednako tako dobijemo i drugo dijete:

d2 = ( 3 4 5 | 1 8 7 6 | 9 2 )

***Custom*** – gradi djecu tako da uzme nasumičan broj (između 1 i n/2, pri čemu n predstavlja broj kraljica) parnih ili neparnih bitova (ovisno o tome da li je neparna ili parna generacija) iz jednog roditelja, te ostale bitove dopuni iz drugog roditelja ako ne postoje konflikti. U slučaju da postoje, moguće je nadopuniti na 3 načina:

- uzmu se bitovi koji nedostaju, te se nasumično nadopune,

- uzmu se bitovi koji nedostaju, te ih postavimo prema mapiranju po kojem smo stavili parne/neparne bitove,

- uzmu se bitovi koji nedostaju, te ih postavimo uzlazno sortirajući.

Npr. dva roditelja

r1 = ( 1 2 3 | 4 5 6 7 | 8 9 )
r2 = ( 4 5 2 | 1 8 7 6 | 9 3 )

bi proizvela dvoje djece na sljedeći način. Prvo se odabere neki nasumičan broj K, neka je u našem slučaju K = 3, tada se gleda da li je generacija parna; ako jest onda se uzimaju neparni bitovi jedinke, a ako je pak neparna, onda se uzimaju parni bitovi. Neka u našem slučaju bude parna generacija, nakon čega dijete izgleda:

d1 = ( 4 x 2 x 8 x x x )
d2 = ( 1 x 3 x 5 x x x )

Dalje, počevši od prvog mjesta gdje nije definiran bit, pokušavamo nadopuniti bitove koji nedostaju ako nisu u konfliktu sa već definiranim bitovima, u redoslijedu u kojem su bili prije križanja, nakon čega djeca izgledaju:

d1 = ( 4 x 2 x 8 6 7 x 9 )
d2 = ( 1 x 3 x 5 7 6 9 x )

Na kraju je preostale bitove (d1: 1,3,5; d2: 2,4,8) moguće popuniti na tri načina: po mapiranju, nasumično ili uzlazno sortirajuće. Mi ćemo uzeti uzlazno sortirajuće, nakon čega dobivamo konačnu djecu:

d1 = ( 4 1 2 3 8 6 7 5 9 )

d2 = ( 1 2 3 4 5 7 6 9 8 )

**Matrično križanje –** operator križanja započinje sa djetetom kojemu su svi bitovi resetirani na 0. Najprije ispita dvije roditeljske jedinke, i kada pronađe jednak bit (koji se nalazi na jednakoj poziciji kod oba roditelja) kod oba roditelja, odgovarajući bit postavi kod djeteta. Nakon toga operator naizmjenično prepisuje skupove bitova iz svakog roditelja, dok više ne postoje bitovi kod oba roditelja, da bi se mogli preposati u dijete a da pri tome ne prekrše osnovno pravilo izgradnje jedinke (najviše jedan bit u retku ili stupcu može biti postavljen na 1). Konačno, ako neki redovi kod djeteta ne sadrže bit postavljen na 1, tada se nasumično popune jedinicama. Pošto tradicionalno operator križanja stvara dva djeteta, ovdje se operator izvršava i drugi puta, no sa zamijenjenim pozicijama roditeljima.

Npr. dva roditelja:

$$r1 = \begin{matrix} 0&1&0&0&0\\ 0&0&1&0&0\\ 1&0&0&0&0\\ 0&0&0&0&1\\ 0&0&0&1&0 \end{matrix} \quad i \quad r2 = \begin{matrix} 0&1&0&0&0\\ 0&0&0&1&0\\ 0&0&1&0&0\\ 1&0&0&0&0\\ 0&0&0&0&1 \end{matrix}$$

Prvo se odrede jednaki bitovi, te kopiraju u dijete:

$$\begin{matrix} 0&1&0&0&0\\ 0&0&1&0&0\\ 1&0&0&0&0\\ 0&0&0&0&1\\ 0&0&0&1&0 \end{matrix} + \begin{matrix} 0&1&0&0&0\\ 0&0&0&1&0\\ 0&0&1&0&0\\ 1&0&0&0&0\\ 0&0&0&0&1 \end{matrix} \Rightarrow d1 = \begin{matrix} 0&1&0&0&0\\ 0&0&0&0&0\\ 0&0&0&0&0\\ 0&0&0&0&0\\ 0&0&0&0&0 \end{matrix}$$

nakon čega se naizmjenično kopiraju setovi bitova iz svakog roditelja u dijete, pri tome pazeći na osnovno pravilo kod izgradnje jedinke:

$$d1 = \begin{matrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{matrix}$$

na kraju, ako neki redovi kod djeteta ne sadrže bit postavljen na 1, tada se nasumično popune jedinicama:

$$d1 = \begin{matrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{matrix}$$

Nakon toga se na isti način stvara i drugo dijete, samo što se roditeljima zamijene mjesta. Operatorom križanja se na kraju dobiva dvoje djece:

$$d1 = \begin{matrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{matrix}$$

$$d2 = \begin{matrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{matrix}$$

### 3.4.2 Mutacije

**Obična** – odabiru se dva nasumična bita, unutar jedne jedinke, te se zamijene. Tako bi npr. mutacijom sljedeće jedinke:

j1= (1 2 3 4 5 6 7 8 9)

uz dva nasumična broja, npr. k = 2, r = 5, dobili sljedeću jedinku:

j1 = (1 5 3 4 2 6 7 8 9)

**Uvjetna** – odabire se jedan nasumičan bit, te se zamijeni sa prethodnim. U slučaju kada je bit onaj početni bit, tada se zamjenjuje sa sljedećim. Tako bi npr. mutacijom sljedeće jedinke:

j1= (1 2 3 4 5 6 7 8 9)

uz dva nasumična broja, npr. k = 4, dobili sljedeću jedinku:

j1 = (1 2 4 3 5 6 7 8 9)

**Matrična mutacija –** operator uzme jedinku, nasumično odabere nekoliko redova i stupaca iz te jedinke, makne setove bitova u presjeku tih redova i stupaca, i nasumično ih razmijeni, po mogućnosti u drugačijoj konfiguraciji. Pri zamjeni se mora paziti na pravilo da se po jednom stupcu ili retku može nalaziti najviše jedna kraljica, tj. jedna jedinica.
Npr. mutira se sljedeća jedinka:

$$j1 = \begin{matrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{matrix}$$

i nasumičnim odabirom se odaberu redovi 1, 3 i 4, te stupci 2 i 5, pri čemu je njihov presjek:

$$j1 = \begin{matrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{matrix}$$

nakon čega se ti bitovi nasumično razmijene, te se dobi mutirana jedinka:

$$j1 = \begin{matrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{matrix}$$

### 3.4.3 Selekcije

**Obična** – nasumično se uzmu tri jedinke iz trenutne populacije, odabere se jedinka sa najboljom vrijednošću dobrote, te se stavlja u listu za križanje.

