UPUTSTVO ZA IRACE NA LINUX – U

Jovana Rađenović

Rad nadgledale:

dr Tatjana Davidović

dr Tatjana Jakšić Krüger

Beograd, 2020.

Sadržaj

Cilj uputstva

Cilj ovog uputstva je da se ubrza i pojednostavi upoznavanje sa irace-om. Napravljeno je jer je mnogima teško da se snađu sa oficijalnim uputstvom za irace (videti [1]). Preporučujemo da se nakon proučavanja ovog uputstva prouči [1]. Takođe, oficijalno uputstvo je podložno čestim dopunama, stoga se preporučuje praćene istih na [2]. Napominjemo da ovo uputstvo ne pokriva rad na Windows operativnom sistemu. Za pitanja ili uočene probleme u radu irace-a, upućujemo na forum koji autori irace-a redovno prate (videti [3]). Takođe, napominjemo da je u toku izrada korisničkog interfejsa pod nazivom "Irace Studio" koji će dodatno olakšati korišćenje ovog softvera.

Trenutna verzija

Trenutno korišćena verzija irace-a je **verzija 3.4.1**.

Zadatak irace – a

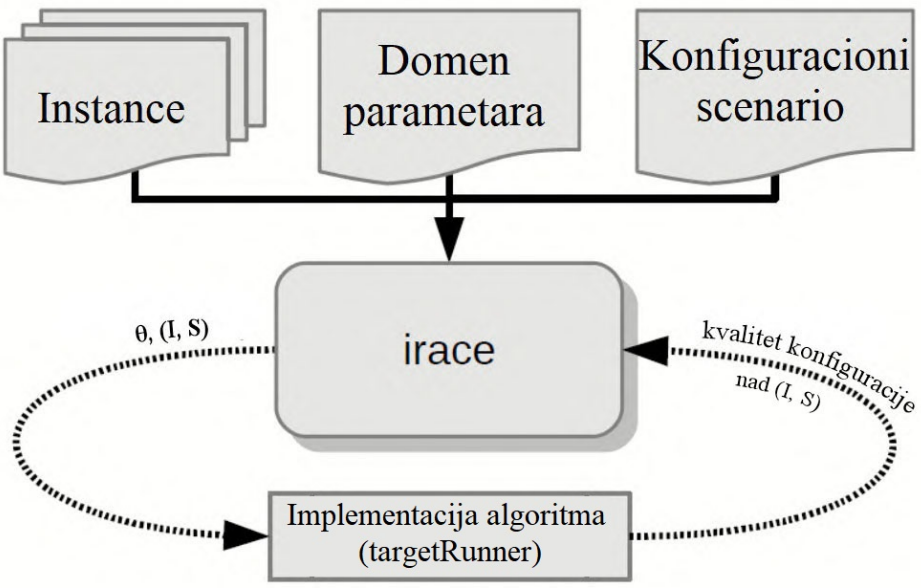
Irace služi za automatsko nameštanje parametara željenog algoritma. Na osnovu poznatih podataka, kao što su instance, domen parametara i slično, irace treba da odredi najpovoljnije vrednosti parametara za dati algoritam. Jednu kombinaciju vrednosti parametara, datog algoritma, nazivaćemo jednom konfiguracijom. Odnosno ukoliko posmatrani algortam ima npr. 2 parametra, *parametar1* koji može da uzme vrednosti iz segmenta [2, 10] i *parametar2* koji uzima vrednosti iz segmenta [3, 40], kombinacija *parametar1* = 5, *parametar2* = 7 predstavlja jednu konfiguraciju.

Način rada irace-a

Kao što se može videti sa slike 1, za rad irace-a neophodno je obezbediti implementaciju *algoritma* čije je parametre potrebno namestiti (nazivaćemo je targetRunner), *domen* za parametre koji se nameštaju, *instance*, i konfiguracioni *scenario* koji podrazumeva zadavanje opcija koje preciznije određuju način rada irace-a. Irace se oslanja na koncepte koji su prvi put uvedeni u mašinskom učenju. Bazira se na procedurama **iteriranih trka**, a sam naziv irace skraćenica je od "iterated racing". Rad irace-a se sastoji od faze treniranja i faze testiranja. Tokom **faze treniranja** sprovodi se određeni broj iteracija. Na početku svake **iteracije** bira se skup konfiguracija, na osnovu domena parametara, koje će se ispitivati, odnosno trkati, u datoj iteraciji. Pod **trkanjem** podrazumeva se poređenje kvaliteta konfiguracija, kako bi se eliminisale one konfiguracije koje daju statistički lošije rezultate. Cilj je da se izlazne vrednosti targetRunner-a **minimizuju**. Kada bi se poređenje kvaliteta konfiguracija vršilo nad jednim parom (*instanca*, *seme*)[1] (uloga semena će biti naknadno objašnjena), najbolja, odnosno najkvalitetnija konfiguracija, bi bila ona za koju se dobija najmanja vrednost na izlazu pri izvršavanju

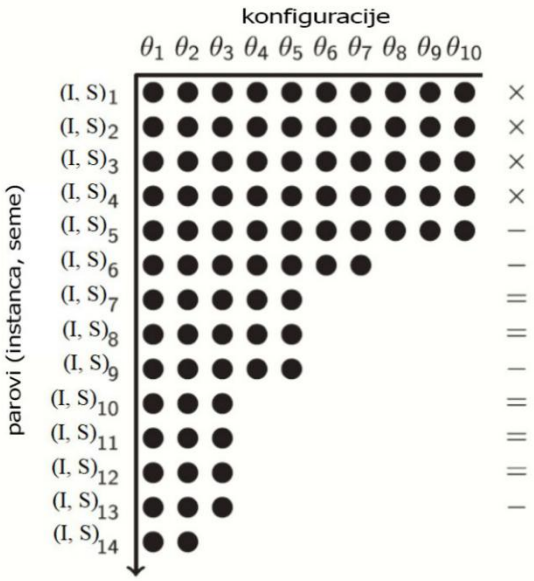
[1] Ukoliko je u pitanju deterministički algoritam, ne posmatraju se parovi (instanca, seme), već samo instance. Nema smisla da se za istu konfiguraciju, više puta izvršava deterministički algoritam, nad istom instancom. O ovome će biti više reči u narednim sekcijama.

targetRunner-a nad datim parom (*instanca*, *seme*). Stoga ćemo izlaznu vrednost targetRunner-a, koji na ulazu dobija jednu konfiguraciju i jedan par (*instanca*, *seme*), nazivati kvalitet zadate konfiguracije nad datim parom (*instanca*, *seme)*[2]. Kako se ispitivanje vrši nad više parova (*instanca*, *seme*), za upoređivanje kvaliteta konfiguracija koristi se statistički test. Poređenje dobijenih rezultata upotrebom odgovarajućeg statističkog testa vrši se nakon što se sve konfiguracije ispitaju nad *firstTest* brojem parova (*instanca*, *seme*). Konfiguracije sa kojima su se ostvarili lošiji rezultati se odbacuju, a trka se nastavlja. Nakon prvog statističkog testa svaki sledeći vrši se nakon što se sve konfiguracije, koje su i dalje u trci, ispitaju nad *eachTest* parova (*instaca*, *seme*). Vrednosti za *firstTest, eachTest*, kao i vrsta statističkog testa samo su neke od opcija koje se zadaju kroz konfiguracioni *scenario*. Grafički prikaz jedne trke dat je na slici 2. Irace sprovodi fazu treniranja, odnosno sprovodi iteracije, sve dok ima dovoljno budžeta na raspolaganju. **Budžet** se zadaje ili kao dozvoljeno vreme za rad ili kao broj poziva targetRunner-a koje je dozvoljeno izvršiti. Korisnik je dužan da zada budžet za rad irace-a, kroz konfiguracioni *scenario*. Ukupan budžet se raspoređuje na iteracije. Trka u okviru iteracije se prekida ukoliko nema dovoljno budžeta u trenutnoj iteraciji da se trka nastavi, kada u trci ostane ne više od *minNbSurvival* konfiguracija ili ukoliko se dostigne *elitistLimit*. Zadavanje budžeta, kao i opcije *minNbSurvival* i *elitistLimit* objašnjene su u sekciji "Opis mogućih opcija za scenario.txt fajl". Po završetku faze treniranja, irace ispisuje konfiguracije koje su dale najbolje rezultate. Tokom **faze testiranja** irace testira najbolju konfiguraciju (ili više najboljih, ukoliko se tako zada kroz konfiguracioni scenario) dobijenu tokom faze treniranja, nad test instancama. Ova faza nije obavezna i sprovodi se samo ukoliko zadamo test instance kroz konfiguracioni *scenario* na koji ćemo u nastavku referisati kao scenario fajl. Detaljnije o radu irace-a može se naći u [4] i [5].



Slika 1: *Shema protoka informacija. θ predstavlja jednu konfiguraciju, dok (I, S) predstavlja jedan par (instanca, seme).*

[2] Izlazna vrednost targetRunner-a u [1] se naziva "cost value of the evaluation".



Slika 2: *Grafički prikaz jedne trke. Svaka tačka označava izlaznu vrednost targetRunner-a (kvalitet jedne konfiguracije nad jednim parom (instanca, seme)), kome su kao ulazne vrednosti prosleđeni odgovarajuća konfiguracija i par (instanca, seme). ‘ ×’ označava da nije sproveden nijedan statistički test, odnosno konfiguracije nisu poređene međusobno, ‘ −’ označava da se statističkim testom odbacila barem jedna konfiguracija, ‘ = ’ označava da se statističkim testom nije odbacila nijedna konfiguracija.*

Instalacija

Irace je implementiran u R-u, stoga je neophodno prvo njega instalirati. R je programski jezik, i okruženje, specijalizovano za statističku obradu i grafičko prikazivanje podataka. Neke opcije koje se mogu zadati irace-u potrebno je implementirati kao funkcije u R-u. Te opcije se retko koriste, stoga u većini slučajeva nije neophodno imati predznanje vezano za rad u R-u kako bi se koristio irace. Međutim, ukoliko korisnik želi da analizira podatke i vrednosti koje irace generiše tokom svog rada, osnovno poznavanje R-a će omogućiti lakše snalaženje.

Postupak instalacije:

1. Instalacija R-a
   U terminalu otkucati sledeću komandu:
   ```
   sudo apt-get install r-base
   ```
   Ova komanda će proći u većini slučajeva. Ukoliko komanda ne prodje pogledati [6].
2. Instalacija irace-a
   U terminalu izvršiti komande:
   - `R` (ovom komandom pokreće se R u terminalu)
   - `install.packages("irace")`

- `library("irace")` (učitavanje paketa; ovo je potrebno pozvati svaki put kada iznova otvorimo R okruženje a želimo da radimo sa irace paketom. Alternativa je komanda: `require("irace")`)
- `system.file(package = "irace")` (ovim ćemo dobiti putanju do mesta na kome je irace instaliran; putanja koja je vraćena, biće nam potrebna u narednoj sekciji)
- `q()` (izlazak iz R okruženja)

Pravljenje neophodnih veza nakon instalacije

U narednoj komandi umesto `kate` može se ukucati naziv bilo kog editora ili `nano` da se fajl otvori direktno u terminalu. U terminalu otkucati sledeću komandu: `kate ~/.bashrc`

Na kraj `.bashrc` fajla dodati:

`export IRACE_HOME =` "ovde, pod znacima navoda, navesti putanju koju smo dobili u prethodnom koraku (u sekciji "Instalacija") pomoću system.file"

```
export PATH = ${IRACE_HOME}/bin/:$PATH
```

sačuvati i zatvoriti fajl. Ovo radimo da bismo mogli da pozivamo irace gde god da smo pozicionirani. Ova promena nije vidljiva u trenutnoj sesiji stoga treba otvoriti novi terminal. Ovaj postupak smo sproveli korektno, ukoliko sa proizvoljne putanje možemo uspešno da izgenerišemo uputstvo: `irace --help`

Neophodni fajlovi i direktorijumi za irace

Da bi se pravilno pokrenuo irace potrebno je kreirati radni direktorijum u kome će se nalaziti implementacija algoritma čije parametre nameštamo, domeni parametara, instance nad kojima će irace ispitivati algoritam u cilju odabira najpogodnijih vrednosti za parametre algoritma i slično.

Napraviti direktorijum. U direktorijumu napraviti sledeće:

- fajl "scenario.txt"
- direktorijum "Instances"
- fajl "instances.txt"
- fajl "parameters.txt"
- "target-runner" (implementacija algoritma čije parametre želimo da podesimo)

Opcioni fajlovi:

- "testing.txt" (ukoliko želimo da se sprovede faza testiranja)
- "configurations.txt" (ukoliko želimo da navedemo inicijalne konfiguracije)
- "forbidden.txt" (ukoliko želimo da zabranimo uzimanje određenih konfiguracija u obzir)

- "target-evaluator" (program, ili R funkcija, koji omogućava odlaganje procene kvaliteta konfiguracija)

Skrećemo pažnju na to da su "scenario.txt", "Instances", "parameters.txt", "target-runner" nazivi koje irace očekuje. Ukoliko se koriste drugi nazivi to treba navesti u fajlu "scenario.txt". O tome smo dali detaljnije u narednim sekcijama. Takođe, u direktorijumu čija je putanja $IRACE_HOME, gde je $IRACE_HOME putanja dobijena u sekciji "Instalacija", postoje direktorijum "examples" u kojem su nam dostupni neki primeri, i direktorijum "templates" u kojem su nam dostupni šabloni za prethodno spomenute fajlove.

scenario.txt

Fajl "scenario.txt" se koristi da bi se izlistali svi fajlovi koje će irace da koristi. Ukoliko ovaj fajl nismo nazvali "scenario.txt", nego npr. "nasFajl.txt", irace ćemo pozvati iz komandne linije sa: `irace --scenario nasFajl.txt`. Na ovaj način reći ćemo irace–u da kao scenario fajl posmatra "nasFajl.txt". Takođe, u scenario fajlu mogu se navesti opcije kojima možemo imati veću kontrolu nad proverama koje irace radi (npr. možemo reći irace–u sa koliko konfiguracija da započne rad, koji statistički test da koristi itd). U nastavku teksta ćemo dati više detalja o spomenutim opcijama. Usput će se navoditi pojedine od njih, a nakon opisa svih potrebnih fajlova biće dat kompletan prikaz svih opcija, u sekciji "Opis mogućih opcija za scenario.txt fajl". Potrebno je imati u vidu da putanje koje se navode kao relativne, su relativne, u odnosu na direktorijum u kome se nalazi scenario fajl.

Instances, instances.txt i testing.txt

U direktorijum "Instances" sačuvati sve instance problema, koje će se koristiti. U fajlu "instances.txt" potrebno je nabrojati instance koje irace treba da koristi za treniranje, odnosno instance na osnovu kojih bi irace trebalo da odredi najbolju (viđenu[3]) konfiguraciju parametara za naš algoritam. U fajlu "testing.txt" potrebno je navesti instance problema koje će se koristiti u fazi testiranja. U fajlovima "instances.txt" i "testing.txt" instance se navode jedna ispod druge. Jedan red je jedna instanca. Instancu možemo navesti tako što ćemo direktno navesti njene vrednosti, ili se može navesti fajl u kome se instanca nalazi. Ukoliko navodimo direktno vrednosti instanci u fajlovima "instances.txt" i "testing.txt", direktorijum "Instances" nam nije potreban. Na primer, neka imamo dve instance koje želimo da navedemo u "instances.txt". Neka prvu instancu čine vrednosti 5 i 4, a dugu vrednosti 4, 5 i 6. Tada u "instances.txt" možemo navesti:

```
5 4
4 5 6
```

[3] Kažemo najbolju viđenu konfiguraciju, jer je ona najbolja od konfiguracija koje su se ispitivale. Dakle, možda postoji bolja od nje koja nije bila posmatrana.

Umesto toga možemo da napravimo dva fajla u direktorijumu "Instances", u kojima ćemo čuvati ove instance. Neka su to "instanca1.txt" i "instanca2.txt". Sadržaj fajla "instanca1.txt" bi u ovom slučaju bio: `5 4`. Sadržaj fajla "instanca2.txt" bi bio: `4 5 6`. U "instances.txt" bismo naveli:

```
instanca1.txt

instanca2.txt
```

U "scenario.txt" staviti (po potrebi):

```
# ovo je komentar
## NAPOMENA: opcije za koje zelimo da koristimo podrazumevane
## vrednosti, nije potrebno navoditi u fajlu scenario.txt

# trainInstancesDir - direktorijum u kome se nalaze instance za
# trening
# Podrazumevana vrednost: "./Instances"
trainInstancesDir = "./Instances"

# trainInstancesFile - fajl u kome su nabrojane instance koje treba
# uzeti za trening
# Podrazumevana vrednost je "".
trainInstancesFile = "./instances.txt"


## Ukoliko je navedeno svojstvo trainInstancesDir instance iz
## trainInstancesFile ce se traziti u tom direktorijumu, a ukoliko
## trainInstancesFile opcija nije navedena, a trainInstancesDir
## jeste, tada ce se za trening uzeti sve instance iz
## trainInstancesDir direktorijuma.

# Odgovarajuce opcije za test instance. Razlika je sto
# testInstancesDir, za razliku od trainInstancesDir, ima ""
# kao podrazumevanu vrednost.
testInstancesDir = "./Instances"
testInstancesFile = "./testing.txt"
```

parameters.txt

U ovom fajlu potrebno je navesti sve parametre našeg algoritma za koje želimo da irace predloži vrednosti. Svaka vrsta u fajlu "parameters.txt" predstavlja specifikacije vezane za jedan parametar. Za svaki parametar navodimo:

naziv oznaka tip domen uslov(opciono)

naziv se odnosi na naziv parametra.

oznaka je string koji se šalje komandnoj liniji zajedno sa vrednošću parametra. Ima ulogu prekidača komandne linije koji se koristi u cilju prosleđivanja vrednosti odgovarajućeg parametra. Vrednost za *oznaku* se navodi pod navodnicima, i potrebno je staviti jedan razmak na kraj, jer se *oznaka*

konkatenira sa vrednošću parametra. Dakle, ukoliko želimo da *oznaka* za neki parametar bude na primer --ozn, kao *oznaku* ćemo navesti "--ozn ", a ne "--ozn". Značaj ove napomene biće prikazan na primeru u sekciji "target-runner".

tip opisuje prirodu parametra. Moguće vrednosti za *tip* su: *i*, *r*, *o*, *c*. Pri čemu:

- *i* je od integer (ako parametar treba da uzima samo celobrojne vrednosti)

- *r* je od real (ako parametar treba da uzima realne vrednosti)

- *c* je od categorical (domen ovakvog parametra je skup stringova)

- *o* je od ordinal (podtip kategoričkih; domen je isti kao za kategoričke ali predstavlja skup koji je uređen po nekom kriterijumu)

domen govori irace-u iz kog skupa može da uzima vrednosti za posmatrani parametar. Za celobrojne i realne parametre *domen* se navodi kao interval oblika (*donja_granica*, *gornja_granica*) kao moguće vrednosti parametra irace uzima vrednosti iz zadatog segmenta (gornja i donja granica su uključene u domen). Za kategoričke i ordinalne parametre *domen* se navodi kao skup stringova oblika (*vr1*, *vr2*, *vr3*, "*vr4 sadrzi razmak*"), navodnike je neophodno navesti samo ako vrednost koja se navodi sadrži beline ili zapete. **Ne postoji ograničenje** za veličinu domena. Ukoliko ne znamo koji je domen najbolje uzeti za dati parametar, preporučuje se zadavanje većeg domena, međutim ukoliko znamo optimalan domen dobro je suziti fokus na njega.

uslov se odnosi na to da li se posmatrani parametar uzima u obzir ili ne, na osnovu vrednosti nekih drugih parametara. Zapisuje se tako što se navede | a zatim logički izraz zapisan R sintaksom. Ukoliko je povratna vrednost izraza TRUE posmatrani parametar se uzima u obzir u trenutno posmatranoj konfiguraciji, ukoliko je FALSE posmatrani parametar se ignoriše. Na primer, ukoliko imamo celobrojni parametar *p1* koji može da uzme vrednosti od 1 do 5, i parametar *p2* koji želimo da se uzme u obzir samo ako je vrednost za *p1* manja od 3, u vrstu u kojoj definišemo *p2* stavićemo:

```
| p1 < 3
```

U "scenario.txt" dodati:

```
# Podrazumevana vrednost: "./parameters.txt"
parameterFile = "./parameters.txt"
```

target-runner

Algoritam čije parametre nameštamo se može implementirati kao funkcija u R-u ili kao program u nekom drugom programskom jeziku. Spomenutu implementaciju nazivaćemo **targetRunner**, dok je prethodno spomenut "target-runner", podrazumevana vrednost za naziv samog fajla. U nastavku će se razmatrati slučaj kada je targetRunner program. Ukoliko korisnik ipak želi da implementira algoritam kao funkciju u R-u, više informacija može naći u sekciji 5.2.2 u [1]. Pod targetRunner programom podrazumevamo **izvršnu verziju**. Dakle, ako je algoritam implementiran npr. u C++ potrebno je prvo

izvršiti kompilaciju. Neka smo npr. sačuvali izvornu verziju koda našeg programa pod nazivom simulirano_kaljenje.cpp. U terminalu možemo izvršiti sledeće:

```
g++  simulirano_kaljenje.cpp  -o  simulirano_kaljenje.o
```

Ovime smo izvršili kompilaciju i izvršni kod našeg programa smo sačuvali pod nazivom "simulirano_kaljenje.o". Za targetRunner ćemo koristiti "simulirano_kaljenje.o". Ukoliko naknadno menjamo program ne smemo zaboraviti da ga ponovno kompajliramo, pre pokretanja irace-a. Potrebno je namestiti da targetRunner na standardni izlaz ispisuje **samo jednu ili dve brojevne vrednosti**. Od ovoga se odstupa ako su nam za procenu kvaliteta jedne konfiguracije potrebne informacije za sve konfiguracije koje se ispituju nad jedinim parom (*instanca*, *seme*), kao što je slučaj kod više-ciljnih algoritama. Više reči o ovome će biti u sekciji "target-evaluator", a više detalja o upotrebi irace-a za nameštanje parametara kod više-ciljnih algoritama, može se naći u sekciji 10.2 u [1]. targetRunner na ulazu dobija jednu konfiguraciju i jedan par (*instanca*, *seme*), o čemu će više reči biti u nastavku ove sekcije. Ukoliko targetRunner namestimo tako da na izlaz ispisuje **samo jednu vrednost**, jedan broj, tu vrednost nazivamo kvalitet zadate konfiguracije nad datim parom (*instanca*, *seme)*. Cilj je da se izlazna vrednost minimizuje. Stoga, za datu vrednost možemo, na primer, uzeti vrednost funkcije cilja, ukoliko rešavamo neki problem minimizacije. Irace određuje najbolje konfiguracije na osnovu statističkih testova i kvaliteta konfiguracija nad parovima (*instanca*, *seme*) viđenih do trenutka izvršavanja testa. Skrećemo pažnju na to da je na izlaz potrebno ispisati samo brojevnu vrednost, stoga nije korektno da se stavi ispis "Vrednost resenja je: " pa broj, već treba ispisati samo taj broj. Ukoliko targetRunner namestimo tako da na izlaz ispisuje **dve vrednosti**, dva broja, prva vrednost će se posmatrati kao prethodno spomenut kvalitet konfiguracije nad odrađenim parom (*instanca*, *seme*), dok će se druga vrednost posmatrati kao vreme izvršavanja targetRunner-a. U ovom slučaju ispis na izlaz treba da bude oblika "kvalitetKonfiguracije vreme". Drugu vrednost, odnosno vreme, obavezno je ispisati u slučaju da se koristi opcija *maxTime* i/ili je opcija *capping* aktivirana. Spomenute opcije se zadaju kroz scenario fajl, o čemu će biti više reči u nastavku.

Prilikom ispitivanja kvaliteta jedne konfiguracije nad jednim parom (*instanca*, *seme*) irace u pozadini pokreće targetRunner, tako što ga poziva iz komandne linije uz dodatne argumente:

ID_konfig ID_inst seme instanca ograničenje(opciono) konfiguracija

ID_konfig predstavlja identifikacioni broj kofiguracije koja se trenutno ispituje.

ID_inst je identifikacioni broj instance nad kojom se trenutno radi.

seme predstavlja vrednost koju ćemo preuzeti kroz naš targetRunner program, i koristiti u našem programu kao seme pri pozivu funkcije koja generiše slučajne vrednosti.

instanca predstavlja apsolutnu putanju do instance, ili samu instancu ukoliko smo u "instances.txt" direktno navodili vrednosti za instance, a ne fajlove u kojima se one mogu naći.

ograničenje se odnosi na vremensko ograničenje za vreme izvršavanja targetRunner programa pri trenutnom pozivu. Prisutno je samo ukoliko je postavljeno svojstvo *boundMax* u fajlu "scenario.txt".

konfiguracija predstavlja niz *oznaka* i vrednosti parametara jedne konfiguracije. Za svaki parametar date konfiguracije prvo se prosleđuje njegova *oznaka*, pa zatim njegova vrednost u datoj konfiguraciji. Parametri se prosleđuju redom kojim su navedeni u "parameters.txt".

Kako bi irace korektno radio potrebno je da kroz naš targetRunner program obradimo prethodno spomenute **argumente komandne linije**. Jedan konkretan primer poziva targetRunner programa koji vrši irace bio bi sledeći:

```
home/user/target-runner 1 12 1234567 home/user/Instances/instanca1.tsp --param1 12 --param2 9.7
```

U datom primeru vrši se nameštanje dva parametra algoritma implementiranog kroz targetRunner. Potrebno je da u našem targetRunner programu preuzmemo 6. argument komandne linije kao vrednost prvog parametra i 8. argument kao vrednost drugog parametra. Takođe, treba proslediti *instancu* na adekvatan način, kao i *seme* (ukoliko nam je potrebno).

Prilikom objašnjavanja strukture "parameters.txt" fajla napomenuto je da je *oznaka* parametra string koji treba da sadrži razmak na kraju. Da smo u "parameters.txt" kao *oznaku* prvog parametra stavili "--param1", bez razmaka na kraju, komandnoj liniji bi u prethodom primeru bilo prosleđeno "--param112" a ne "--param1 12".

U "scenario.txt" dodati:

```
# Podrazumevana vrednost: "./target-runner"
targetRunner = "./target-runner"
```

configurations.txt

Kao što je naznačeno ovaj fajl je opcioni. U njemu se mogu navesti inicijalne konfiguracije, što nam omogućava da obezbedimo kvalitetne početne konfiguracije (ukoliko su nam poznate neke) za fazu treniranja. Inicijalne konfiguracije podrazumevaju one konfiguracije kojima irace započinje prvu iteraciju. Ukoliko navedemo manje konfiguracija nego što je potrebno za prvu iteraciju (potrebno je *nbConfigurations* konfiguracija), dodatne konfiguracije će biti odabrane uniformno. Osvrnimo se sada na strukturu samog "configurations.txt" fajla. U prvoj vrsti fajla pišemo nazive parametara (uzimamo one nazive koje smo naveli u fajlu "parameters.txt"). Jedna kolona je jedan parametar, dok jedna vrsta odgovara jednoj konfiguraciji parametara algoritma. Ukoliko neki parametar nije korišćen u nekoj konfiguraciji, kod njega će umesto konkretne vrednosti stajati NA. Skrećemo pažnju na to da inicijalne konfiguracije moraju biti u skladu sa ograničenjima postavljenim kroz "parameters.txt" i "forbidden.txt".

Primer sadržaja "configurations.txt" fajla:

```
param1 param2 mode real mutation
5        NA   "x2" 2.0   "low"
```

Ukoliko se koristi "configurations.txt" u "scenario.txt" dodati:

```
# Podrazumevana vrednost je "".
configurationsFile = "./configurations.txt"
```

forbidden.txt

Ovaj fajl je opcioni. U svakom redu fajla navodi se logički izraz zapisan R sintaksom. Na početku svake iteracije irace vrši odabir konfiguracija koje će se ispitivati u trenutnoj iteraciji. Pre početka ispitivanja odabranih konfiguracija proverava se da li neka od odabranih konfiguracija zadovoljava neki od izraza koje smo zadali kroz "forbidden.txt" fajl. Ukoliko za neku konfiguraciju parametara povratna vrednost nekog od tih izraza bude TRUE, ta konfiguracija se odbacuje. Primeri validnih logičkih operatora koji se mogu koristiti u fajlu su: `==`, `!=`, `>=`, `>`, `<`, `&`, `|`, `!` i `%in%`.

Primer sadržaja "forbidden.txt" fajla :

```
(param1 < 3) & (param2 == 0.0)
```

U ovom slučaju ukoliko prvi parametar odabrane konfiguracije ima vrednost manju od 3, a drugi parametar ima vrednost 0, tada će vrednost logičkog izraza datog u primeru, za posmatranu konfiguraciju, biti TRUE, te se posmatrana konfiguracija odbacuje i bira se druga.
Skrećemo pažnju na to da se pri poređenju kategoričkih i ordinalnih parametara vrši leksikografsko poređenje, jer se kategorički i ordinalni parametri posmatraju kao stringovi. U slučaju da napomenute parametre želimo da poredimo kao numeričke vrednosti, možemo koristiti `as.numeric`. Neka smo npr. u fajlu "parameters.txt" definisali neki parametar sa:

```
a     "--a "     c     (0, 2, 3, 5)
```

Dakle, `a` je kategorički parametar koji može uzeti vrednosti `"0"`, `"2"`, `"3"` i `"5"`, a čija je *oznaka* `"--a "`. Uslov `a > 10` će imati vrednost TRUE za `a = 5`, jer se poređenje vrši leksikografski. Kako bismo kategorički parametar `a` poredili kao broj, možemo iskoristiti konverziju `as.numeric(a)`.

U "scenario.txt" dodati (ukoliko se koristi "forbidden.txt"):

```
# Podrazumevana vrednost je "".
forbiddenFile = "./forbidden.txt"
```

target-evaluator

Obično targetRunner vraća broj koji je rezultat poziva našeg programa za razmatranu konfiguraciju i par (*instanca*, *seme)*. Taj broj smo do sada nazivali kvalitetom date konfiguracije nad datim parom (*instanca*, *seme)*. Postoje slučajevi kada želimo da se procena kvaliteta odloži dok u jednoj trci[4] ne proverimo sve konfiguracije nad jednim parom (*instanca*, *seme*). Ovo se radi ukoliko su nam za procenu kvaliteta jedne konfiguracije potrebne informacije za sve konfiguracije koje se testiraju nad tim parom (*instanca*, *seme*). Dakle, targetEvaluator omogućava odlaganje procene kvaliteta konfiguracija. Retko se koristi, stoga ga nećemo detaljnije razmatrati. Za više informacija pogledati sekciju 5.3 u [1].

[4] Jedna trka podrazumeva poređenje konfiguracija u toku jedne iteracije.

U scenario.txt dodati (ukoliko se koristi target-evaluator):

```
# Podrazumevana vrednost je "".
targetEvaluator = ""
```

Opis mogućih opcija za scenario.txt fajl

Ovde ćemo dati kompletan spisak opcija za scenario fajl. U fajl je potrebno uvrstiti samo opcije koje su od interesa. Za sve opcije koje ne navedemo irace će koristiti podrazumevane vrednosti.