### 3.4.4 Elitizmi

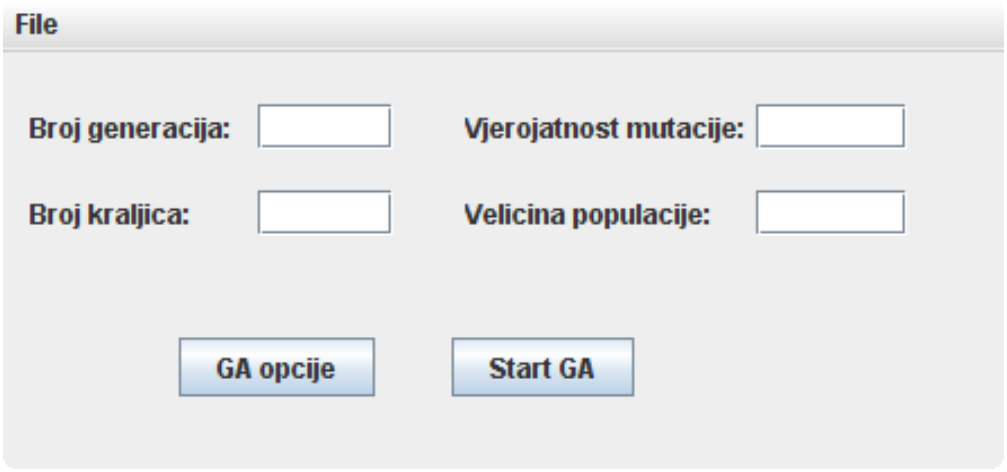
**Uvjetan** – uzima se elitna (najbolja jedinka, tj. jedinka sa najboljom vrijednošću dobrote) jedinka iz prošle generacije i stavlja u populaciju trenutne generacije, samo

onda ako u trenutnoj generaciji ne postoji jedinka sa jednakim ili boljom vrijednošću dobrote, ili ako ne postoji već jednaka jedinka, tj. jedinka sa jednakim rasporedom kraljica kao elitna jedinka.

**Apsolutan** – uzima se elitna (najbolja) jedinka iz prošle generacije i stavlja u populaciju trenutne generacije ako već u trenutnoj generaciji ne postoji jednaka jedinka.

# 4. Grafičko sučelje

Zbog lakšeg upravljanja operatorima genetskog algoritma i lakšeg pregledavanja rezultata istih, je napravljeno jednostavno grafičko sučelje.




**Slika 7** *Osnovni dio grafičkog sučelje programa*

Elementi osnovnog dijela grafičkog sučelja:

- **Broj generacija –** upisuje se broj generacija genetskog algoritma. Ako se odabere opcija SuperTest (koja će biti kasnije objašnjena), tada se ovaj broj generacija ne uzima u obzir.

- **Broj kraljica –** zadaje se broj kraljica, čime se i zapravo određuje kompleksnost samog genetskog algoritma i veličina prostora rješenja. Ako se odabere opcija SuperTest (koja će biti kasnije objašnjena), tada se ovaj broj generacija ne uzima u obzir.

- **Vjerojatnost mutacije –** zadaje se sa kojom će vjerojatnošću doći do mutacije jedinke unutar populacije. Ako se odabere opcija SuperTest (koja će biti kasnije objašnjena), tada se ovaj broj generacija ne uzima u obzir.

- **Veličina Populacije –** zadaje se veličina populacije, tj. koliko će sadržavati jedinki svaka populacija kod izvršavanja genetskog algoritma. Ako se odabere opcija SuperTest (koja će biti kasnije objašnjena), tada se ovaj broj generacija ne uzima u obzir.

- **GA opcije –** pokreće se posebni izbornik gdje se mogu birati operatori genetskog algoritma, ili ako se želi odabrati opcija SuperTest

- **Start GA –** pokreće genetski algoritam sa odabranim postavkama

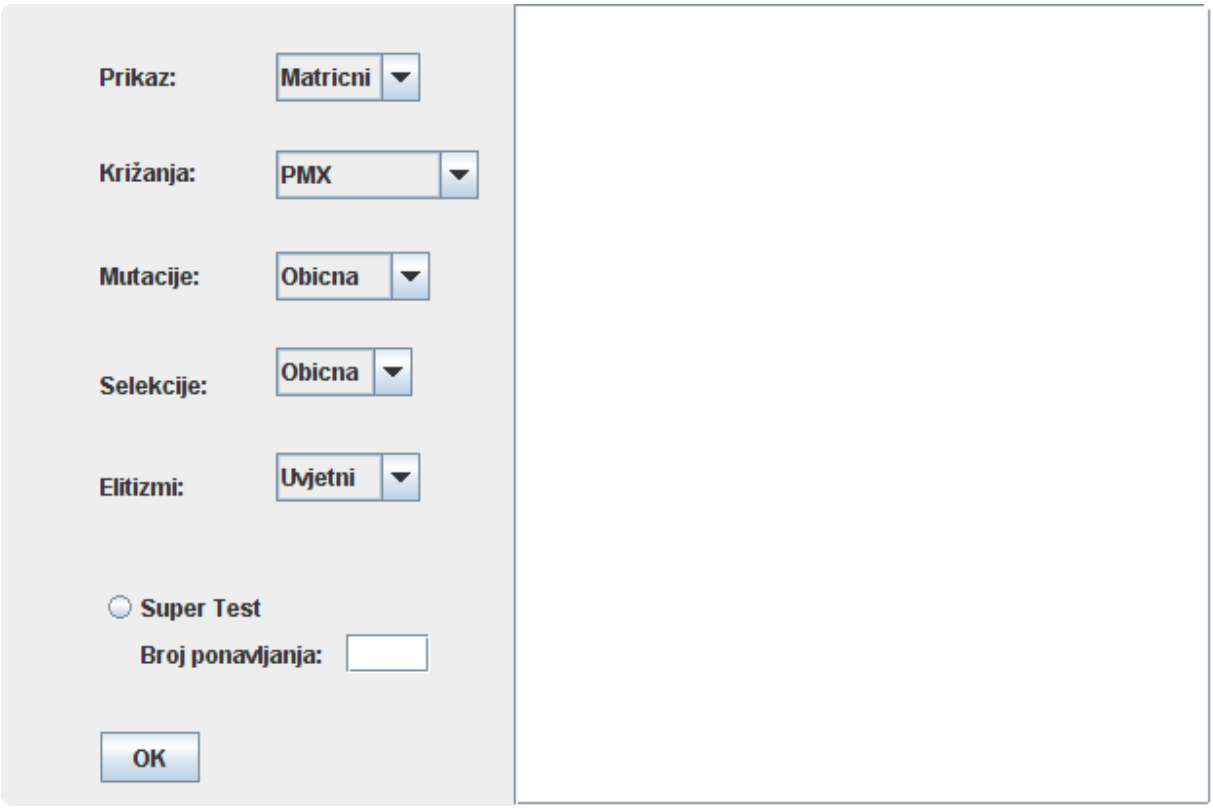
- **File –** glavni izbornik sa sljedećim elementima:

    - **Open Config File –** otvara konfiguracijsku datoteku, te automatski postavlja parametre genetskog algoritma jednake onima iz konfiguracijske datoteke

    - **Open Results –** otvara datoteku sa rezultatima od prošlih testiranja, te ih prikazuje u zasebnom prozoru

    - **Save Config File –** sprema trenutne postavke genetskog algoritma u konfiguracijsku datoteku

    - **Save Results –** sprema u datoteku rezultate zadnje pokrenutog genetskog algoritma




**Slika 8** *Konfiguracijski dio sučelja programa*

Elementi osnovnog dijela grafičkog sučelja:

- **Prikaz –** iz padajućeg se izbornika odabire vrsta prikaza rješenja, tj. jedinki kod genetskog algoritma

- **Križanja –** iz padajućeg se izbornika odabire vrsta križanja za odabrani prikaz rješenja

- **Mutacije -** iz padajućeg se izbornika odabire vrsta mutacija za odabrani prikaz rješenja

- **Selekcije -** iz padajućeg se izbornika odabire vrsta selekcija za odabrani prikaz rješenja

- **Elitizmi -** iz padajućeg se izbornika odabire vrsta elitizma za odabrani prikaz rješenja

- **Super Test –** svi prije uneseni parametri genetskog algoritma se zanemaruju, osim vrste prikaza rješenja i parametra «Broj ponavljanja», te se pokreće genetski algoritam i ukupno izvodi onoliko puta, koliko je zadano kod parametra «Broj ponavljanja», te za svako ponavljanje se pokreće svaka moguća kombinacija(definirano u kodu) parametara križanja, mutacija, selekcija i elitizma, za prikaz permutiranim nizom

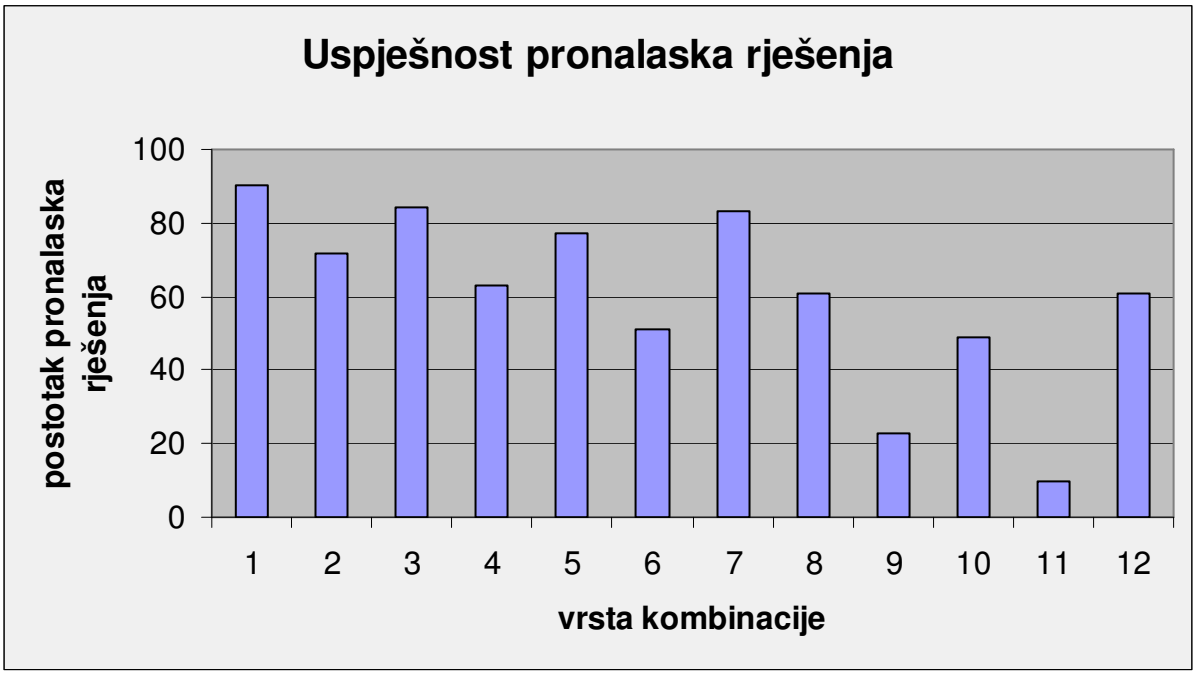
- **Broj ponavljanja –** unosi se broj ponavljanja genetskog algoritma, ukoliko se odabere opcija «Super Test»

U slučaju da netko nije upoznat sa detaljima kako funkcionira neki parametra genetskog algoritma, kod konfiguracijskog dijela grafičkog sučelja je desna strana sačuvana za opis zadnje promijenjenog parametra (na početku je opis prvog po redu u padajućem izborniku za prikaz rješenja).