Opšte opcije:

opcija: **`execDir`**
opis: Direktorijum u kome se pokreće targetRunner program. Podrazumevana vrednost je direktorijum u kome se nalazi scenario.txt.
podrazumevana vrednost: "./"
sintaksa: `execDir = "./"`

opcija: **`logFile`**
opis: Fajl u kome će biti sačuvani rezultati koji se dobijaju tokom rada irace-a. Način na koji je moguće čitati sadržaj ovog fajla dat je u sekciji "Čitanje rezultata iz logFile-a". Kao vrednost za *logFile* potrebno je navesti ili apsolutnu putanju željenog fajla ili relativnu u odnosu na *execDir*. Ukoliko ne navedemo ovu opciju irace će kreirati fajl "irace.Rdata" u kome će beležiti rezultate.
podrazumevana vrednost: "./irace.Rdata"
sintaksa: `logFile = "./irace.Rdata"`

opcija: **`debugLevel`**
opis: Nivo debagovanja. Postaviti na 0 da se ne prikazuju poruke vezane za debagovanje. Veći brojevi obezbeđuju detaljnije poruke.
podrazumevana vrednost: 0
sintaksa: `debugLevel = 0`

opcija: **`seed`**
opis: Broj koji će se koristiti kao seme za generator (pseudo)slučajnih brojeva, u okviru targetRunner-a. Zadaje se pozitivan ceo broj. Ukoliko se kao vrednost stavi NA, "" ili NULL, biće generisano proizvoljno seme.
podrazumevana vrednost: NA
sintaksa: `seed = NA`

opcija: **`repairConfiguration`**
opis: Funkcija u R-u koju sami implementiramo. Uzima novogenerisanu konfiguraciju i pravi određene izmene nad njom (one koje mi definišemo). Dakle, izmene se vrše pre evaluacije konfiguracije i pre provere ograničenja iz *forbiddenFile*. Funkcija treba da primi tri

argumenta. Prvi argument je *data.frame*[5] koji se sastoji od jednog skupa vrednosti za parametre (odnosno jedne konfiguracije), a kome su nazivi kolona nazivi parametara.
Primer:
```
param1    param2    param3
  1         8         NA
```
Drugi argument je lista parametara. Može se iskoristiti R funkcija `readParameters` u obliku: `parameters <- readParameters(file = "parameters.txt").` Treći argument je opcija *digits*.
Primer: Naredni kod razmešta parametre jedne konfiguracije, tako da budu poređani rastuće.
```
repairConfiguration = function (configuration, parameters, digits)
{
      columns <- c("p1","p2","p3")
      # cat("Before"); print(configuration)
      configuration[columns] <- sort(configuration[columns])
      # cat("After"); print(configuration)
      return(configuration)
}
```
podrazumevana vrednost: `""`
sintaksa: `repairConfiguration = ""`

opcija: **`postselection`**
opis: Nakon svih iteracija moguće je uraditi trku najboljih konfiguracija iz svake iteracije. Opcija *postselection* se odnosi na procenat budžeta koji će se koristiti za tu trku. Nešto više detalja o ovoj opciji može se naći u sekciji 10.10 u [1].
podrazumevana vrednost: 0
sintaksa: `postselection = 0`

opcija: **`aclib`**
opis: Omogući/onemogući AClib mod. Ova opcija omogućava kompatibilnost sa GenericWrapper4AC (videti [9]) kao targetRunner skript. Ukoliko `aclib=1`, *digits* će biti postavljeno na 15.
podrazumevana vrednost: 0
sintaksa: `aclib = 0`

Opcije vezane za elitizam:

opcija: **`elitist`**
opis: Omogući/onemogući elitizam. U svakoj iteraciji beleže se konfiguracije koje su se najbolje pokazale tokom trke u toj iteraciji, a među kojima ne postoji statistički značajna razlika. Kada je elitizam omogućen N^{elite} od napomenutih konfiguracija se proglašavaju elitnim. U i-toj iteraciji važi $N_i^{elite}=\min\{N^{min}, N_i^{surv}\}$, gde je N^{min} broj koji odgovara vrednosti koja se zadaje pomoću opcije *minNbSurvival, a* N_i^{surv} je broj konfiguracija koje su žive nakon trke u datoj iteraciji. Elitne konfiguracije se prenose u narednu iteraciju, i na osnovu njihovih vrednosti biraju se vrednosti za nove konfiguracije. Statističke testove je moguće vršiti u bilo kom trenutku trke, zahvaljujući svojstvima *firstTest* i *eachTest*. Elitne konfiguracije se mogu odbaciti ukoliko se pokažu kao loše u trenutnoj iteraciji, međutim, nije dozvoljeno odbaciti

[5] Tabela ili struktura nalik dvodimenzionom nizu, u kojoj svaka kolona sadrži vrednosti jedne promenljive, a svaka vrsta sadrži jedan skup vrednosti svih promenljivih.

elitne konfiguracije pre nego što se sve neelitne konfiguracije u tekućoj iteraciji ne ispitaju nad svim parovima (*instanca*, *seme*) nad kojima su tokom prethodnih iteracija ispitivane elitne konfiguracije.

podrazumevana vrednost: 1
sintaksa: `elitist = 1`

opcija: **`elitistNewInstances`**

opis: Broj novih parova (*instanca*, *seme*) koji će biti dodati u svakoj iteraciji. Ispitivanje konfiguracija u tekućoj iteraciji se najpre vrši nad novim parovima, a zatim se koriste parovi viđeni u prethodnim iteracijama. Moguće je da se neće proći svi prethodni parovi u trenutnoj iteraciji, usled nedostatka budžeta, ili ukoliko se na primer dostigne ograničenje zadato preko svojstva *elitistLimit* ili svojstva *minNbSurvival*. Ukoliko se prođu svi novi i svi stari parovi (*instanca*, *seme*), a budžet i ograničenja to dozvoljavaju, irace će dodati još novih parova za posmatranje u tekućoj iteraciji. Kako se elitne konfiguracije prenose u narednu iteraciju, nije ih potrebno ponovo ispitivati nad parovima (*instanca*, *seme*) nad kojima su one već ispitane, odnosno nije potrebno da se opet poziva targetRunner već se u statističkim testovima koriste ranije izračunate vrednosti. Opciju koristiti samo ako `elitist=1`.

podrazumevana vrednost: 1
sintaksa: `elitistNewInstances = 1`

opcija: **`elitistLimit`**

opis: Maksimalan broj uzastopnih statističkih testova koje je dozvoljeno izvršiti u jednoj trci, a da nije došlo do eliminacije nijedne konfiguracije, nakon što su svi stari parovi (*instanca*, *seme*) viđeni u trenutnoj trci. Ukoliko se limit dostigne trenutna trka se zaustavlja. Dakle ako je limit 2 to znači da ukoliko su izvršena dva uzastopna testa takva da ni u jednom od njih nije došlo do eliminacije nijedne konfiguracije, trenutna trka se zaustavlja, ali samo ukoliko su u trenutnoj trci viđeni svi parovi (*instanca*, *seme*) koji su se pojavljivali u svim prethodnim iteracijama. Upotrebiti 0 da nema limita. Opciju koristiti samo ako `elitist=1`.

podrazumevana vrednost: 2
sintaksa: `elitistLimit = 2`

Ukoliko je `elitist=0` parovi (*instanca*, *seme*) se ne prenose iz prethodnih iteracija u naredne, već se u svakoj iteraciji posmatraju novi parovi.

Interne irace opcije:

opcija: **`nbIterations`**

opis: Ukoliko se koristi podrazumevana vrednost ove opcije, irace računa **minimalan** broj iteracija N^{iter} koje će se izvršiti, kao $N^{iter} = \lfloor 2 + \log_2 N^{param} \rfloor$, gde je N^{param} broj parametara koje podešavamo pomoću irace-a. U zavisnosti od rezultata koji se ostvaruju tokom faze treniranja, moguće je da će irace izvršiti više od N^{iter} iteracija. U slučaju da se zada vrednost različita od podrazumevane za ovu opciju, ta vrednost će predstavljati **maksimalan** broj iteracija koje irace može da izvrši. Od budžeta i broja konfiguracija koje je potrebno ispitati u svakoj iteraciji zavisi da li će se izvršiti tačno zadati broj iteracija. Postavljanje ove opcije na vrednost različitu od podrazumevane može dovesti do toga da irace stane sa radom ranije nego što bi inače, što može prouzrokovati lošije rezultate. Strogo se preporučuje korišćenje podrazumevane vrednosti.

podrazumevana vrednost: 0

sintaksa: `nbIterations = 0`

opcija: **`nbExperimentsPerIteration`**

opis: Broj izvršavanja targetRunner programa pri jednoj iteraciji irace-a. Ukoliko se koristi podrazumevana vrednost, ovaj broj varira iz iteracije u iteraciju i zavisi od indeksa iteracije i preostalog budžeta. Više detalja dato je u [5]. Preporučuje se korišćenje podrazumevane vrednosti.

podrazumevana vrednost: 0

sintaksa: `nbExperimentsPerIteration = 0`

opcija: **`sampleInstances`**

opis: Ukoliko je `sampleInstances=1` instance za fazu treniranja se uzimaju proizvoljnim redosledom. Ukoliko želimo da irace uzima instance baš redom kojim smo ih naveli u *trainInstancesFile* , odnosno u *trainInstancesDir* u slučaju da nismo prosledili fajl kroz *trainInstancesFile,* onda treba postaviti `sampleInstances=0`. Uzimanje instanci redom može prouzrokovati pristrasnost ka instancama neke grupe. Za primer videti tekst iznad slike 4.

podrazumevana vrednost: 1

sintaksa: `sampleInstances = 1`

opcija: **`minNbSurvival`**

opis: Ukoliko broj opstalih konfiguracija u trci nije veći od *minNbSurvival*, tekuća iteracija se prekida i prelazi se na sledeću. Ukoliko se koristi podrazumevana vrednost, ovaj broj se računa kao $\lfloor 2 + \log_2 N^{param} \rfloor$, gde je N^{param} broj parametara koje podešavamo pomoću irace-a.

podrazumevana vrednost: 0

sintaksa: `minNbSurvival = 0`

opcija: **`nbConfigurations`**

opis: Broj konfiguracija koje će se razmatrati u svakoj iteraciji (ako je broj npr. 30, u svakoj iteraciji će se ispitivati (trkati) po 30 konfiguracija). Ukoliko se koristi podrazumevana vrednost, za svaku iteraciju se računa broj konfiguracija posebno, na osnovu svojstva *nbExperimentsPerIteration*, indeksa iteracije i svojstva *mu*. Više detalja dato je u [5]. Preporučuje se korišćenje podrazumevane vrednosti.

podrazumevana vrednost: 0

sintaksa: `nbConfigurations = 0`

opcija: **`mu`**

opis: Parametar koji se koristi za određivanje broja konfiguracija koje će se razmatrati u iteraciji. Broj konfiguracija se računa tako da bude dovoljno budžeta u trci da se ispitaju sve konfiguracije na najmanje $\mu + \min(5, j)$, (*instanca*, *seme*) parova, gde je j indeks trenutne iteracije. Vrednost za μ će biti prilagođena tako da nije manja od vrednosti za *firstTest*. Preporuka je koristiti podrazumevanu vrednost, a ukoliko je potrebno, menjati *firstTest* i *eachTest* umesto *mu*.

podrazumevana vrednost: 5

sintaksa: `mu = 5`

opcija: **softRestart**
opis: Omogući/onemogući "soft restart" strategiju koja izbegava preuranjenu konvergenciju probabilističkog modela. Kada je novoodabrana konfiguracija "slična" roditeljskoj konfiguraciji, nad probabilističkim modelom ovih konfiguracija se vrši "soft restart". Spomenuta sličnost se određuje preko Hemingovog rastojanja[6] kod kategoričkih i ordinalnih parametara, odnosno preko *softRestartThreshold* opcije ukoliko su u pitanju numerički parametri. Za više informacija o ovoj metodi pogledati [5].
podrazumevana vrednost: 1
sintaksa: `softRestart = 1`

opcija: **softRestartThreshold**
opis: Definiše dozvoljenu "sličnost" numeričkih parametara. Ukoliko su odabrane vrednosti u nekoj konfiguraciji "slične" sa roditeljskom konfiguracijom izvršiće se "soft restart" (ako je omogućena opcija *softRestart*). Ukoliko ne navedemo ovu opciju a koristimo opciju *softRestart*, ili postavimo *softRestartThreshold* na NA, NULL ili "", vrednost će biti računata kao $10^{-digits}$, gde se digits odnosi na svojstvo *digits*.
podrazumevana vrednost: `""`
sintaksa: `softRestartThreshold = ""`

Opcije vezane za parametre:

opcija: **parameterFile**
opis: Fajl u kome se navode parametri koji se nameštaju, kao i njihov opis. Videti sekciju "parameters.txt" za više informacija.
podrazumevana vrednost: "./parameters.txt"
sintaksa: `parameterFile = "./parameters.txt"`

opcija: **forbiddenFile**
opis: Fajl pomoću koga se mogu zabraniti određene konfiguracije. Videti sekciju "forbidden.txt" za više informacija.
podrazumevana vrednost: `""`
sintaksa: `forbiddenFile = "./forbidden.txt"`

opcija: **digits**
opis: Maksimalan broj decimala koje su od značaja kod realnih parametara.
podrazumevana vrednost: 4
sintaksa: `digits = 4`

Opcije vazane za targetRunner:

opcija: **targetRunner**
opis: Implementacija algoritma čije parametre nameštamo. Za više informacija pogledati sekciju "target-runner".

[6] Hemingovo rastojanje između dve niske jednakih dužina jednako je broju mesta na kojima se odgovarajući karakteri ne poklapaju. Primer: Hemingovo rastojanje između "bliska" i "spiska" je 2.

podrazumevana vrednost: "./target-runner"
sintaksa: `targetRunner = "./target-runner"`

opcija: **`targetRunnerRetries`**
opis: Broj ponovnih pokušaja poziva targetRunner-a ukoliko prvobitni poziv nije uspeo.
podrazumevana vrednost: 0
sintaksa: `targetRunnerRetries = 0`

opcija: **`targetRunnerData`**
opis: Opcioni podaci koji se mogu proslediti targetRunner-u. Može se koristiti za prosleđivanje trajnih podataka (eng. persistent data).
podrazumevana vrednost: ""
sintaksa: `targetRunnerData = ""`

opcija: **`targetRunnerParallel`**
opis: Možemo napisati funkciju u R-u radi potpune kontrole paralelizacije targetRunner-a. Irace poziva funkciju u obliku: `targetRunnerParallel(experiments, exec.target.runner, scenario, target.runner)`, gde je `experiments` lista koja opisuje konfiguracije i instance, `exec.target.runner` je interna funkcija unutar irace-a koja se brine o pozivu `target.runner` (što je četvrti argument funkcije `targetRunnerParallel`), proverava izlaz i pokreće ponovo ukoliko je došlo do greške (videti *targetRunnerRetries*), `scenario` je lista koja opisuje konfiguracioni scenario, a `target.runner` je funkcija koja poziva targetRunner. Ukoliko je targetRunner funkcija u R-u, onda je `target.runner` isto što i targetRunner.
Primer:

```
targetRunnerParallel <- function(experiments, exec.target.runner,
scenario)
{
      return (lapply(experiments, exec.target.runner, scenario = scenario,
      target.runner = target.runner))
}
```

Možemo namestiti pozive funkcije kako god želimo, jedino što se zahteva jeste da funkcija *targetRunnerPatallel* vraća listu iste dužine kao `experiments`, i da je svaki element te liste odgovarajuća povratna vrednost targetRunner-a za jedan eksperiment. Za više detalja pogledati glavu 6 u [1], stavka 4.
podrazumevana vrednost: ""
sintaksa: `targetRunnerParallel = ""`

opcija: **`targetEvaluator`**
opis: Opcioni program ili funkcija u R-u. Vraća numeričku vrednost nakon što se sve konfiguracije u jednoj trci ispitaju nad jednim parom (*instanca*, *seme*). Za više detalja pogledati sekciju "target-evaluator".
podrazumevana vrednost: ""
sintaksa: `targetEvaluator = ""`

opcija: **`deterministic`**
opis: Ako je algoritam za koji želimo da podesimo parametre deterministički, konfiguracije će biti ispitivane samo jednom po instanci. Imati u vidu da, ukoliko je broj instanci koje se koriste za fazu treniranja manji od vrednosti za *firstTest*, neće biti sproveden nijedan statistički test. Kod

stohastičkih algoritama, konfiguracija će biti testirana nad istom instancom više puta, sa različitim semenima. Moguće vrednosti za *deterministic* su 0 i 1, gde 1 označava da je u pitanju deterministički algoritam.

podrazumevana vrednost: 0
sintaksa: `deterministic = 0`

opcija: **`parallel`**
opis: Broj poziva targetRunner-a koje treba izvršiti paralelno. Vrednosti 0 i 1 označavaju da nema paralelizacije.
podrazumevana vrednost: 0
sintaksa: `parallel = 0`

opcija: **`loadBalancing`**
opis: Omogući/onemogući balansiranje opterećenja (eng. load balancing) pri izvođenju paralelnih poziva targetRunner-a. Iako korisno, ovo svojstvo je nekada bolje onemogućiti, radi dobijanja na brzini.
podrazumevana vrednost: 1
sintaksa: `loadBalancing = 1`

opcija: **`mpi`**
opis: Omogući/onemogući upotrebu MPI (Message Passing Interface) protokola. Zahteva Rmpi paket (implementacija MPI u R-u) za paralelna izvršavanja targetRunner-a. Ukoliko je *mpi*=1 svojstvo *parallel* kontroliše broj podređenih čvorova. Npr. ukoliko je *mpi*=1 i *parallel*=N (gde je N neki broj) koristiće se N podređenih i jedan master čvor, i biće izvršeno do N poziva targetRunner-a paralelno. Podsećamo da iako korisno, *loadBalancing* svojstvo je nekada bolje onemogućiti, radi dobijanja na brzini. Korisnik je dužan da obezbedi odgovarajuće MPI okruženje. Primer upotrebe MPI na SGE klasteru se može naći na putanji `$IRACE_HOME/bin/parallelirace-mpi`, gde je `$IRACE_HOME` putanja dobijena u sekciji "Instalacija".
podrazumevana vrednost: 0
sintaksa: `mpi = 0`

opcija: **`batchmode`**
opis: Ova opcija se koristi u radu sa klasterima koji primaju paketne poslove (eng. batch jobs). Specifikuje kako irace čeka poslove da se završe kada targetRunner šalje poslove klasterima. Moguće opcije su: **sge**, **pbs**, **torque** i **slurm**. Sam targetRunner mora slati poslove klasteru koristeći, npr. *qsub* ili *squeue*, i vraćati samo ID posla. Takođe, potreban je *targetEvaluator* kako bi se ocenili, a zatim prsosledili irace-u, rezultati koje vraća targetRunner. U kombinaciji sa *batchmode* može se koristiti opcija *parallel*, kako bi se ograničio broj poslova koji se šalju istovremeno. Primeri se mogu naći u folderu koji je preuzet prilikom instalacije irace-a, odnosno na putanji `$IRACE_HOME/examples/batchmode-cluster/`, gde je `$IRACE_HOME` putanja dobijena u sekciji "Instalacija".
podrazumevana vrednost: 0
sintaksa: `batchmode = 0`

Opcije vazane za inicijalne konfiguracije:

opcija: **`configurationsFile`**
opis: Fajl koji sadrži inicijalne konfiguracije. Za više informacija videti sekciju "configurations.txt".
podrazumevana vrednost: `""`
sintaksa: `configurationsFile = "./configurations.txt"`

Opcije vazane za trening instance:

opcija: **`trainInstancesDir`**
opis: Direktorijum u kome se nalaze instance za trening. Za više informacija videti sekciju "Instances, instances.txt i testing.txt".
podrazumevana vrednost: "./Instances"
sintaksa: `trainInstancesDir = "./Instances"`

opcija: **`trainInstancesFile`**
opis: Fajl u kome su navedene instance koje treba uzeti za trening. Za više informacija videti sekciju "Instances, instances.txt i testing.txt".
podrazumevana vrednost: `""`
sintaksa: `trainInstancesFile = "./instances.txt"`

Ukoliko je navedeno svojstvo *trainInstancesDir* , instance iz *trainInstancesFile* će se tražiti u tom direktorijumu, a ukoliko *trainInstancesFile* opcija nije navedena, a *trainInstancesDir* jeste, tada će se za trening uzeti sve instance iz *trainInstancesDir* direktorijuma.

Opcije vazane za budžet:

opcija: **`maxExperiments`**
opis: Maksimalni broj poziva targetRunner programa. Određuje budžet za trening. Najčešće se uzima broj između 1 000 i 100 000, a koji broj je najbolje uzeti zavisi od složenosti našeg programa, veličine domena parametara i sličnosti instanci.
podrazumevana vrednost: 0
sintaksa: `maxExperiments = 0`

opcija: **`maxTime`**
opis: Maksimalno vreme ukupnog izvršavanja svih poziva targetRunner programa u sekundama. Irace proračunava vreme izvršavanja pojedinačnog poziva targetRunner-a kako bi se na osnovu *maxTime* utvrdilo koliko ekserimenata (tj. koliko poziva targetRunner-a) može da se izvrši tokom faze treniranja. Budžet za ovu procenu, zadaje se preko *budgetEstimation* opcije. Dobijena procena se ažurira nakon svake iteracije, na osnovu dobijenih rezultata. Ukoliko se koristi opcija *maxTime*, sa vrednošću različitom od 0, targetRunner mora da se napiše tako da vraća i vreme izvršavanja (dakle vraća dve vrednosti umesto jedne).
podrazumevana vrednost: 0
sintaksa: `maxTime = 0`

Mora se zadati ili *maxExperiments* ili *maxTime*. Ovim opcijama zadaje se budžet za rad irace-a.

opcija: **`budgetEstimation`**
opis: Deo budžeta (broj manji od 1) koji se koristi za procenu vremena izvršavanja targetRunner-a, na osnovu čega se predviđa broj mogućih poziva targetRunner-a. Koristi se samo kada je *maxTime* > 0.
podrazumevana vrednost: 0.02
sintaksa: `budgetEstimation = 0.02`

Opcije vazane za statistički test:

opcija: **`testType`**
opis: Statistički test koji će se koritsiti za eliminaciju konfiguracija. Mogući testovi su: **F-test** (Fridmanov test), **t-test** (parni t-test bez korekcija), **t-test-bonferroni** (t-test sa Bonferonijevom korekcijom za višestruka poređenja), **t-test-holm** (t-test sa Holmovom korekcijom za višestruka porđenja). Korišćenje testova sa višestrukim poređenjima se ne preporučuje, jer su pretežno "previše strogi" i teško se dolazi do željenih rezultata. Preporučuje se korišćenje podrazumevanih vrednosti. Više o f-testu i t-testu, kao i o odabiru odgovarajućeg testa, može se naći u sekciji 10.6 u [1].
podrazumevana vrednost: **t-test**, ako je opcija *capping* omogućena
F-test, ineče
sintaksa: `testType = "F-test"`

U dosadašnjim verzijama irace-a, podrazumevana vrednost za *testType* je F-test, čak i kada je opcija *capping* aktivirana. U verziji 3.5 ispravljen je ovaj propust.

opcija: **`firstTest`**
opis: Broj parova (*instanca*, *seme*) koji će se razmatrati pre prvog eliminacionog testa. Ako je npr. *firtstTest*=5, targetRunner će biti pozvan *nbConfigurations* puta za svaki od prvih pet razmatranih (*instanca*, *seme*) parova, a zatim će se izvršiti statistički test. Vrednost za *firstTest* mora biti deljiva sa vrednošću od *eachTest*.
podrazumevana vrednost: 5
sintaksa: `firstTest = 5`

opcija: **`eachTest`**
opis: Broj (*instanca*, *seme*) parova razmatranih između dva eliminaciona testa.
podrazumevana vrednost: 1
sintaksa: `eachTest = 1`

opcija: **`confidence`**
opis: Nivo pouzdanosti statističkog testa.
podrazumevana vrednost: 0.95
sintaksa: `confidence = 0.95`

Opcije vazane za kontrolu i minimizaciju vremena izvršavanja:

Opcije iz ove sekcije koriste se kada je cilj minimizacija vremena izvršavanja. U sekciji "target-runner" napomenuli smo da targetRunner vraća ili jednu vrednost oblika "kvalitetKonfiguracije", ili dve vrednosti u obliku "kvalitetKonfiguracije vreme". Ako je cilj minimizacija vremena izvršavanja, targetRunner kao kvalitet konfiguracije treba da vraća upravo vreme izvršavanja, odnosno vreme koje je potrebno da se dostigne neka ciljna vrednost. U slučaju da se budžet zadaje preko *maxTime,* a cilj je minimizacija vremena izvršavanja, targetRunner mora vraćati dva puta vreme izvršavanja (treba da vraća dve vrednosti u obliku: "vreme vreme"). Drugi argument *vreme* se koristi samo kako bi se pratio preostali budžet.

opcija: **`capping`**
opis: Omogući/onemogući "adaptive capping" tehniku. Naredni primer daje motivaciju za uvođenje napomenute tehnike. Neka se vrši posmatranje dve konfiguracije, θ_1 i θ_2. Neka vreme koje je potrebno da se ispita konfiguracija θ_1 nad N=100, (*instanca*, *seme*) parova, iznosi 10 sekundi (srednja vrednost vremena izvršavanja je 0.1 sekunda po paru (*instanca*, *seme*)). Neka je za konfiguraciju θ_2 potrebno 100 sekundi da se izvrši targetRunner samo nad prvim (*instanca*, *seme*) parom. Označimo srednju vrednost vremena izvršavanja targetRunner-a za θ_1 sa ms_{θ_1}, a za θ_2 sa ms_{θ_2}. Želimo da uporedimo te dve vrednosti. Kako su nam poznata vremana izvršavanja za θ_1 nad svakim (*instanca*, *seme*) parom, za θ_2 nije potrebno da posmatramo svih 100 parova. Izvršavanje targetRunner-a za θ_2 nad prvim parom možemo zaustavti već nakon $10+\varepsilon$ sekundi, jer već tada možemo napraviti poređenje koje nas zanima, dakle ne moramo čekati svih 100 sekundi koliko smo rekli da bi trebalo targetRunner-u da ispita θ_2 nad prvim (*instanca*, *seme*) parom. Za preostalih 99 parova vreme izvršavanja mora biti ne manje od 0 sekundi, te dobijamo $\frac{(10+\varepsilon)+99\cdot 0}{100} = 0.1+\frac{\varepsilon}{100} \leq ms_{\theta_2}$, odnosno dobijamo donje ograničenje za srednje vreme izvršavanja za θ_2. Kako je $ms_{\theta_1} \leq 0.1+\frac{\varepsilon}{100} \leq ms_{\theta_2}$, sigurni smo da θ_1 ima prednost u odnosu na θ_2. Dati primer iskazan na nešto drugačiji način dostupan je u [7]. Kada se koristi *capping* opcija irace zadaje vremensko ograničenje za svako izvršavanje targetRunner-a, uzimajući kao uzor vremena izvršavanja targetRunner-a **nad elitnim konfiguracijama**. Ukoliko dolazi do prekoračenja vremena za neku konfiguraciju, izvršavanje targetRunner-a za datu konfiguraciju se prekida i konfiguracija se odbacuje. Takođe, kada je aktivirana opcija *capping,* podrazumevana vrednost za statistički test, odnosno vrednost opcije *testType* je t-test (a inače je F-test). Za korišcenje ove opcije mora biti `elitist`=1. Više detalja o upotrebi "adaptive capping" tehnike u irace-u može se naći u [8].
podrazumevana vrednost: 0
sintaksa: `capping = 0`

U dosadašnjim verzijama irace-a, podrazumevana vrednost za *testType* je F-test, čak i kada je opcija *capping* aktivirana. U verziji 3.5 ispravljen je ovaj propust.

opcija: **`boundType`**
opis: Metod za računanje performansi elitnih konfiguracija, u cilju postavljanja ograničenja korišćenih za "adaptive capping" tehniku. Moguće vrednosti su **"candidate"** i **"instance"**. Ukoliko se koristi vrednost "candidate", performanse elitnih konfiguracija računaju se na osnovu rezultata nad svim do tada viđenim (*instanca*, *seme*) parovima. Konkretno,

performansa p_i^s , elitne konfiguracije *s* u trenutku kada se posmatra par (*instanca*, *seme*) sa oznakom (ID-jem) *i*, proračunava se kao srednja vrednost svih vremena izvršavanja targetRunner-a za elitnu konfiguraciju *s,* dobijenih na prethodnim (*instanca*, *seme*) parovima, i na trenutnom paru *i*. Ukoliko se koristi vrednost "instance", performansa se posmatra na svakom (*instanca*, *seme*) paru posebno.

podrazumevana vrednost: "candidate"
sintaksa: `boundType = "candidate"`

opcija: **`cappingType`**
opis: Koja će mera biti korišćena pri obradi performansi elitnih konfiguracija u cilju određivanja ograničenja spomenutih kod *capping* opcije. Vrednost ove opcije utiče na računanje vrednosti b_i. Za značaj vrednosti b_i videti tekst nakon ove opcije. Ukoliko se koristi podrazumevana vrednost ove opcije, b_i će se računati kao $b_i = \underset{\theta_s \in \Theta^{elite}}{median}\{p_i^s\}$, gde je Θ^{elite}

skup svih elitnih konfiguracija, θ_s je elitna konfiguracija *s,* a p_i^s je objašnjena u okviru *boundType* opcije. Moguće vrednosti za *cappingType* su **median**, **mean**, **worst** i **best**.

podrazumevana vrednost: "median"
sintaksa: `cappingType = "median"`

Ograničenje za novu konfiguraciju *j* se računa kao:

$$k_i^j = \begin{cases} b^{\max}, k'^j_i > b^{\max} \\ \min\{b_i, b^{\max}\}, k'^j_i \le 0 \\ k'^j_i, inače \end{cases}$$

gde $k'^j_i = b_i \cdot i + b_{\min} - p^j_{i-1} \cdot (i-1)$
p_{i-1}^j je performansa konfiguracije *j* pre posmatranja para (*instanca*, *seme*) sa oznakom (ID-jem) *i*. Mala konstanta b_{min} se dodaje zbog grešaka u računanju vremena. Uz statistički test koristi se i elimiacioni test u vidu $b_i + b_{\min} < p_i^j$. Ograničenje za izvršavanje se stalno menja u toku rada irace-a, ali maksimalno vreme (b^{max}) zadato sa opcijom *boundMax* se nikada ne prekoračuje. b_i se računa na način opisan kod opcije *cappingType*.

opcija: **`boundMax`**
opis: Maksimalno ograničenje (b^{max}) za vreme izvršavanja targetRunner-a. Mora biti navedeno ukoliko je opcija *capping* aktivirana.
podrazumevana vrednost: 0
sintaksa: `boundMax = 0`

opcija: **`boundDigits`**
opis: Preciznost koja se koristi za računanje vremena izvršavanja. Mora biti navedeno ukoliko je opcija *capping* aktivirana.
podrazumevana vrednost: 0
sintaksa: `boundDigits = 0`

opcija: **boundPar**

opis: Kada se dostigne *boundMax* vreme izvršavanja, a targetRunner nije uspešno završio svoj rad (eng. timed out executions), koristi se kazna (eng. penalty) poznata kao PARX i *boundMax* se množi konstantom X koja se zadaje opcijom *boundPar*. Izvršenja targetRunner-a na kojima je primenjena kazna uzimaju se u obzir pri eliminacionim testovima i upoređivanju kvaliteta konfiguracija (ne uzimaju se u obzir prilikom određivanja ograničenja).

podrazumevana vrednost: 1

sintaksa: `boundPar = 1`

opcija: **boundAsTimeout**

opis: Zamenjuje ocenu kvaliteta konfiguracije sa *boundMax* ukoliko dolazi do prekoračenja ograničenja.

podrazumevana vrednost: 1

sintaksa: `boundAsTimeout = 1`

Opcije vazane za oporavak:

opcija: **recoveryFile**

opis: Na kraju svake iteracije irace čuva rezultate u log fajlu zadatom opcijom *logFile*. U slučaju da dođe do prekida rada irace-a, moguće je upotrebiti prethodno generisan log fajl za oporavak (eng. recovery) izvršenja irace-a. Preimenovati log fajl koji će se koristiti za oporavak u npr. "irace-backup.Rdata", kako bi se razlikovao od novog logFile-a (koji će se generisati prilikom oporavka), i postaviti ga kao fajl za oporavak. Navesti apsolutnu ili relativnu putanju u odnosu na trenutni direktorijum. Podaci će biti iskorišćeni kako bi irace mogao da nastavi rad tamo gde je stao. Ne menjati ostale opcije u scenario fajlu, niti fajl koji se koristi za oporavak, jer to može prouzrokovati promene u rezultatima. Oporavak raditi na istoj mašini. Svi fajlovi koji su bili korišćeni u pozivu irace-a za koji se radi oporavak, moraju biti dostupni kada se vrši oporavak. Primer: Pokrenuli smo irace bez zadavanja opcija *logFile* i *recoveryFile*. Rad programa se prekinuo ili smo ga mi prekinuli nakon neke iteracije (mora da prođe barem jedna iteracija kao bi oporavak bio moguć). Želimo da irace nastavi rad tamo gde je stao.

1. Ulazimo u folder u kome nam se sačuvao logFile, kako u primeru koristimo podrazumevane vrednosti (jer kako nismo naveli opciju *logFile*, koristi se njena podrazumevana vrednost što je "./irace.Rdata"), taj fajl se nalazi u *execDir* pod nazivom "irace.Rdata". Fajl "irace.Rdata" preimenujemo u "irace-backup.Rdata".
2. U "scenario.txt" dodamo *recoveryFIle*="./irace-backup.Rdata" i sačuvamo.
3. Pokrenemo opet irace. On će uzeti u obzir dosadašnje rezultate i nastaviti rad. Kreiraće novi log fajl pod nazivom "irace.Rdata" (zbog toga smo prvi takav fajl preimenovali).

podrazumevana vrednost: `""`

sintaksa: `recoveryFile = "./irace-backup.Rdata"`

Opcije vazane za fazu testiranja:

opcija: **testInstancesDir**

opis: Direktorijum u kome se nalaze test instance. Za više informacija videti sekciju "Instances, instances.txt i testing.txt".

podrazumevana vrednost: `""`
sintaksa: `testInstancesDir = "./Instances"`

opcija: **`testInstancesFile`**
opis: Fajl u kome su navedene instance koje treba uzeti za testiranje. Za više informacija videti sekciju "Instances, instances.txt i testing.txt".
podrazumevana vrednost: `""`
sintaksa: `testInstancesFile = "./testing.txt"`

opcija: **`testNbElites`**
opis: Maksimalan broj elitnih konfiguracija vraćenih od strane irace-a prilikom treninga, koje će se testirati ukoliko su zadate instance za testiranje (opcijama *testInstancesDir* i/ili *testInstancesFile*). Videti sledeću opciju i primere nakon nje radi boljeg razumevanja.
podrazumevana vrednost: 1
sintaksa: `testNbElites = 1`

opcija: **`testIterationElites`**
opis: Omogući/onemogući testiranje elitnih konfiguracija pronađenih u svakoj iteraciji. Videti naredne primere radi pojašnjenja.
podrazumevana vrednost: 0
sintaksa: `testIterationElites = 0`

Primeri za *testNbElites* i *testIterationElites*:
primer1: Neka je *testIterationElites* = 0, a *testNbElites* = 1. Tada će se samo konfiguracija koja se najbolje pokazala u fazi treninga (finalna najbolja) koristiti u fazi testiranja.
primer2: Neka je *testIterationElites* = 1, a *testNbElites* = 1. Tada će se pored krajnje najbolje, testirati i najbolje konfiguracije iz svake iteracije (po jedna iz svake iteracije).
primer3: Neka je *testIterationElites* = 1, a *testNbElites* = 2. Tada će se testirati finalna najbolja i druga po redu najbolja konfiguracija, kao i po dve najbolje konfiguracije nađene u svakoj iteraciji.