# 5. Rezultati

## *5.1 Rezultati kombinacija svih operatora genetskog algoritma*

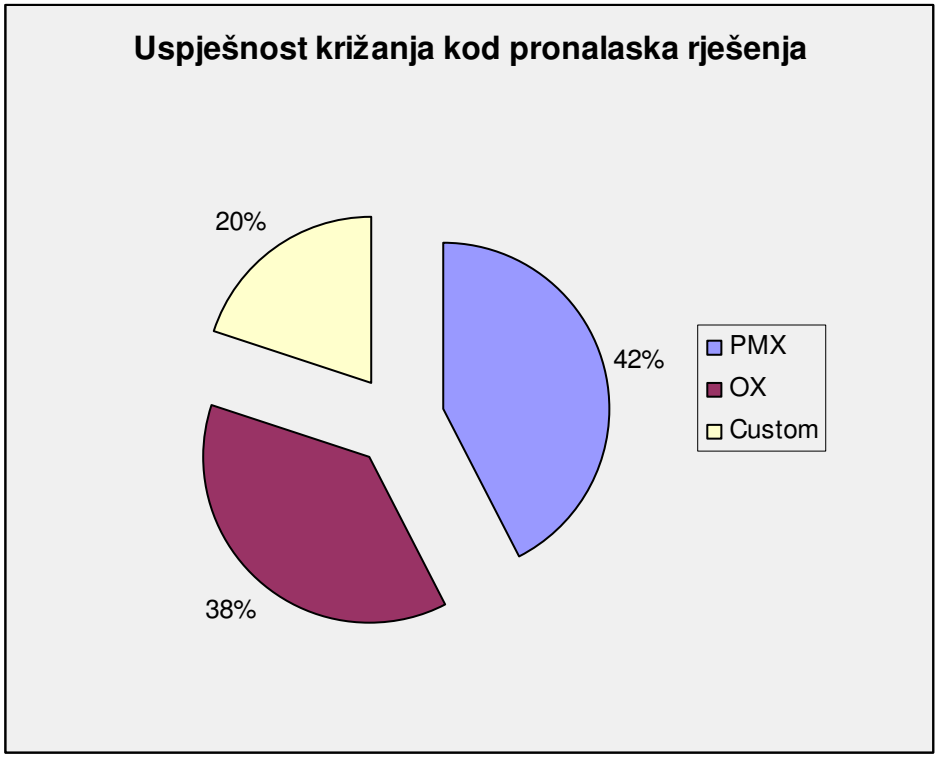
Za testiranje se koristio posebni algoritam, «SuperTest», koji uzima sve postojeće vrste genetskih operatora, stvara 10 (može se i posebno podesiti ovaj parametar u grafičkom sučelju) različitih postavki za broj generacija, vjerojatnost mutacije, broj kraljica i veličinu populacije, te za svaku od tih postavki pokreće sve kombinacije operatora genetskog algoritma i na kraju ispisuje prosjek. Na slici 9 se mogu vidjeti rezultati testiranja za problem od 30 kraljica. Najbolje je prošla kombinacija 1, sa prosjekom od 90% pronalaska rješenja, dok je najlošije prošla kombinacija 11 sa samo prosječnih 10% pronalaska rješenja. Kod smanjenja veličine problema, tj. broja kraljica, smanjuje se i razlika između kombinacija genetskih operatora, dok se kod povećanja broja kraljica i povećava razlika između kombinacija genetskih operatora.




**Slika 9** Uspješnost pronalaska rješenja za pojedinu kombinaciju operatora

| Broj kombinacije | Vrsta Križanja | Vrsta mutacije | Vrsta selekcije | Vrsta elitizma |
|---|---|---|---|---|
| Kombinacija 1: | PMX | obična | obična | uvjetni |
| Kombinacija 2: | PMX | uvjetna | obična | uvjetni |
| Kombinacija 3: | PMX | obična | obična | apsolutni |
| Kombinacija 4: | PMX | uvjetna | obična | apsolutni |
| Kombinacija 5: | OX | obična | obična | uvjetni |
| Kombinacija 6: | OX | uvjetna | obična | uvjetni |
| Kombinacija 7: | OX | obična | obična | apsolutni |
| Kombinacija 8: | OX | uvjetna | obična | apsolutni |
| Kombinacija 9: | Custom | obična | obična | uvjetni |
| Kombinacija 10: | Custom | uvjetna | obična | uvjetni |
| Kombinacija 11: | Custom | obična | obična | apsolutni |
| Kombinacija 12: | Custom | uvjetna | obična | apsolutni |

**Tabela 1** *Kombinacije svih operatora genetskog algoritma*




**Slika 10** *Uspješnost pojedine vrste križanja kod pronalaska rješenja*

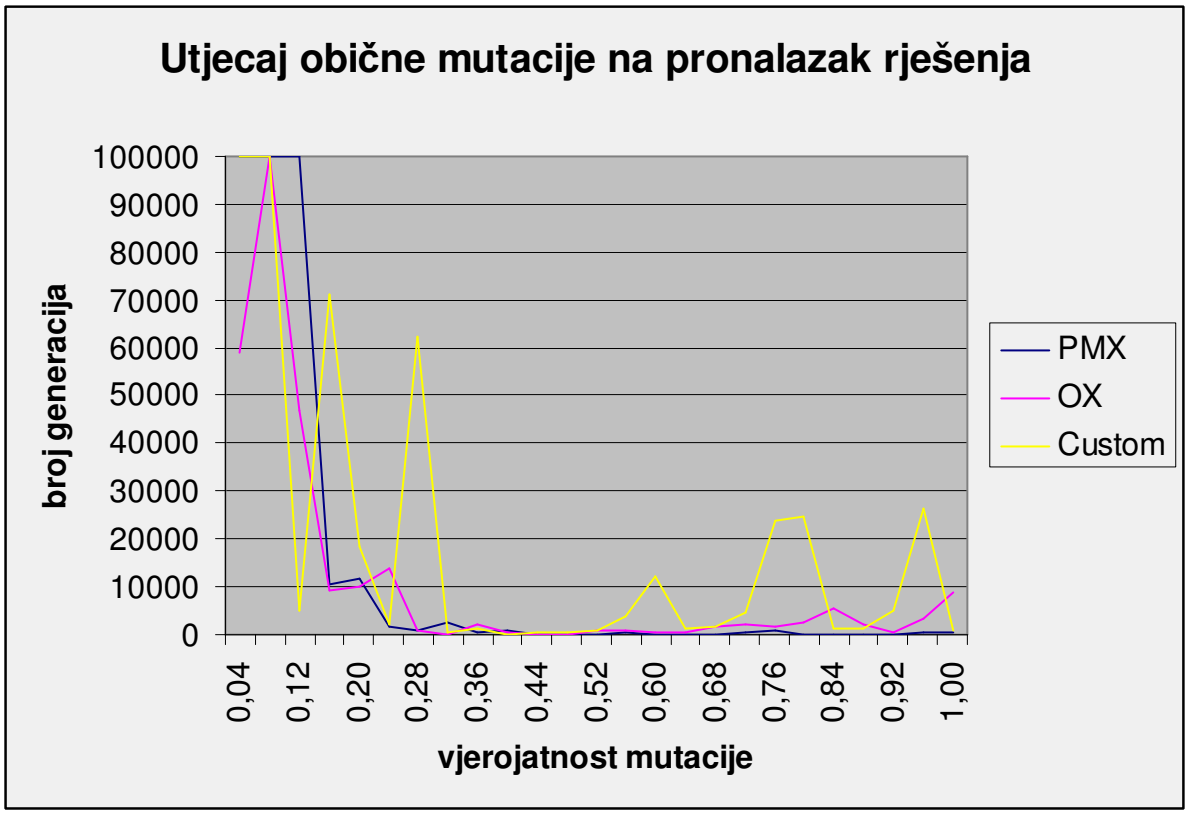
Na slici 10 se može vidjeti uspješnost pronalaska križanja s obzirom samo na vrste križanja, pri čemu najbolje prolazi PMX križanje, sa prosječnih 78% pronalaska rješenja, slijedi ga OX sa 69%, te je najgore prošlo Custom križanje sa 36.5%.

## *5.2 Utjecaj mutacije*


**Slika 11** *Utjecaj mutacije na pronalazak rješenja*

Na slici 11 se može vidjeti prosječni utjecaj mutacije na pronalazak rješenja veličine n = 40. Broj generacija predstavlja broj u kojoj je generaciji pronađeno rješenje za zadani problem. Jasno se može vidjeti da mala vjerojatnost mutacije (manja od 0.3) smanjuje učinak pronalaska rješenja zadanog problema. Najbolje rezultate daje vjerojatnost mutacije vrijednosti 0.5, iako vrijednosti vjerojatnosti mutacije u rasponu od 0.4-0.96 daju vrlo slične rezultate, što bi se moglo objasniti tim da bez mutacije,

genetski algoritam vrlo brzo zaglavi u lokalnom optimumu, te se sve više smanjuje šansa za pronalaskom globalnog rješenja.



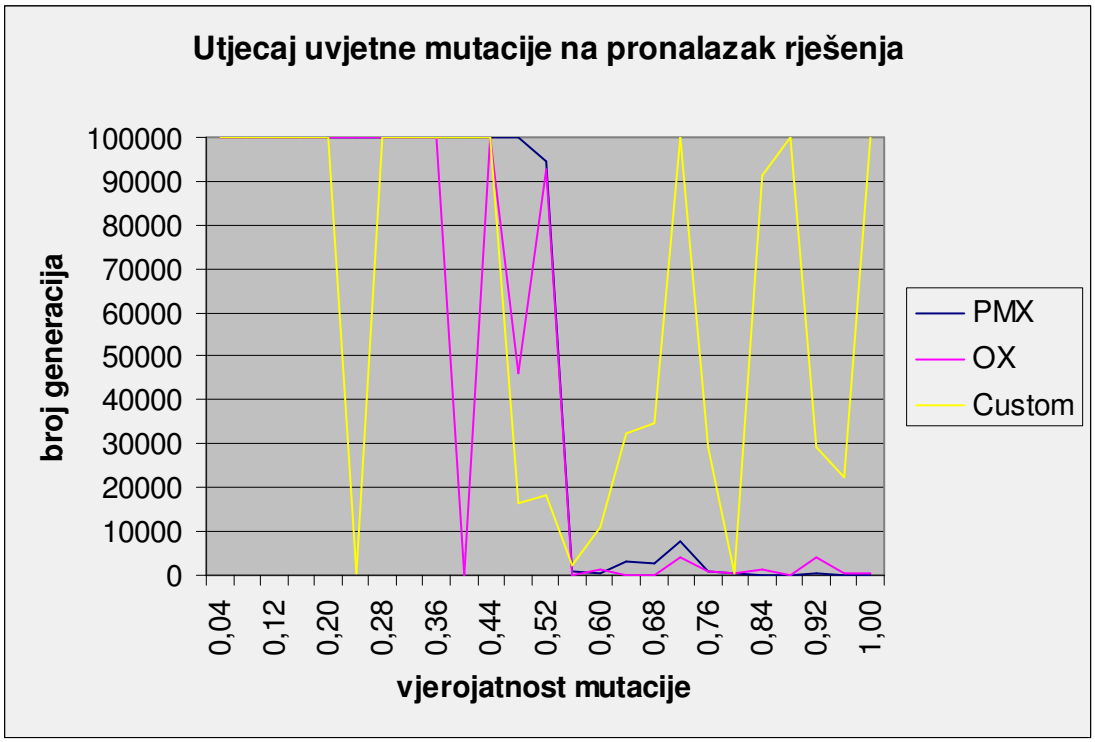

**Slika 12** *Utjecaj obične mutacije kod različitih križanja na pronalazak rješenja*

| Mutacija | 0,04 | 0,20 | 0,44 | 0,60 | 0,80 | 1,00 |
|---|---|---|---|---|---|---|
| PMX | >100000 | 11653 | 28 | 29 | 41 | 409 |
| OX | 59158 | 10251 | 196 | 564 | 2709 | 8603 |
| Custom | >100000 | 18499 | 351 | 12164 | 24814 | 902 |

**Tabela 2** *Utjecaj obične mutacije kod različitih križanja*

Na slici 12 se može vidjeti utjecaj obične mutacije kod pojedinih križanja na pronalazak rješenja. Kao i kod prosječnog utjecaja, može se i ovdje vidjeti da mala vrijednost vjerojatnosti mutacije dopušta genetskom algoritmu da zaglibi u lokalnom optimumu, te tako ne pronađe globalno rješenje, tj. pravo rješenje zadanog problema.

Za PMX i OX križanja se rezultati u velikoj mjeri poklapaju sa dobivenim prosječnim utjecajem mutacije, dok je kod Custom križanja krivulja nešto kaotičnija zbog svoje ovisnosti o parnosti trenutne generacije. Još jedan pokazatelj nekonstantnosti je taj, da u jednom pokretanju je uspio naći rješenje za 255 generacija, dok kod sljedećeg pokretanja, sa identičnom početnom generacijom i postavkama operatora, nije pronašao rješenje unutar granice od 100 000 generacija.