Pokretanje i rad irace-a

Pokretanje i rad ćemo demonstrirati na primeru. Za targetRunner ćemo koristiti program koji implementira simulirano kaljenje za rešavanje problema p-hab centra neograničenih kapaciteta sa višestrukim alokacijama (UMApHCP; videti sekciju 2 u [10]). Simulirano kaljenje nije implementirano na standardni način sa temperaturom, već je korišćen nešto drugačiji pristup. Irace se koristi u ovom primeru da bi se odredio maksimalan broj iteracija koje će se koristiti kao kriterijum zaustavljanja. Primer je uprošćen radi demonstracije rada irace-a.

Da bismo pravilno pokrenuli irace potrebno je da pripremimo neophodne fajlove, koji su navedeni u sekciji "Neophodni fajlovi i direktorijumi za irace". Na Desktopu je napravljen direktorijum pod nazivom “iraceOpet”, a u njemu direktorijum "Instances" i fajlovi "instances.txt", "parameters.txt", "scenario.txt", "test.txt" i "simulirano_kaljenje.cpp". Potrebno je da se u terminalu pozicioniramo u direktorijum iraceOpet i prvo kompajliramo naš targetRunner program (ukoliko to nismo već uradili).

Izvršeno je `g++ simulirano_kaljenje.cpp -o simulirano_kaljenje.o`. U direktorijumu "Instances" nalaze se AP instance (koje se mogu naći na [11] (fajl `phab1.txt`)). Radi jednostavnosti za treniranje su uzete samo 2 instance, kao i za testiranje (u praksi bismo koristili više). Parametar koji želimo da podesimo je broj iteracija. Granice njegovog domena su zadate bez posebnog razmatranja njihovog kvaliteta.

Sadržaji fajlova:

-instances.txt:

```
AP403.tsp
AP502.tsp
```

-parameters.txt:

```
MAX_ITERACIJA    "--maxi "  i     (50, 3000)
```

-scenario.txt:

```
maxExperiments = 1000
parameterFile = "./parameters.txt"
targetRunner = "./simulirano_kaljenje.o"
trainInstancesDir = "./Instances"
trainInstancesFile = "./instances.txt"
testInstancesDir = "./Instances"
testInstancesFile = "./test.txt"
```

-test.txt:

```
AP102.tsp
AP405.tsp
```

Imajući u vidu instance sa kojima radimo, iz prakse nam je poznato da je potrebno koristiti znatno veće vrednosti za MAX_ITERACIJA, od onoga što nam dozvoljava domen koji smo zadali u "parameters.txt". Ipak, ovu informaciju ne koristimo za dati primer. U scenario fajlu nismo morali da navedemo opcije parameterFile i **train**InstancesDir (ali **test**InstancesDir moramo navesti) jer koristimo njihove podrazumevane vrednosti. Takođe smo mogli naš **izvršni** program da nazovemo "target-runner" umesto "simulirano_kaljenje.o". U tom slučaju ne bismo morali da navedemo ni opciju targetRunner jer znamo da je njena podrazumevana vrednost "./target-runner". Dakle dovoljno bi bilo navesti:

```
maxExperiments = 1000
trainInstancesFile = "./instances.txt"
testInstancesDir = "./Instances"
testInstancesFile = "./test.txt"
```

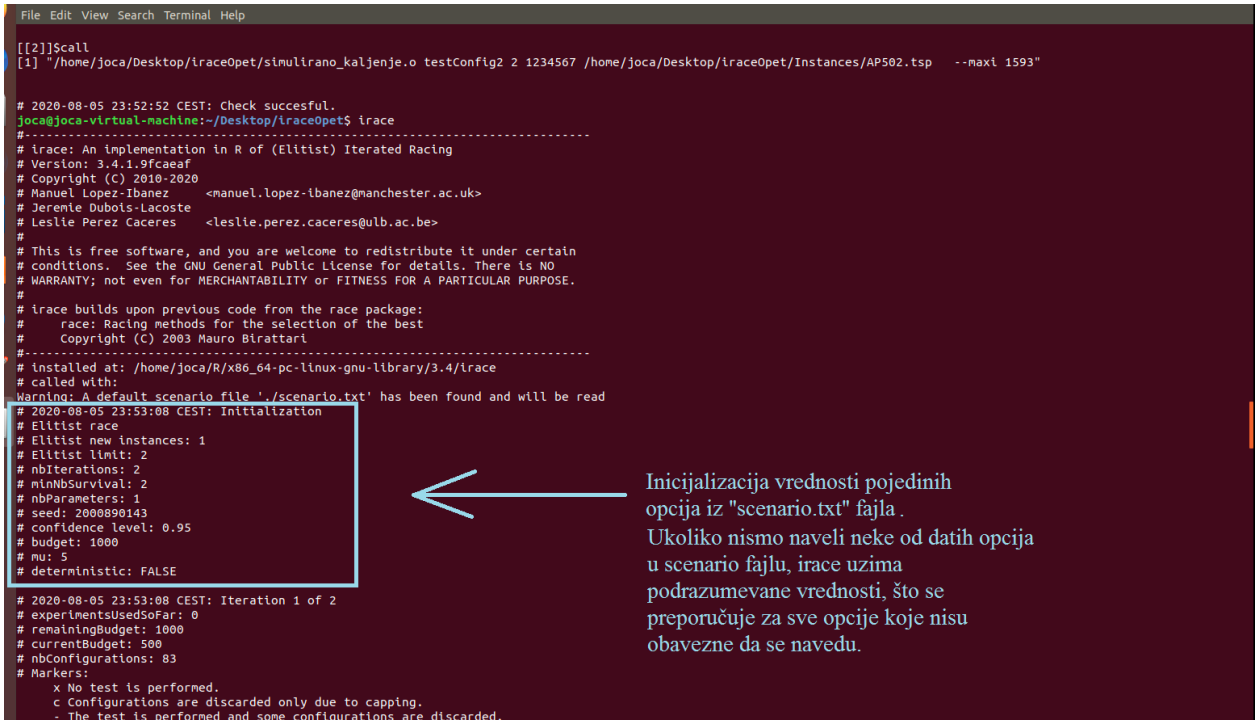
Ispravnost našeg scenarija možemo proveriti tako što ćemo u terminalu napisati putanju do mesta na kome je irace instaliran (možemo je dobiti kao na početku; videti `system.file` u sekciji "Instalacija") i opciju "**--check**".

Primer:

```
/home/joca/R/x86_64-pc-linux-gnu-library/3.4/irace/bin/irace --check
```

Irace će nam ispisati sve opcije iz scenario fajla zajedno sa njihovim vrednostima, ispisaće način na koji se targetRunner poziva iz komandne linije i testiraće targetRunner program. Ukoliko smo sve uradili kako treba provera bi trebalo da bude uspešna.

Kada smo pripremili sve što nam je potrebno i uspešno izvršili proveru, možemo pokrenuti irace. Pozicionirani u našem direktorijumu sa svim potrebnim fajlovima, pokrećemo irace komandom: `irace`. Najpre se ispisuju neke informacije o softveru, zatim vrednosti pojedinih opcija (videti sliku 3).



Slika 3: *Poziv irace-a.*

Svaki poziv irace-a mora imati unapred određen budžet za rad, koji mi zadajemo. Budžet se mora zadati ili kao maksimalni broj poziva targerRunner-a, što se zadaje preko *maxExperiments* u "scenario.txt" fajlu, ili kao maksimalno dozvoljeno vreme za rad, što se zadaje preko *maxTime* u "scenario.txt" fajlu. Sa slike 3 vidimo da je *budget:* 1000, a kako smo zadali *maxExperiments* u scenario fajlu, ovo znači da je irace-u dozvoljeno da 1000 puta pozove targetRunner program. Sa slike 3 vidimo da je aktiviran elitizam što podrazumeva da će se u svakoj iteraciji određivati elitne konfiguracije, odosno one koje su se najbolje pokazale tokom iteracije. Elitne konfiguracije se prenose u narednu iteraciju i na osnovu njihovih vrednosti, biraju se vrednosti za nove konfiguracije. Da smo u "scenario.txt" fajlu naveli vrednost za *nbIterations*, bilo bi izvršeno **maksimalno** *nbIterations* iteracija. Moguće je da se maksimalni broj iteracija neće dostići usled nedostatka budžeta. Međutim, kako nismo naveli vrednost za *nbIterations* u scenario fajlu, koristi se podrazumevana vrednost. *nbIterations* u ovom slučaju predstavlja **minimalan** broj iteracija, odnosno, predstavlja procenu broja

iteracija koje će se izvršiti. Ukoliko ostane dovoljno neiskorišćenog budžeta biće izvršeno još iteracija. Sa slike 3 vidimo da će u našem slučaju biti odrađene najmanje 2 iteracije. U svakoj iteraciji ispituje se, odnosno trka se, *nbConfigurations* konfiguracija. Da smo zadali vrednost za *nbConfigurations* u scenario fajlu, u svakoj iteraciji bi se ispitivalo tačno toliko konfiguracija. Kako nismo naveli napomenutu opciju, irace za svaku iteraciju sam bira vrednost za *nbConfigurations*, pri čemu vrednost ne mora biti ista za svaku iteraciju. U prvoj iteraciji se na osnovu domena parametara generiše *nbConfigurations* konfiguracija. Da smo naveli inicijalne konfiguracije u "configurations.txt", one bi se usvojile i generisalo bi se još onoliko konfiguracija koliko treba da bi ukupan broj ispitivanih konfiguracija bio *nbConfigurations*. U kasnijim iteracijama prenose se elitne konfiguracije iz prethodne iteracije (ukoliko se koristi elitizam, što jeste slučaj u našem primeru) i generiše se onoliko novih, koliko treba, tako da ih ukupno bude *nbConfigurations*.

Počevši od prve iteracije informacije o budžetu se iskazuju kroz *remainingBudget*, što predstavlja količinu neiskorišćenog budžeta u datom trenutku, i *currentBudget*, što predstavlja budžet za trenutnu iteraciju. Na početku svake iteracije, a pre početka poređenja konfiguracija, odnosno trke, *remainingBudget* se deli na broj iteracija koje još nisu odrađene (što uključuje i trenutno posmatranu iteraciju) i time se dobija *currentBudget* za trenutnu iteraciju. Moguće je da se neće ceo *currentBudget* iskoristiti u tekućoj iteraciji, kao što ni irace možda neće iskoristiti ceo ukupan budžet koji mu je dat (jer budžet predstavlja **maksimalni** broj eksperimenata[7] koji se mogu izvršiti). Kao što smo rekli, u našem primeru planirano je da se izvrše 2 iteracije. Pogledajmo prvu iteraciju na slici 4. U prvoj iteraciji u našem primeru *remainingBudget*, čija je vrednost 1000, deli se na 2 iteracije i time se dobija *currentBudget* = 500 za prvu iteraciju. Razmotrimo slučaj u kome je planirano izvršavanje 7 iteracija a trenutno se započinje treća. Vrednost za *currentBudget*, za treću iteraciju, biće jednaka $\frac{remainingBudget}{5}$, ukoliko ovaj broj nije ceo, uzima se prvi manji ceo broj.

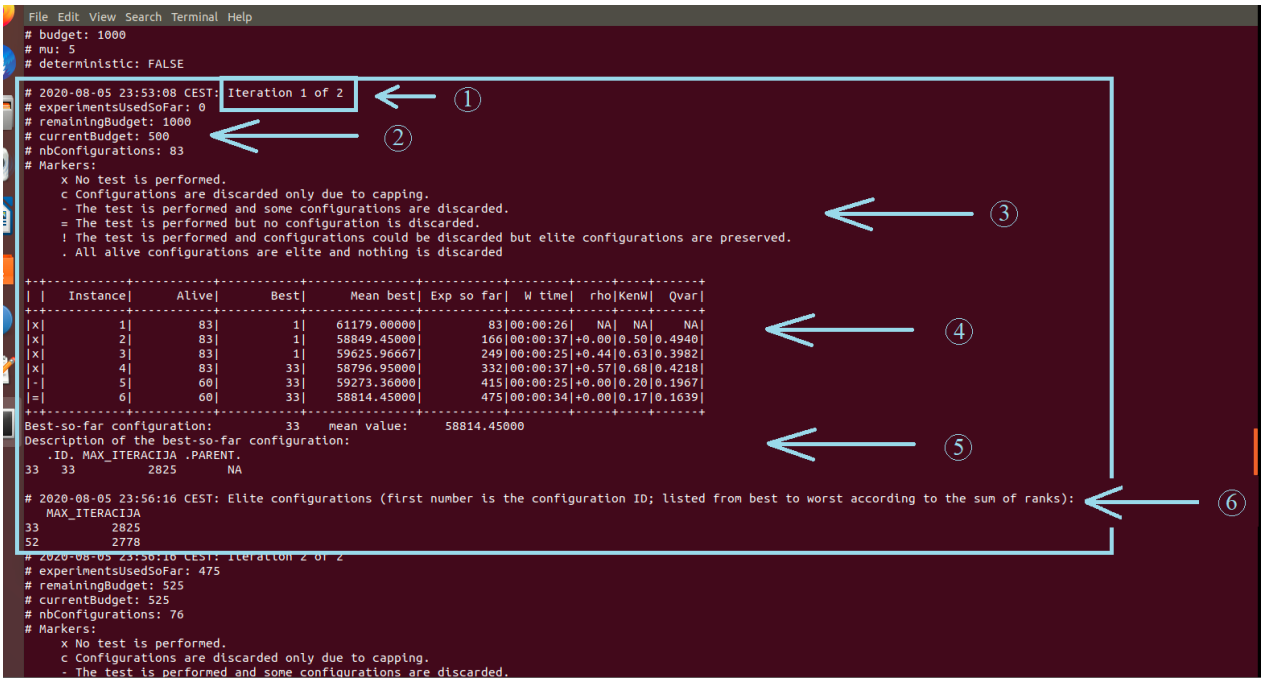
Iz dela 2 sa slike 4 vidi se da se za prvu iteraciju generisalo 83 kofiguracije (*nbConfigurations* = 83) koje su se ispitivale nad uređenim parovima (*instanca*, *seme*). Određeni broj parova (*instanca*, *seme*) generiše se pre početka ispitivanja i svakom paru se dodeljuje identifikacioni broj (ID). Svaka od instanci uparuje se sa većim brojem semena. Ispitivanje konfiguracija vrši se nad nekim od generisanih parova. Pod kolonom **Instance** u tabeli na slici 4 prikazani su napomenuti ID-jevi. Naglašavamo da iako je naziv napomenute kolone "Instance", identifikacioni brojevi dati u koloni odgovaraju parovima (*instanca*, *seme*). Skrećemo pažnju na to da se u [1] na pojedinim mestima pod instancom podrazumeva par (*instanca*, *seme*). U ovom uputstvu pravljena je jasna razlika između tih pojmova. U našem primeru zadali smo dve instance za fazu treniranja. Parova (*instanca*, *seme*) ima mnogo više, što se može videti u sekciji "Čitanje rezultata iz logFile-a". Dodavanje semena i posmatranje iste instance više puta u toku trke, ima smisla samo za algoritme stohastičke prirode (naš algoritam jeste stohastički). Kao što je napomenuto u opisu opcije *deterministic*, ukoliko je u pitanju deterministički algoritam jedna instanca se posmatra samo jednom u toku jedne trke. Vratimo se na tabelu na slici 4. Irace započinje ispitivanje u prvoj iteraciji tako što sve generisane konfiguracije ispituje nad prvim parom (*instanca*, *seme*) (prvi red u tabeli). U kolonu **Alive** upisuje se koliko je konfiguracija i dalje živo. Konfiguracije "umiru", odnosno odstranjuju se, ukoliko se nakon statističkog testa pokažu kao značajno lošije u poređenju sa konfiguracijom koja je u tom trenutku najkvalitetnija. U kolonu **Best** upisuje se ID konfiguracije koja je najbolja do sada u trenutnoj iteraciji (ukoliko ima više podjednako

[7] Jedan eksperiment podrazumeva jedno izvršavanje targetRunner-a.

dobrih upisuje se ona čiji je ID najmanji). U kolonu **Mean best** upisuje se srednja vrednost izlaznih vrednosti targetRunner-a za pozive trenutno najbolje konfiguracije. U našem primeru pri ispitivanju 83 konfiguracije nad prvim parom (instanca, *seme*) najbolja konfiguracija je bila ona za koju je ID = 1. To vidimo iz prvog reda tabele. Dalje se ispituju sve konfiguracije nad drugim parom (*instanca*, *seme*) i opet je najbolja konfiguracija ona za koju je ID = 1 (drugi red u tabeli). U prvom redu, u koloni **Mean best** stoji baš povratna vrednost targetRunner-a koji je pozvan za prvi par (*instanca*, *seme*) i konfiguraciju sa ID-jem 1. U drugom redu u koloni Mean best biće vrednost koja se dobila sabiranjem vrednosti koju targetRunner vrati kada se pozove za drugi par (*instanca*, *seme*) i konfiguraciju sa ID-jem 1 i prethodno dobijene vrednosti za istu konfiguraciju za prvi par (*instanca*, *seme*), podeljena sa 2. Statistički testovi se podrazumevano rade nakon što se konfiguracije ispitaju nad 5 parova (instanca, *seme*). Ukoliko želimo da se statistički test izvrši ranije, ili kasnije u odnosu na podrazumevanu vrednost, možemo postaviti vrednost za *firstTest* u `"scenario.txt"`. Zbog prethodno napomenutog, tek od petog reda tabele postoji mogućnost da je neka konfiguracija odstranjena. U petom redu prve kolone tabele date na slici 4, stoji znak `"|-|"`. Ako pogledamo deo označen sa 3 na slici 4 vidimo da to znači da je statistički test odrađen i da su neke konfiguracije odstranjene. U istom redu u koloni Alive stoji 60 što znači da su odstranjene 23 konfiguracije. Vrednost kolone **Exp so far** odgovara broju eksperimenata (izvršavanja targetRunner-a) izvršenih do trenutka posmatranja, u toku trenutne iteracije. Broj parova (*instanca*, *seme*) nad kojima ispitujemo konfiguracije zavisi od budžeta, broja preživelih konfiguracija i broja eksperimenata koji su do sada odrađeni. Ako posmatramo peti red tabele, vidimo da je broj do sada izvršenih eksperimenata 415. Naš budžet za trenutnu iteraciju je *currentBudget* = 500 (videti deo 2 na slici 4), što znači da nam je u trenutnoj iteraciji na raspolaganju još 500 - 415 = 85 eksperimenata. Broj preživelih konfiguracija je 60 što znači da ukoliko posmatramo još jedan par (*instanca*, *seme*), dakle šesti par, trebaće nam još 60 eksperimenata. To je u redu jer nam je na raspolaganju 85 eksperimenata. Dakle posmatraćemo i šesti par. Nakon toga nijedna konfiguracija nije odstranjena, i dalje ih je 60. U **Exp so far** je sada 475, dakle imamo pravo na još 25 eksperimenata a nama treba 60 da bismo mogli da vršimo ispitivanje nad još jednim parom (*instanca*, *seme*), stoga ovde stajemo a tih 25 eksperimenata koje nismo iskoristili prebacujemo u naredne iteracije. Svojstvo *experimentsUsedSoFar* se može dobiti sabiranjem poslednjih vrednosti kolone **Exp so far** u svim prethodnim iteracijama, dok se *reamainingBudget* dobija kad od početnog budžeta oduzmemo *experimentsUsedSoFar*. Ukoliko je aktivirana opcija *capping*, nakon kolone Instance pojaviće se i kolona **Bound** koja označava vremensko ograničenje za dati par (*instanca*, *seme*). Kolona **W-time** označava `"wall time"`, odnosno koliko je vremena irace potrošio nad datim parom (*instanca*, *seme*). Vreme odgovara onom na satu. **Rho** je Spirmanov koeficijent korelacije ranga, **KenW** je Kendalov W koeficijent slaganja. Definicija za **Qvar** se može naći u [12]. Vrednosti blizu 1 za **rho** i **KenW**, odnosno vrednosti blizu 0 za **Qvar**, označavaju homogen scenario. Homogen scenario podrazumeva konzistentno ponašanje targetRunner-a nad svim (*instanca*, *seme*) parovima. Dobre konfiguracije imaju dobre rezultate nad svim posmatranim parovima, i slično loše konfiguracije se nad svim posmatranim parovima pokazuju kao loše. Nasuprot tome, u heterogenim scenarijima naš algoritam se ponaša različito za različite (*instanca*, *seme*) parove, odnosno kvalitet konfiguracija nije konstantan. Nad nekim parovima se dobro pokaže neka konfiguracija, a nad nekim se ista konfiguracija pokaže kao loša. Zamislimo da radimo sa heterogenim scenarijom. Pitamo se: Da li želimo da nađemo konfiguracije koje se pokazuju dobro nad svim (*instanca*, *seme*) parovima ali nisu najbolje možda ni za jedan od tih parova?

- Da. U ovom slučaju preporučuje se da se vodi računa o dve stvari. Prvo, instance problema treba da se uzimaju proizvoljnim redosledom, odnosno da je vrednost opcije *sampleInstances* u scenario fajlu postavljena na 1, što je njena podrazumevana vrednost. Zašto je ovo dobro? Razmotrimo slučaj u kome imamo skup instanci, koji sadrži instance iz dve klase. Pod klasom podrazumevamo skup instanci istog tipa. U fajlu za trening instance, naveli smo prvo sve instance iz prve klase, pa zatim sve iz druge. Hoćemo da irace uzima instance redom (dakle postavili smo *sampleInstances*=0). Neka zbog ograničenog budžeta eksperimenti mogu da se izvrše nad ukupno 7 (*instanca*, *seme*) parova, a nama je prvih 7 instanci iz prve klase. Tada nećemo proći nijednu instancu druge klase. Samim tim, rezultat neće biti onakav kakav želimo.

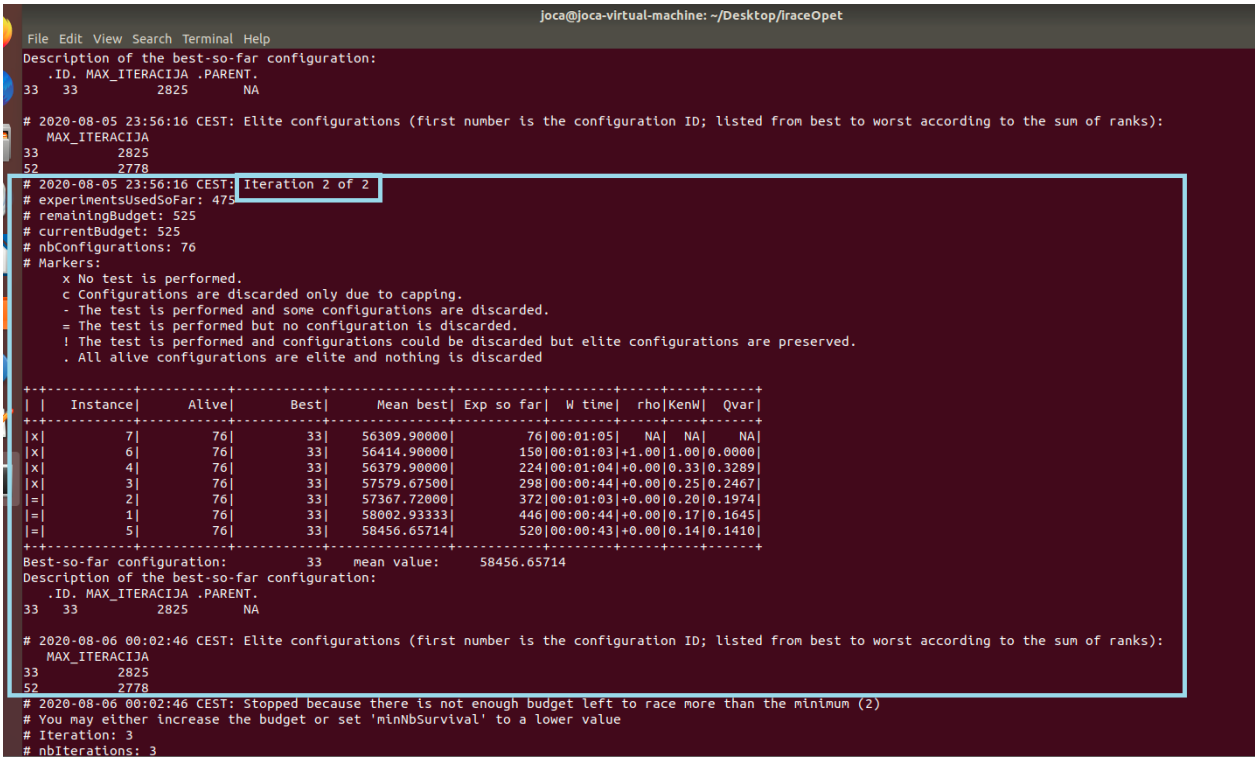
  Drugo, moglo bi biti korisno povećati broj (*instanca*, *seme*) parova nad kojima se vrše eksperimenti pre nego što se uradi statistički test, kako bi se prošlo kroz instance iz više klasa. Ukoliko se koristi elitizam ovo se postiže tako što se poveća vrednost za *elitistNewInstances*, a ukoliko se elitizam ne koristi može se povećati vrednost za *firstTest*, u istu svrhu.
- Ne, već želimo da nađemo konfiguracije parametara algoritma koje će biti dobre za svaku instancu ponaosob. Tada je najbolje podeliti instance prema njihovoj prirodi u manje skupove sličnih. U tom slučaju koristićemo svaki od tih podskupova kao zaseban skup trening instanci i tražiti od irace-a da nam predloži konfiguracije za svaki od tih skupova posebno. Dakle, ako podelimo naš skup instanci na N podskupova, napravićemo fajl za trening instance i fajl za test instance, koristeći instance iz prvog podskupa i pokrenuti irace. Kao rezultat ćemo dobiti konfiguracije koje su se najbolje pokazale nad prvim skupom instanci. Nakon toga napravićemo fajl za trening instance i fajl za test instance, koristeći instance iz drugog podskupa i pokrenuti irace. Kao rezultat ćemo dobiti konfiguracije koje su se najbolje pokazale nad drugim skupom instanci. Ovaj postupak ćemo ponoviti za svaki od N podskupova. Ukupno ćemo napraviti N fajlova za trening i N fajlova za test instance i pokrenuti irace N puta, svaki put menjajuci opcije *trainInstancesFile* i *testInstancesFile* u `"scenario.txt"`.



Slika 4:

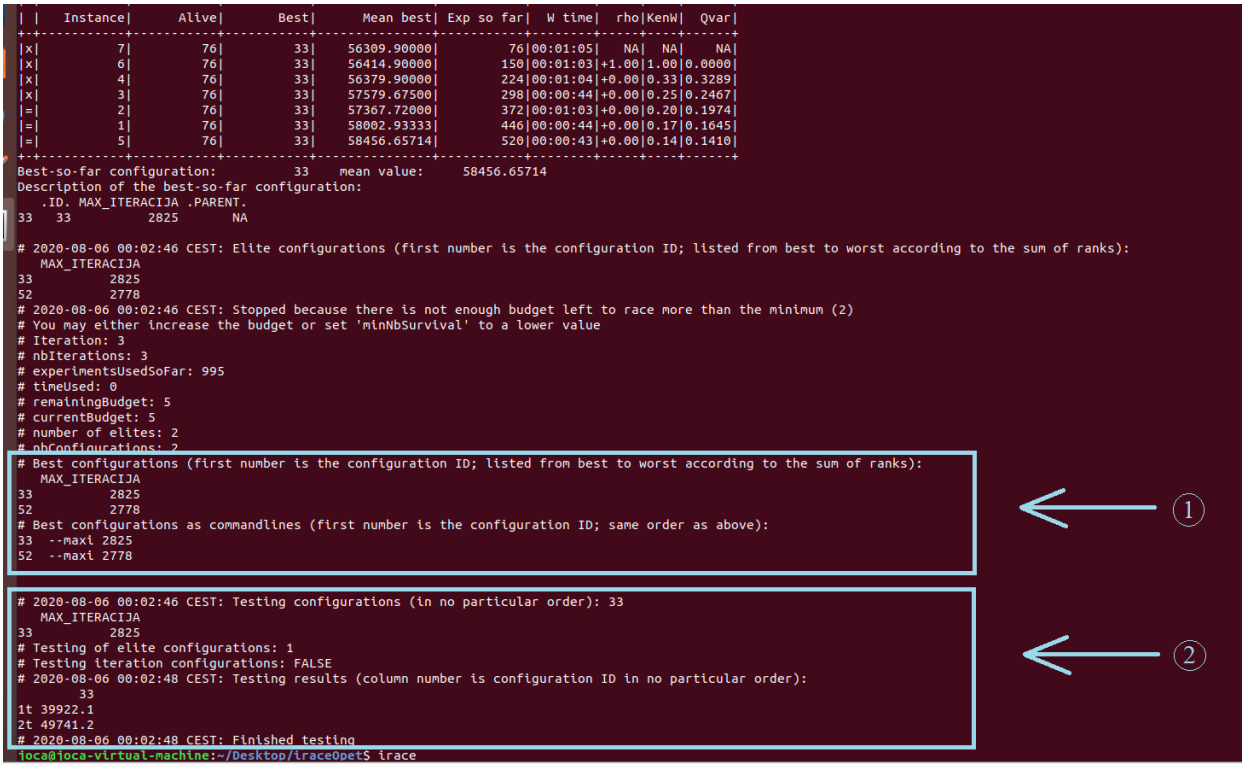
1. *Prva iteracija.*
2. *experimentsUsedSoFar – koliko je eksperimenata do sada odrađeno*
   remainingBudget – koliko eksperimenata može da se uradi do kraja
   currentBudget – koliko eksperimenata može da se uradi u trenutnoj iteraciji
   nbConfigurations – broj konfiguracija koje treba ispitati u trenutnoj iteraciji
3. *Oznake u tabeli koje označavaju da li je odrađen statistički test u tom koraku i ako jeste da li su odbačene neke konfiguracije.*
4. *Tabela rezultata.*
5. *Konfiguracija koja se pokazala kao najbolja u ovoj iteraciji (ukoliko ima više statistički podjednako dobrih konfiguracija uzima se prva od njih, odnosno ona sa najmanjim ID-jem).*
6. *Elitne konfiguracije iz trenutne iteracije, poređane od najbolje ka najlošijoj (ukoliko ima više statistički podjednako dobrih konfiguracija uzimaju se one sa najmanjim ID-jevima).*

Nakon prve iteracije elitne konfiguracije se prenose u drugu iteraciju i generiše se koliko treba novih konfiguracija. Sve se odigrava kao u prvoj itearciji. Videti sliku 5.



Slika 5: *Druga iteracija.*

Nakon dve iteracije irace proverava da li je ostalo dovoljno budžeta da se uradi dodatna iteracija (videti podnožje slike 5), kako to nije slučaj završava se faza treniranja i ispisuju se najbolje (viđene) konfiguracije (videti deo 1 na slici 6). Konfiguracije su poređane od najbolje ka najlošijoj, prema sumi rangova (videti sekciju 3.2 u [13] za više informacija o Fridmanovom statističkom testu). U fazi testiranja vrši se testiranje najbolje konfiguracije nad zadatim instancama i ispisuju se povratne vrednosti targetRunner-a (videti deo 2 na slici 6). U ovom slučaju testiranje je vršeno samo za najbolju konfiguraciju. Ukoliko želimo to da promenimo, to možemo uraditi pomoću opcija *testNbElites* i *testIterationElites* u "scenario.txt". Za razliku od faze treniranja, u fazi testiranja, jednoj test instanci se dodeljuje jedno seme. Pa se tartgetRunner pokreće po **jednom** za svaku test *instancu*, uz korišćenje njoj dodeljenog *semena*. Ukoliko želimo da se testiranje vrši više puta nad istom instancom a uz različito seme, to možemo postići tako što ćemo više puta navesti istu instancu u fajlu u kom navodimo test instance. Vrednosti dodeljenih semena se mogu izlistati, što ćemo videti u narednoj sekciji.

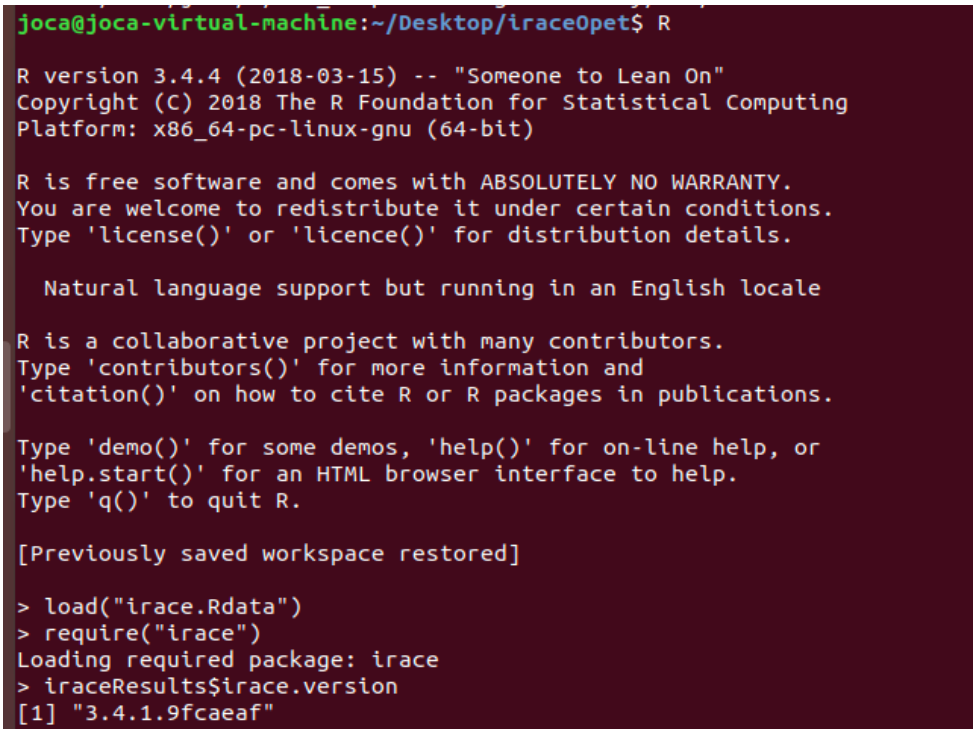


Slika 6: *1. Ispis najboljih (viđenih) konfiguracija.*

2. Testiranje najbolje (viđene) konfiguracije nad test instancama.

Čitanje rezultata iz logFile-a

Irace je završio sa radom i dao nam preporuku za vrednosti parametara razmatranog algoritma. Ukoliko želimo više informacija o rezultatima koji su se dobijali tokom rada irace-a na našem problemu, možemo otvoriti R okruženje u terminalu, komandom `R`. Najpre treba da učitamo fajl u kome su sačuvane vrednosti koje je irace dobio, naredbom `load`, i da učitamo paket irace, naredbom `require`. Ovo radimo svaki put kada otvorimo R okruženje. Trenutnu verziju irace-a možemo videti naredbom `iraceResults$irace.versoin`. Videti sliku 7.