**Slika 13** *Utjecaj uvjetne vjerojatnosti kod različitih križanja na pronalazak rješenja*

| Mutacija | 0,04 | 0,20 | 0,40 | 0,60 | 0,88 | 1,00 |
|---|---|---|---|---|---|---|
| PMX | >100000 | >100000 | >100000 | 376 | 44 | 203 |
| OX | >100000 | >100000 | 114 | 1207 | 79 | 372 |
| Custom | >100000 | >100000 | >100000 | 10892 | >100000 | >100000 |

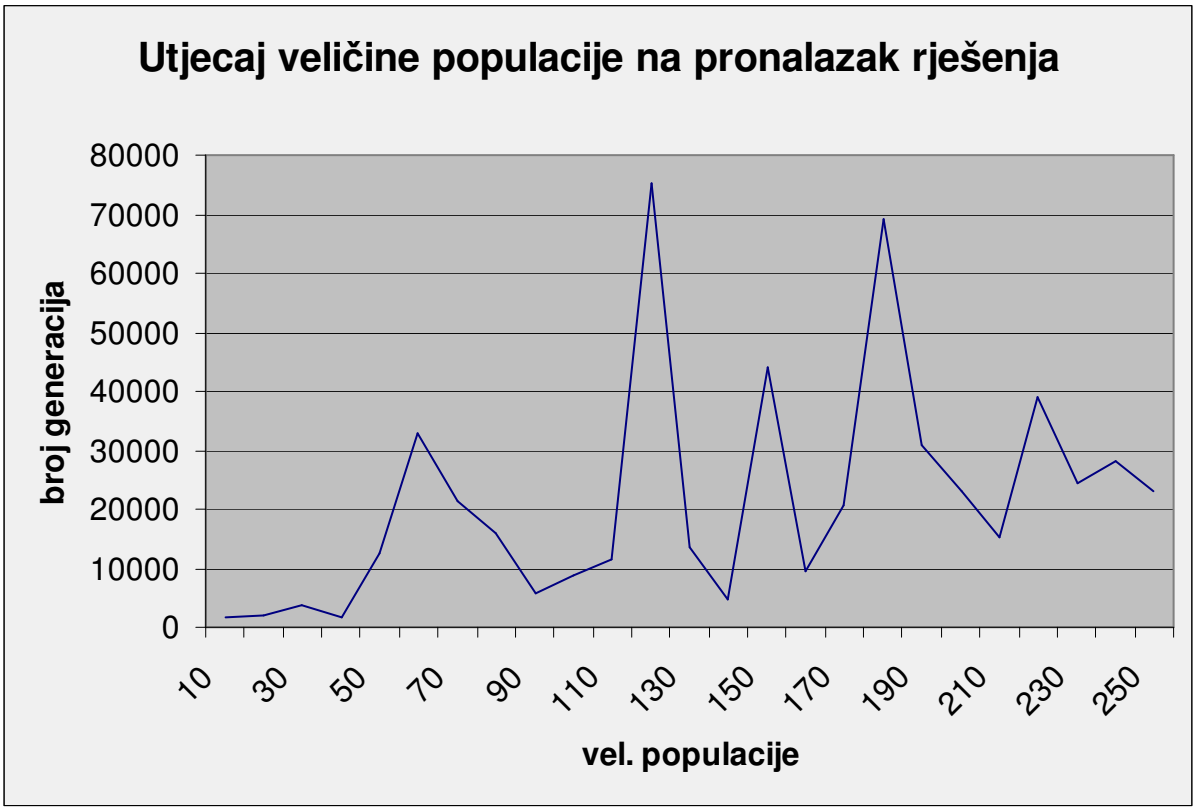
**Tabela 3** *Utjecaj uvjetne vjerojatnosti kod različitih križanja*

Na slici 13 se može vidjeti utjecaj uvjetne mutacije na operatore križanja. Custom je opet kaotičan, iako ovoga puta još više nego kao sa običnom mutacijom. PMX i OX

su od vjerojatnosti 0.58, približno jednako kao i sa običnom mutacijom, ali su zato do vjerojatnosti vrijednosti od 0.58 dosta kaotičniji, i ne pronalaze rješenje unutar ograničenja od 100 000 generacija, jer zaglibe u lokalnom optimumu, tj. dolazi do toga da se jedinke razlikuju u svega par bitova, no tih par bitova uvelike utječe na konačni rezultat. Ovi rezultati još jednom potvrđuju važnost operatora mutacije općenito, a posebno za problem n kraljica.

## *5.4 Utjecaj veličine populacije*

Za testiranje se koristio genetski algoritam sa PMX križanjem, običnom mutacijom i selekcijom, te uvjetnim elitizmom, pri čemu se veličina populacije povećavala sa 10 jedinki na 270 jedinki po populaciji.