Slika 7: *R okruženje.*

"irace.Rdata" fajl, koji je irace kreirao, sadrži informacije koje će pročitati R objekat **iraceResults**. Taj objekat predstavlja listu koja sadrži:

- objekat *scenario* (sadrži opcije iz scenario fajla)
- verziju za irace
- objekat *parameters*
- objekat *allConfigurations*
- listu *allElites*
- vektor *iterationElites* koji sadrži ID-jeve najboljih konfiguracija svake iteracije (za svaku iteraciju po 1)
- vektor *rejectedConfigurations* koji sadrži ID-jeve odbačenih konfiguracija
- matricu *experiments* koja sadrži sve izlazne vrednosti targetRunner-a nad svim parovima (*instanca*, *seme*) koji su korišćeni u iteracijama, i svim posmatranim konfiguracijama
- *experimentLog* matricu koja sadrži log svih eksperimenata koje je irace izvršio, a sastoji se od kolona *iteration*, *instance*, *configuration*, kao i kolone *time* ukoliko targetRunner ispisuje vremena izvršavanja i kolone *bound* koja se dodaje ukoliko je opcija *capping* omogućena
- *softRestart* vektor koji govori da li je *softRestart* izvršen za svaku od instanci ponaosob
- listu *state* na osnovu koje se radi oporavak
- *testing* listu koja sadrži matricu *experiments* (matrica eksperimenata pri testiranju) i listu *seeds* (lista *semena* korišćenih pri testiranju)

Elementi liste se mogu u R-u ispisati korišćenjem $ ili [[]] (kao npr. `iraceResults$irace.version` ili `iraceResults[["irace.version"]]`). Ukoliko želimo da se ispiše sve što se nalazi u R objektu, možemo upotrebiti naziv tog objekta. Dakle u ovom slučaju: `iraceResults`. Ako želimo da se ispiše samo određeni broj elemenata iz objekta, možemo iskoristiti funkciju *head*. Njen prvi argument je objekat čiji elementi nas zanimaju, a drugi opcioni argument je broj elemenata koje želimo da vidimo. Ukoliko se ne navede drugi argument biće prikazano 6 elemenata, a ako se navede kao drugi argument npr. 200 a elemenata ima 8, ispisaće se tih 8 postojećih elemenata. Da smo pozvali npr. `head(iraceResults, n = 1)`, ispisao bi nam se objekat *scenario*. Slično funkcija *tail* vraća elemente od kraja, dakle `tail(iraceResults, n = 1)` ispisuje samo poslednji element iz *iraceResults*.

Vratimo se sada na naš primer za koji smo koristili irace. Najpre ćemo izvršiti komande za učitavanje podataka i paketa (videti sliku 8).

```
> load("irace.Rdata")
> require("irace")
```

Slika 8: *Učitavanje paketa i podataka.*

Umesto da ispišemo sve podatke iz *iraceResults* posmatraćemo posebno njegove elemente. Nećemo se baviti ispisom elemenata kao npr. *scenario* i *parameters,* čiji nam je sadržaj već poznat. Ukoliko želimo da ispišemo ove podatke, možemo koristiti `iraceResults$scenario` i `iraceResults$parameters`, respektivno.

Lista svih generisanih parova (*instanca*, *seme*) se može izlistati pomoću `iraceResults$state$.irace$instancesList` (videti sliku 9). Tokom ispitivanja parametara algoritma, irace ne koristi sve raspoložive generisane parove. U našem primeru korišćeno je samo prvih 7 od generisanih parova.

```
> iraceResults$state$.irace$instancesList
    instance       seed
1          2  830967740
2          1  688720299
3          2 1410361231
4          1  380160547
5          2 2028762002
6          1 1142543995
7          1  467361645
8          2 2052604666
9          1 1112440519
10         2 1687849033
11         1    3049555
12         2 1332678343
13         1  164367094
14         2  233225823
15         1 1733938889
16         2 1951231942
17         1 1946668717
18         2  195087668
19         2  987705270
20         1  413708250
21         2 2049950914
22         1  560441247
23         1 1500000494
24         2 1405246073
25         1  881090654
26         2  297752675
27         1  697767694
28         2  898518418
29         1  528682555
30         2  564301418
31         2  632427046
32         1 1705178830
33         2 1216964837
34         1 1231588248
```

a) Početak liste.

```
478        2 1723159295
479        1 1032832710
480        2  792681777
481        2  336716520
482        1  536626382
483        2  634289560
484        1 1630041977
485        1 1079200243
486        2  933598248
487        1  781947220
488        2  452556366
489        1  925229893
490        2 1615895921
491        1  859740958
492        2  999386811
493        2 1202459840
494        1  964604764
495        2  118205016
496        1  484107615
497        2 1769956643
498        1  386494697
499        1  214215466
500        2 1369010843
```

b) Kraj liste.

Slika 9:. *Lista svih generisanih (instanca, seme) parova i njihovi ID-jevi.*

Posmatrajmo element ***allConfigurations*** (videti sliku 10). Ovaj objekat je *data frame* tip podataka čiji svaki red odgovara jednoj konfiguraciji.

```
> head(iraceResults$allConfigurations, n = 200)
    .ID. MAX_ITERACIJA .PARENT.
1      1          2297       NA
2      2           535       NA
3      3          1052       NA
4      4          1580       NA
5      5          2270       NA
6      6           464       NA
7      7           220       NA
8      8           318       NA
9      9           862       NA
10    10           925       NA
11    11           810       NA
12    12          2340       NA
```

a) Početak prikaza rezultata.

```
151  151          2818       33
152  152          2831       33
153  153          2868       33
154  154          2823       33
155  155          2819       33
156  156          2803       52
157  157          2771       52
```

b) Kraj prikaza rezultata.

Slika 10: *Element allConfigurations.*

Za ispis sadržaja datog objekta koristili smo funkciju *head*. Na slici 10 prikazan je samo deo rezultata. Vraćene su sve konfiguracije jer ih ima 157 a kao drugi argument funkcije *head* stavili smo n = 200. Ukoliko želimo sve konfiguracije, kao što smo već rekli, možemo samo pozvati `iraceResults$allConfigurations` (bez upotrebe funkcije *head*). Kolona .PARENT. pokazuje na osnovu koje konfiguracije je dobijena posmatrana konfiguracija. Ukoliko je posmatrana konfiguracija dobijena na osnovu neke druge prethodno ispitane konfiguracije, u toj koloni će stajati ID te „roditeljske“ konfiguracije, a u suprotnom će stajati NA. Nakon prve iteracije, ukoliko se koristi elitizam, nove konfiguracije se generišu na osnovu probabilističkog modela elitnih konfiguracija iz prethodnih iteracija.

Ako želimo da nam se izlista određena konfiguracija na osnovu ID-ja, možemo upotrebiti funkciju *getConfigurationById*, koja kao prvi argument prima logFile a kao drugi ID konfiguracije koju želimo (videti sliku 11).

```
> getConfigurationById(logFile = "irace.Rdata", ids = 33)
   .ID. MAX_ITERACIJA .PARENT.
33   33          2825       NA
> getConfigurationById(logFile = "irace.Rdata", ids = 30)
   .ID. MAX_ITERACIJA .PARENT.
30   30          1456       NA
> getConfigurationById(logFile = "irace.Rdata", ids = 52)
   .ID. MAX_ITERACIJA .PARENT.
52   52          2778       NA
```

Slika 11: *Dohvatanje konfiguracije na osnovu ID-ja.*

Ako želimo elitne konfiguracije po iteracijama izlistaćemo sadržaj liste ***allElites***. Možemo upotrebiti funkciju *print* kao na slici 12, ili samo otkucati `iraceResults$allElites`. . U listi ***allElites*** nalaze se ID-jevi konfiguracija koji odgovaraju onim u *iraceResults$allConfigurations*. Jedan element liste sadrži ID-jeve elitnih konfiguracija iz jedne iteracije. Elitne konfiguracije jedne iteracije poređane su od najbolje ka najgoroj, s' tim što ukoliko su dve konfiguracije podjednako dobre prvo se navodi ona sa manjim ID-jem. U našem primeru smo imali 2 iteracije sa po 2 elitne konfiguracije u svakoj od iteracija. Sa slike 12 vidimo da su u prvoj iteraciji elitne konfiguracije one sa ID-jevima 33 i 52, pri čemu 52 nije bolja od 33. U drugoj iteraciji je isti slučaj.

```
> print(iraceResults$allElites)
[[1]]
[1] 33 52

[[2]]
[1] 33 52
```

Slika 12: *Elitne konfiguracije po iteracijama.*

Ukoliko želimo samo najbolje konfiguracije iz svake iteracije (dakle po jednu iz svake), odnosno njihove ID-jeve, možemo uraditi sledeće: `print(iraceResults$iterationElites)` ili samo `iraceResults$iterationElites`. Prva konfiguracija rezultujućeg vektora odgovara najboljoj viđenoj konfiguraciji u prvoj iteraciji, dok poslednja odgovara finalnoj najboljoj konfiguraciji (onoj koja je bila najbolja nakon poslednje iteracije). Ako želimo da vidimo vrednosti parametara svih elitnih konfiguracija možemo pozvati: `getFinalElites(logFile = "irace.Rdata", n = 0)`. Ukoliko nas interesuju parametri za određen broj elitnih konfiguracija, možemo drugi argument, n, postaviti na taj broj. Tako na slici 13 izdvaja se samo prva elitna konfiguracija i smešta se u *best.config*.

Ukoliko želimo izlazne vrednosti za targetRunner, dobijene korišćenjem najbolje konfiguracije za parove (*instanca*, *seme*) koji su korišćeni u iteracijama, potrebno je izvršiti komande prikazane na slici 13. U promenljivu *best.config* izdvojena je najbolja konfiguracija. U promenljivu *id* izdvojen je identifikacioni broj najbolje konfiguracije. Pomoću treće linije komandi, u promenljivu *all.exp* izdvojene su izlazne vrednosti targetRunner-a za odgovarajuću konfiguraciju. Poslednjom linijom komandi, iz *all.exp* izbačeni su oni slučajevi kada je izlazna vrednost targetRunner-a NA. Vrednost

NA u ovom slučaju označava da targetRunner nije pozivan za datu konfiguraciju nad datim parom (*instanca*, *seme*), što se može desiti ukoliko: 1) data konfiguracija nije bila generisana kada se koristio taj par (*instanca*, *seme*), 2) konfiguracija je bila odbačena usled loših performansi i nije testirana nad datim parom ili 3) trka se prekinula pre nego što se razmatrao dati par.

```
> best.config <- getFinalElites(iraceResults = iraceResults, n = 1)
> id <- best.config$.ID.
> all.exp <- iraceResults$experiments[,as.character(id)]
> all.exp[!is.na(all.exp)]
      1       2       3       4       5       6       7
61179.0 56519.9 61179.0 56309.9 61179.0 56519.9 56309.9
```

Slika 13: *Izlazne vredsnosti targetRunner-a za najbolju konfiguraciju nad svim parovima (instanca, seme) koji su korišćeni u fazi treninga. To su u ovom slučaju parovi sa ID-jevima od 1 do 7 (lista ID-jeva se može videti na slici 7).*

Na slici 14 dat je primer za prve dve konfiguracije. Prvih 7 vrednosti odgovaraju testiranju sa prvom konfiguracijom, a drugih 7 sa drugom.

```
> best.config <- getFinalElites(iraceResults = iraceResults, n = 2)
> id <- best.config$.ID.
> all.exp <- iraceResults$experiments[,as.character(id)]
> all.exp[!is.na(all.exp)]
 [1] 61179.0 56519.9 61179.0 56309.9 61179.0 56519.9 56309.9 61179.0 56519.9
[10] 61179.0 56309.9 61179.0 56519.9 56309.9
```

Slika 14: *Izlazne vredsnosti targetRunner-a za prve dve najbolje konfiguracije, nad svim parovima (instanca, seme) koji su korišćeni u fazi treninga.*

Na slici 15 urađeno je slično, samo ovoga puta za sve konfiguracije. Na slici nije prikazan kompletan ispis.

```
> id <- iraceResults$allConfigurations$.ID.
> all.exp <- iraceResults$experiments[,as.character(id)]
> all.exp[!is.na(all.exp)]
  [1] 61179.0 56519.9 61179.0 56619.5 61179.0 56519.9 61179.0 56619.5 61179.0
 [10] 56915.6 61179.0 61179.0 56519.9 61179.0 56619.5 61179.0 56519.9 61179.0
 [19] 56519.9 61179.0 56619.5 61179.0 56519.9 61179.0 56519.9 61179.0 56619.5
 [28] 61179.0 56519.9 61179.0 56619.5 61676.3 56915.6 61179.0 61179.0 57368.3
 [37] 63533.7 56915.6 61179.0 61179.0 57368.3 61676.3 56915.6 61179.0 61179.0
 [46] 56519.9 61179.0 56619.5 61179.0 56519.9 61179.0 56519.9 61179.0 56619.5
 [55] 61179.0 56519.9 61179.0 56519.9 61179.0 56619.5 61179.0 56519.9 61179.0
 [64] 56519.9 61179.0 56619.5 61179.0 56519.9 61179.0 56519.9 61179.0 56619.5
 [73] 61179.0 56519.9 61179.0 56519.9 61179.0 56619.5 61179.0 56519.9 61179.0
 [82] 56519.9 61179.0 56619.5 61179.0 56519.9 61179.0 56519.9 61179.0 56619.5
 [91] 61179.0 56519.9 61179.0 56519.9 61179.0 56619.5 61179.0 56519.9 61179.0
[100] 56519.9 61179.0 56619.5 61179.0 56519.9 61179.0 56519.9 61179.0 56619.5
[109] 61179.0 56519.9 61179.0 56519.9 61179.0 56619.5 61179.0 56519.9 61179.0
[118] 56519.9 61179.0 56619.5 61179.0 56519.9 61179.0 56519.9 61179.0 56619.5
[127] 61179.0 56519.9 61179.0 57368.3 63036.1 56915.6 61179.0 61179.0 56519.9
[136] 61179.0 56619.5 61179.0 56519.9 61179.0 56519.9 61179.0 56619.5 61179.0
[145] 56519.9 61179.0 56519.9 61179.0 56619.5 61179.0 56519.9 61179.0 56519.9
[154] 61179.0 56619.5 61179.0 56519.9 61179.0 56519.9 61179.0 56619.5 61179.0
[163] 56519.9 61179.0 56519.9 61179.0 56619.5 61179.0 56519.9 61179.0 56519.9
[172] 61179.0 56619.5 61179.0 56519.9 61179.0 57368.3 63533.7 56915.6 66899.0
[181] 61179.0 60176.4 63533.7 56915.6 66899.0 61179.0 56519.9 61179.0 56309.9
[190] 61179.0 56519.9 56309.9 61179.0 58296.5 63533.7 56915.6 66899.0 61179.0
[199] 56619.5 61676.3 56915.6 61179.0 61179.0 56519.9 61179.0 56619.5 61179.0
[208] 56519.9 61179.0 56519.9 61179.0 56619.5 61179.0 56519.9 61179.0 56519.9
[217] 61179.0 56619.5 61179.0 56519.9 61179.0 56519.9 61179.0 56619.5 61179.0
[226] 56519.9 61179.0 56519.9 61179.0 56619.5 61179.0 56519.9 61179.0 56519.9
```

Slika 15:. *Povratne vredsnosti targetRunner-a za sve konfiguracije, nad svim parovima (instanca, seme) koji su korišćeni u fazi treninga*

Na slici 16 smo prikazali kako da izdvojimo instance i semena za, na primer, eksperimente izvršene za najbolju konfiguraciju. Prvih pet linija komandi objasnili smo na prethodnim primerima. Šestom linijom komandi štampamo ID-jeve parova (*instanca*, *seme*) koje smo dohvatili prethodnim komandama, i smeštamo podatke u *pair.id*. Sedmom linijom u index smeštamo vrednosti kolone "instance" iz liste (*instanca*, *seme*) parova, ali samo one vrednosti koje se nalaze u vrstama zadatim sa *pair.id*. Odnosno, izdvajamo indekse instanci koje se nalaze u parovima (*insatnca*, *seme*) čiji su ID-jevi obuhvaćeni vektorom *pair.id*. Dakle u ovom slučaju biće uzete vrednosti iz kolone "instance" koje se nalaze u prvih sedam vrsta. Osmom linijom komandi izlistavamo nazive instanci kojima odgovaraju prethodno izvučeni indeksi. Preskočimo za sada devetu liniju komandi. Preko *pair.id* i *index* vidimo šta je dohvaćeno (10. i 11. linija komandi). Dvanaestom linijom izlistavamo semena koja odgovaraju prethodnim instancama, i sa kojima čine parove (*instanca*, *seme*). Razlog preskakanja devete linije komandi se može videti u narednoj napomeni.