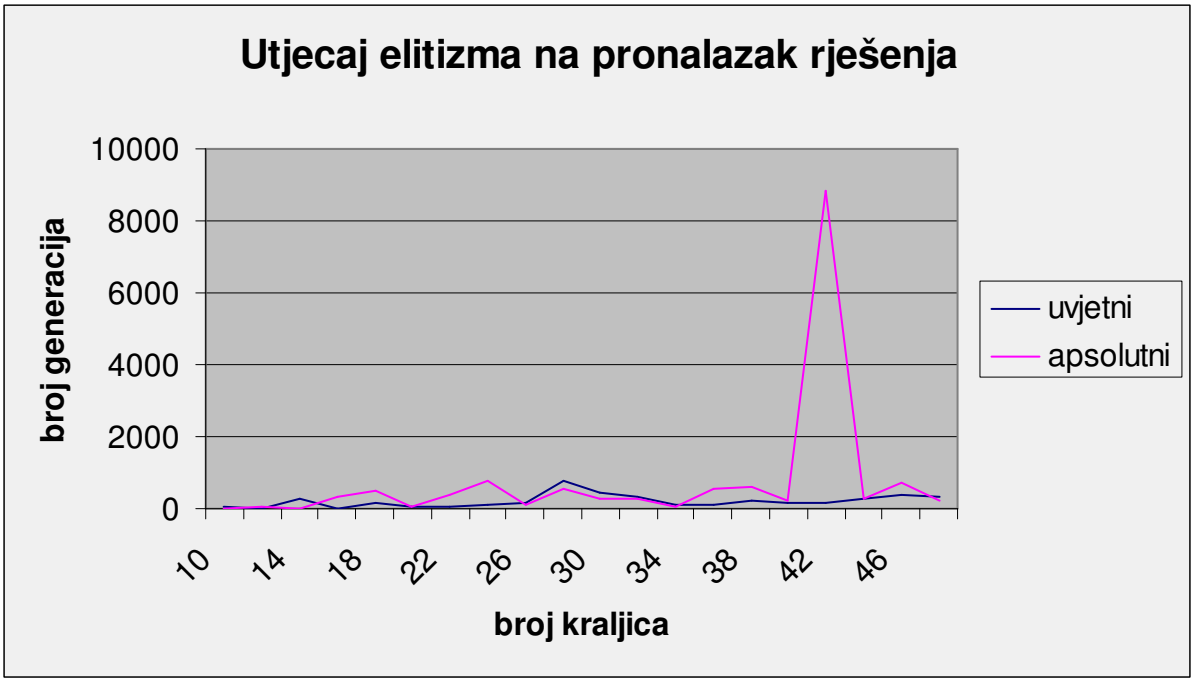



**Slika 14** *Utjecaj veličine populacije na pronalazak rješenja*

Kod promatranja prosječnog utjecaja veličine populacije na broj generacija potrebnim genetskom algoritmu da pronađe rješenje, se dobiju zanimljivi rezultati. Iako se i ovdje jasno vidi nekonzistentnost genetskog algoritma kod broja generacija potrebnih za pronalazak rješenja, jasno se vidi pad efikasnosti algoritma pri povećanju jedinki unutar populacije. Genetskom algoritmu očito više odgovara manja populacija, tj. uz manje populacije puno brže pronalazi rješenje, nego pri većem broju jedinki unutar populacije. Problem je da se kod većih populacija očito srednja vrijednost dobrote brže proširi na ostale jedinke, nego kod manjih populacija, te ne dopušta veću raznolikost među jedinkama, što za posljedicu ima veći broj generacija potrebnih za pronalazak rješenja.

### *5.3 Razlika između operatora elitizma*

Za testiranje se koristio genetski algoritam sa PMX križanjem, običnom mutacijom i selekcijom. Broj kraljica se povećavao sa 10 do 48, dok je veličina populacije od 20 jedinki ostala konstantna.



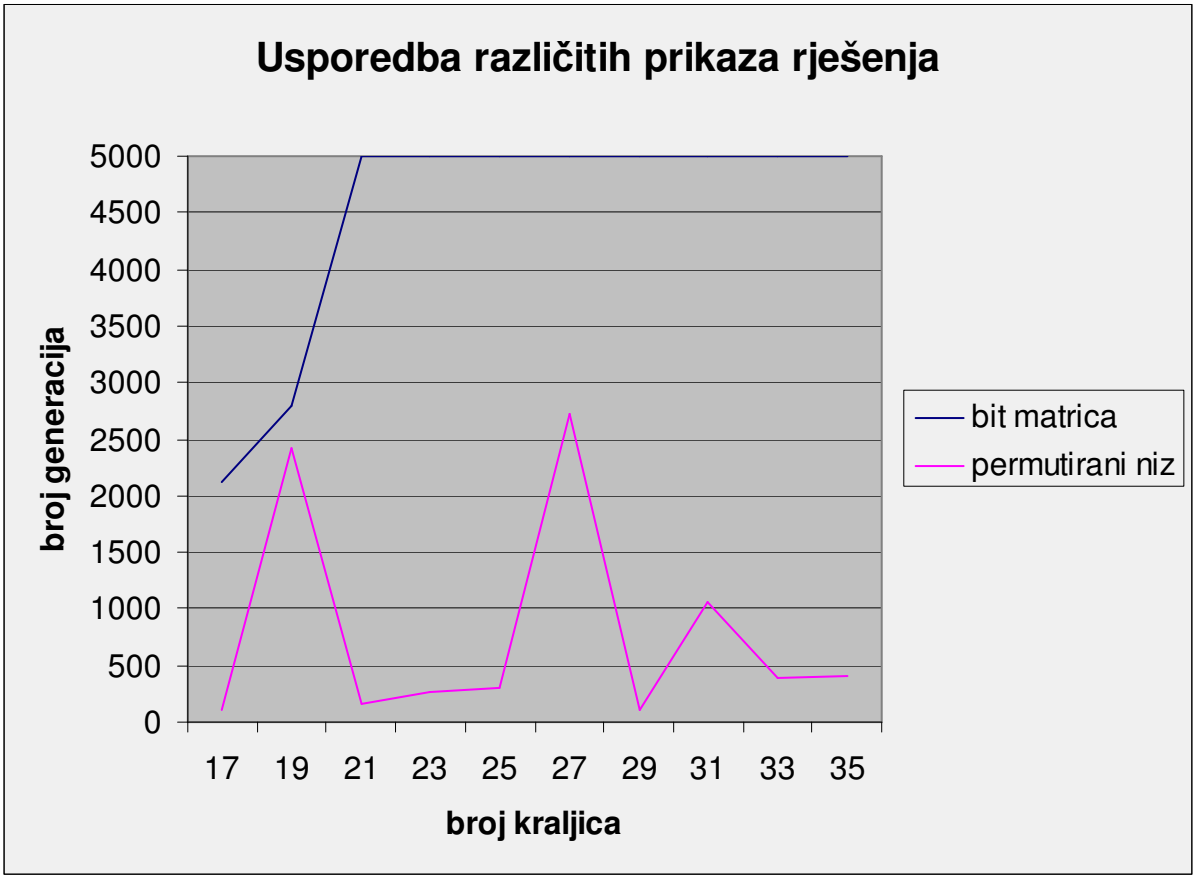

**Slika 15** *Utjecaj elitizma na pronalazak rješenja*

Sa slike 15 se jasno vidi da su operatori elitizma vrlo podjednaki kod utjecaja na genetski algoritam pri pronalasku rješenja za problem N kraljica. Ovaj «skok» kod

apsolutnog elitizma, nije toliko posljedica samog operatora, već više posljedica stohastičke prirode genetskog algoritma.

## *5.4 Razlika između različitih prikaza rješenja*

Za testiranje se koristio genetski algoritam sa PMX križanjem, običnom mutacijom i selekcijom, te uvjetnim elitizmom, dok se kod bit matrice koristio uvjetni elitizam, sa ostalim genetskim operatorima implementiranim samo za taj prikaz. Broj kraljica se uzeo iz raspona [17, 35], koji se jednoliko povećavao kod testiranja.




**Slika 156** *Usporedba različitih prikaza rješenja*

| Vrsta prikaza: | N = 21 | N = 23 | N = 25 |
|---|---|---|---|
| Bit matrica | > 5000 | > 5000 | > 5000 |
| Permutirani niz | 163 | 2728 | 393 |

**Tabela 4** *Usporedba različitih prikaza rješenja*

Kako se može vidjeti sa slike 16 i tabele 4, prikaz rješenja permutiranim nizom daje mnogo bolje rezultate kod problema N kraljica, nego prikaz bit matricom. Prikaz bit matricom ne daje nikakve prednosti spram ostalih prikaza rješenja, baš suprotno, puno je teže prilagođavati operatore tom prikazu rješenja, jer promjenom samo jednog bita se dobiva jedinka nedopuštenih karakteristika.

# 6. Zaključak

Svrha ovog rada je bila proučiti učinkovitost različitih operatora genetskog algoritma i skupova parametara na tipičnom kombinatoričkom problemu, problemu N kraljica. Kao što je i poznato, genetski algoritam ne jamči da će svakim svojim pokretanjem pronaći rješenje zadanom problemu. Ta nepredvidljivost genetskog algoritma je pod velikim utjecajem operatora mutacija i križanja, što se može i vidjeti iz rezultata eksperimentiranja sa različitim operatorima i skupinama parametara. Za testiranje genetskog algoritma uzeta su dva prikaza rješenja, s permutiranim nizom i bit matricom, tri različita operatora križanja i dva različita operatora mutacije za permutirani niz, po jedan operator križanja i mutacija za bit matricu, te dva različita operatora za selekciju i elitizam, koja su se iskoristila na oba prikaza rješenja. Najboljim rješenjem pokazale su se dvije kombinacije, ona sa PMX križanjem, običnom mutacijom, običnom selekcijom i uz oba operatora elitizma (uvjetni i apsolutni), uz prikaz rješenja permutiranim nizom.

Ukupni rezultati genetskog algoritma su zadovoljavajući, bez obzira što postoje neke druge metode koje mogu brže pronaći rješenje za zadani problem N kraljica(kao što je brzi algoritam kojeg su izradili Rok Sosic i Jun Gu, a koji se može primijeniti na problem sa milijunima kraljica [3]), zbog toga što genetski algoritam radi slijepo pretraživanje prostora rješenja (koji se kod ovog problema eksponencijalno povećava s povećanjem broja kraljica), bez nekog ugrađenog specifičnog znanja, već samo uz pomoć svojih genetskih operatora. Jedino što ga je usmjeravalo ka rješenju problema N kraljica je bila vrijednost dobrote pojedine jedinke unutar populacije.

# 7. Literatura


[1] Marin Golub, M.G.., «Genetski algoritam», prvi dio, http://www.zemris.fer.hr/~golub/ga/ga_skripta1.pdf, 12.04.2008

[2] Wikipedia, «Genetic algorithm», http://en.wikipedia.org/wiki/Genetic_algorithm, 12.04.2008.

[3] Rok Sosic, Jun Gu, R.S., J.G., «A Polynomial Time Algorithm for the N-Queens Problem», Department of Computer Science, University of Utah, 1990.

[4] Abdollah Homaifar, Joseph Tumer, Samia Ali, A.H., J.T., S.A., «The N-queens problem and genetic algorithms», Machine Intellifence Laboratory, Department of Electrical Engineering, North Carolina A&T State University, 1992

[5] Marin Golub, M.G., «Paralelni genetski algoritmi», http://www.zemris.fer.hr/~golub/ga.html, 2004

[6] Vedran Lovrečić, diplomski rad, «Podešavanje parametara genetskog algoritma», 2006

[7] Wikipedia, «NP problem», http://en.wikipedia.org/wiki/NP_problem, 12.02.2008.

[8] Michalewicz, Z, "Genetic Algorithms + Data Structures = Evolutionary Programs", Springer-Verlag, Berlin, 1992.

# Rješavanje problema N kraljica uz pomoć genetskog algoritma


## Sažetak:

Problem n-kraljica je klasičan kombinatorički problem u području umjetne inteligencije. Pošto problem ima jednostavnu, regularnu strukturu, i inherentne je kompleksnosti, korišten je za stvaranje mjernih programa za algoritme pretraživanja umjetne inteligencije. Problem 4-kraljica je najjednostavniji primjer problema n-kraljica, za koji postoji rješenje. Potrebno je postaviti četiri kraljica na šahovsku ploču veličine 4x4, tako da se nijedan par kraljica ne može međusobno napasti. To znači da nijedan par kraljica se ne može nalaziti u istom redu ili stupcu ili na istoj dijagonali. U generalnom problemu se treba postaviti N kraljica na šahovsku ploču veličine NxN, tako da si nijedan par kraljica ne može napadati.

Za rješavanje ovog NP problema je potreban algoritam, koji je efikasan u pretrazi i optimizacijskim okolinama, isto tako mora biti sposoban zadovoljiti ograničenja problema. Ne postoji niti jedan algoritam polinomne složenosti da riješi NP-težak problem, ali postoje neke nedeterminističke metode koje omogućuju rješavanje NP problema u polinomijalnom vremenu. Jedna od takvih metoda je genetski algoritam, no mora se upamtiti da je genetski algoritam samo aproksimativan, tj. ne garantira pronalaženje rješenja zadanog problema.

Svrha ovog rada je proučiti učinkovitost različitih operatora genetskog algoritma i skupova parametara na problemu N kraljica. Konkretno, implementirana su dva različita prikaza rješenja (preko permutiranog niza i bit matrice), zatim tri različita operatora križanja (PMX, OX, Custom) i dva različita operatora mutacije (običan, uvjetan) za prikaz rješenja permutiranim nizom, po jedan operator križanja i mutacije za bit matricu, te po dva operatora za elitizam i jedan za selekciju, koja su primijenjena na oba prikaza rješenja.



**Ključne riječi:** umjetna inteligencija, genetski algoritam, genetski operatori, kombinatorički problem, problem N kraljica

# Solving the N queens problem with genetic algorithm


## Summary:

The N queens problem is a classical combinatorial problem in the artificial intelligence. Since the problem has a simple and regular structure, and inherent complexity, it has been widely used as a benchmark for search algorithms in artificial intelligence. The 4 queens problem is the simplest case of the N queens problem with a solution. The problem is to place four queens on a 4x4 chessboard so that no two queens can capture each other. This means that no two queens can occupy the same row, the same column, or the same diagonal. In the general case, the set of N queens is to be placed on a NxN chessboard so that no two queens can attack each other.

The N queens problem requires an algorithm that is efficient in search and optimization environments. and also one that is able to work with constraint satisfaction problems. There is no algorithm with polynomial complexity that can solve a NP-complete problem, but there are some non-deterministic methods that can solve a NP-complete problem in polynomial time. One of these methods is a genetic algorithm, but it has to be kept in mind, that this algorithm is only approximative, which means it doesn't guarantee that it will find a solution every time it's used.

The purpose of this study is to analyze different genetic operators and sets of parameters on the N queens problem. There are two different representations implemented(bit-matrix and path representation), three different crossovers(PMX, OX, Custom) and two different mutations(ordinary, conditional) for the path representation, one crossover and mutation for bit-matrix representation, and two different kinds of elitism, and one selection that are used for both representations.



**Keywords:** artificial intelligence, genetic algorithm, genetic operators, combinatorial problem, the N queens problem