Napomena: u [1], u delu sekcije 9.2 u kome se opisuje *experiments* stoji `iraceResults$state.irace$instancesList[index, "seed"]`, a treba da piše `iraceResults$state.irace$instancesList[pair.id, "seed"]`, jer u *pair.id* smo sačuvali ID-jeve parova (*instanca*, *seme*) koje smo koristili i želimo da izlistamo semena. Ako stavimo `index` ponavljaće se 2 semena jer *index* čuva indekse instanci. Kako smo koristili samo dve to su onda 1 i 2. Videti sliku 9 za listu parova (*instanca*, *seme*). Ovo će biti ispravljeno u narednim verzijama oficijalnog uputstva (videti [1]).

```
> best.config <- getFinalElites(iraceResults = iraceResults, n = 1)
> id <- best.config$.ID.
> pair.id <- names(all.exp[!is.na(all.exp)])
> all.exp <- iraceResults$experiments[,as.character(id)]
> all.exp[!is.na(all.exp)]
      1       2       3       4       5       6       7
61179.0 56519.9 61179.0 56309.9 61179.0 56519.9 56309.9
> pair.id <- names(all.exp[!is.na(all.exp)])
> index <- iraceResults$state$.irace$instancesList[pair.id,"instance"]
> iraceResults$scenario$instances[index]
[1] "/home/joca/Desktop/iraceOpet/Instances/AP502.tsp"
[2] "/home/joca/Desktop/iraceOpet/Instances/AP403.tsp"
[3] "/home/joca/Desktop/iraceOpet/Instances/AP502.tsp"
[4] "/home/joca/Desktop/iraceOpet/Instances/AP403.tsp"
[5] "/home/joca/Desktop/iraceOpet/Instances/AP502.tsp"
[6] "/home/joca/Desktop/iraceOpet/Instances/AP403.tsp"
[7] "/home/joca/Desktop/iraceOpet/Instances/AP403.tsp"
> iraceResults$state$.irace$instancesList[index,"seed"]
[1] 688720299 830967740 688720299 830967740 688720299 830967740 830967740
> pair.id
[1] "1" "2" "3" "4" "5" "6" "7"
> index
[1] 2 1 2 1 2 1 1
> iraceResults$state$.irace$instancesList[pair.id,"seed"]
[1]  830967740  688720299 1410361231  380160547 2028762002 1142543995  467361645
```

Slika 16:. *Ispis ID-jeva parova (instanca, seme), i ispis instanci i semena koji se nalaze u datim parovima.*

Ukoliko je potrebno da se radi oporavak to se radi na osnovu podataka u objektu state. Kompletne podatke koji se nalaze u ovom objektu možemo izlistati sa `iraceResults$state`. Ukoliko želimo da vidimo probabilistički model za naš parametar MAX_ITERACIJA, možemo iskoristiti `iraceResults$state$model["MAX_ITERACIJA"]`. Videti sliku 17. Prve vrednosti konfiguracija se biraju uniformno, dok se vrednosti za konfiguracije nakon prve iteracije dobijaju na osnovu probabilističkog modela, koji se stalno ažurira, a pravi se na osnovu najboljih konfiguracija. Kod numeričkih parametara nove konfiguracije se biraju na osnovu probabilističkog modela uz pomoć odsečene normalne raspodele, te se vrši ažuriranje očekivanja i standardne devijacije, dok se kod kategoričkih vrši ažuriranje verovatnoća odabira određenih vrednosti iz diskretnog domena.

```
> iraceResults$state$model["MAX_ITERACIJA"]
$MAX_ITERACIJA
$MAX_ITERACIJA$`33`
[1]   19.93243 2825.00000

$MAX_ITERACIJA$`52`
[1]   19.93243 2778.00000
```

Slika 17: *Probabilistički model za parametar MAX_ITERACIJA.*

Ako želimo vrednosti koje vraća targetRunner nad test instancama, i vrednosti semena **pri testiranju,** možemo koristiti `iraceResults$testing$experiments` i `iraceResults$testing$seeds`, respektivno. Videti sliku 18. U ovom slučaju, kao što je već napomenuto, testiranje je vršeno samo za najbolju konfiguraciju čiji je ID 33.

```
> iraceResults$testing$experiments
        33
1t 39922.1
2t 49741.2
> iraceResults$testing$seeds
         1t         2t
2134431151 1969445510
```

Slika 18: *Izlazne vrednosti targetRunner-a za test instance, i semena koja su korišćena prilikom testiranja, uz te instance.*

Podaci za *experimentLog*, *softRestart* i *rejectedConfigurations* mogu se dohvatiti preko `iraceResults$experimentlog`, `iraceResults$softRestart` i `iraceResults$rejectedConfigurations`, respektivno.

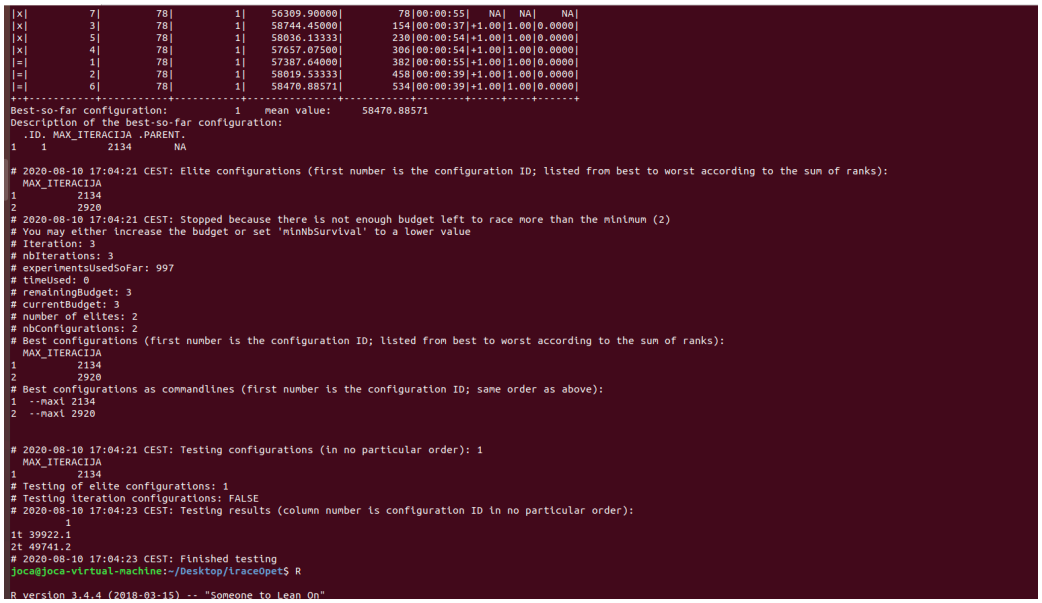
Posvetimo još malo pažnje matrici *experiments*. Preko `iraceResults$experiments` možemo izlistati izlazne vrednosti targetRunner-a u svim eksperimentima. U objektu *experiments* po kolonama idu konfiguracije (sve korišćene kroz sve iteracije; znamo da se elitne prenose iz jedne iteracije u drugu ali izračunavanje pomoću njih se vrši samo jednom nad jednim parom (*instanca*, *seme*), odnosno ako se konfiguracija prenese u novu iteraciju ne poziva se opet targetRunner za tu konfiguraciju nad parovima (*instanca*, *seme*) koji su već viđeni, već samo ako ima novih parova), dok po vrstama idu parovi (*instanca*, *seme*). Ne pokreće se targetRunner u svakoj iteraciji za sve parove (*instanca*, *seme*) zato negde stoji NA i takođe neke konfiguracije umiru tokom iteracije pa ne stižu da se ispitaju nad nekim parovima (*instanca*, *seme*), i u tom slučaju ide NA. Posmatraćemo novi primer. Svi podaci su isti kao u prethodnom primeru, samo je opet pokrenut irace. Na slici 19 se mogu videti rezultati irace-a.

```
> q()
Save workspace image? [y/n/c]: n
joca@joca-virtual-machine:~/Desktop/iraceOpet$ irace
#------------------------------------------------------------------------
# irace: An implementation in R of (Elitist) Iterated Racing
# Version: 3.4.1.9fcaeaf
# Copyright (C) 2010-2020
# Manuel Lopez-Ibanez     <manuel.lopez-ibanez@manchester.ac.uk>
# Jeremie Dubois-Lacoste
# Leslie Perez Caceres    <leslie.perez.caceres@ulb.ac.be>
#
# This is free software, and you are welcome to redistribute it under certain
# conditions.  See the GNU General Public License for details. There is NO
# WARRANTY; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
#
# irace builds upon previous code from the race package:
#     race: Racing methods for the selection of the best
#     Copyright (C) 2003 Mauro Birattari
#------------------------------------------------------------------------
# installed at: /home/joca/R/x86_64-pc-linux-gnu-library/3.4/irace
# called with:
Warning: A default scenario file './scenario.txt' has been found and will be read
# 2020-08-10 16:55:31 CEST: Initialization
# Elitist race
# Elitist new instances: 1
# Elitist limit: 2
# nbIterations: 2
# minNbSurvival: 2
# nbParameters: 1
# seed: 39246201
# confidence level: 0.95
# budget: 1000
# mu: 5
# deterministic: FALSE

# 2020-08-10 16:55:31 CEST: Iteration 1 of 2
# experimentsUsedSoFar: 0
# remainingBudget: 1000
# currentBudget: 500
# nbConfigurations: 83
# Markers:
     x No test is performed.
     c Configurations are discarded only due to capping.
     - The test is performed and some configurations are discarded.
     = The test is performed but no configuration is discarded.
     ! The test is performed and configurations could be discarded but elite configurations are preserved.
     . All alive configurations are elite and nothing is discarded
```

a)

```
     x No test is performed.
     c Configurations are discarded only due to capping.
     - The test is performed and some configurations are discarded.
     = The test is performed but no configuration is discarded.
     ! The test is performed and configurations could be discarded but elite configurations are preserved.
     . All alive configurations are elite and nothing is discarded

+-+----------+-----------+----------+--------------+----------+-------+-----+----+------+
| |   Instance|      Alive|      Best|      Mean best| Exp so far|  W time|  rho|KenW|  Qvar|
+-+----------+-----------+----------+--------------+----------+-------+-----+----+------+
|x|          1|         83|         1|    56309.90000|         83|00:00:38|   NA|  NA|    NA|
|x|          2|         83|         1|    58744.45000|        166|00:00:26|+0.47|0.73|0.2300|
|x|          3|         83|         1|    59555.96667|        249|00:00:26|+0.64|0.76|0.2313|
|x|          4|         83|         1|    58796.95000|        332|00:00:39|+0.73|0.80|0.2365|
|-|          5|         48|         1|    58361.46000|        415|00:00:38|+0.18|0.34|0.3206|
|-|          6|         38|         1|    58831.05000|        463|00:00:22|+0.00|0.17|0.1623|
+-+----------+-----------+----------+--------------+----------+-------+-----+----+------+
Best-so-far configuration:             1    mean value:     58831.05000
Description of the best-so-far configuration:
  .ID. MAX_ITERACIJA .PARENT.
1    1          2134       NA

# 2020-08-10 16:58:44 CEST: Elite configurations (first number is the configuration ID; listed from best to worst according to the sum of ranks):
  MAX_ITERACIJA
1          2134
2          2920
# 2020-08-10 16:58:44 CEST: Iteration 2 of 2
# experimentsUsedSoFar: 463
# remainingBudget: 537
# currentBudget: 537
# nbConfigurations: 78
# Markers:
     x No test is performed.
     c Configurations are discarded only due to capping.
     - The test is performed and some configurations are discarded.
     = The test is performed but no configuration is discarded.
     ! The test is performed and configurations could be discarded but elite configurations are preserved.
     . All alive configurations are elite and nothing is discarded

+-+----------+-----------+----------+--------------+----------+-------+-----+----+------+
| |   Instance|      Alive|      Best|      Mean best| Exp so far|  W time|  rho|KenW|  Qvar|
+-+----------+-----------+----------+--------------+----------+-------+-----+----+------+
|x|          7|         78|         1|    56309.90000|         78|00:00:55|   NA|  NA|    NA|
|x|          3|         78|         1|    58744.45000|        154|00:00:37|+1.00|1.00|0.0000|
|x|          5|         78|         1|    58036.13333|        230|00:00:54|+1.00|1.00|0.0000|
|x|          4|         78|         1|    57657.07500|        306|00:00:54|+1.00|1.00|0.0000|
|=|          1|         78|         1|    57387.64000|        382|00:00:55|+1.00|1.00|0.0000|
|=|          2|         78|         1|    58019.53333|        458|00:00:39|+1.00|1.00|0.0000|
|=|          6|         78|         1|    58470.88571|        534|00:00:39|+1.00|1.00|0.0000|
```

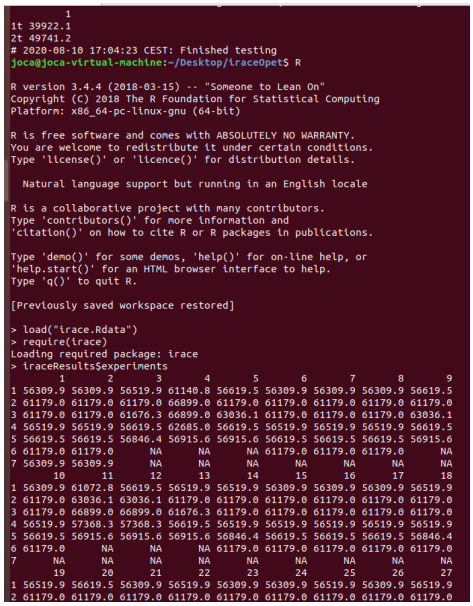
b)



c)

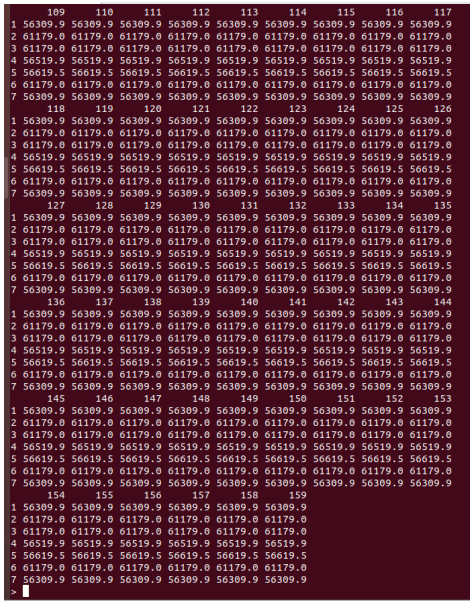
Slika 19: *Rezultati irace-a.*

A sada pokrenimo R okruženje i pogledajmo *iraceResults$experiments* na slici 20 (nije dat kompletan prikaz rezultata).



a)

b)



c)

Slika 20: *Deo ispisa marice experiments.*

Vidimo npr. sa slike 19. a) da je u prvoj iteraciji korišćeno 83 konfiguracije, jer *nbConfigurations*=83. Takođe iz tabele za 1. iteraciju vidimo da su se u 1. iteraciji posmatrali parovi (*instanca*, *seme*) sa ID-jevima od 1 do 6, dok se kasnije javlja i 7. Stoga za prve 83 konfiguracije u matrici *experiments* za par (*instanca*, *seme*) sa ID-jem 7 nemamo vrednosti. Osim za 1. i 2. konfiguraciju (1. i 2. kolona na slici 20. a)). Zašto?

Pa ako pogledamo sliku 19. b) i rezultate nakon prve iteracije videćemo da su 1. i 2. konfiguracija elitne, dakle one su prebačene u narednu iteraciju i tu su testirane za 7. par (*instanca*, *seme*).

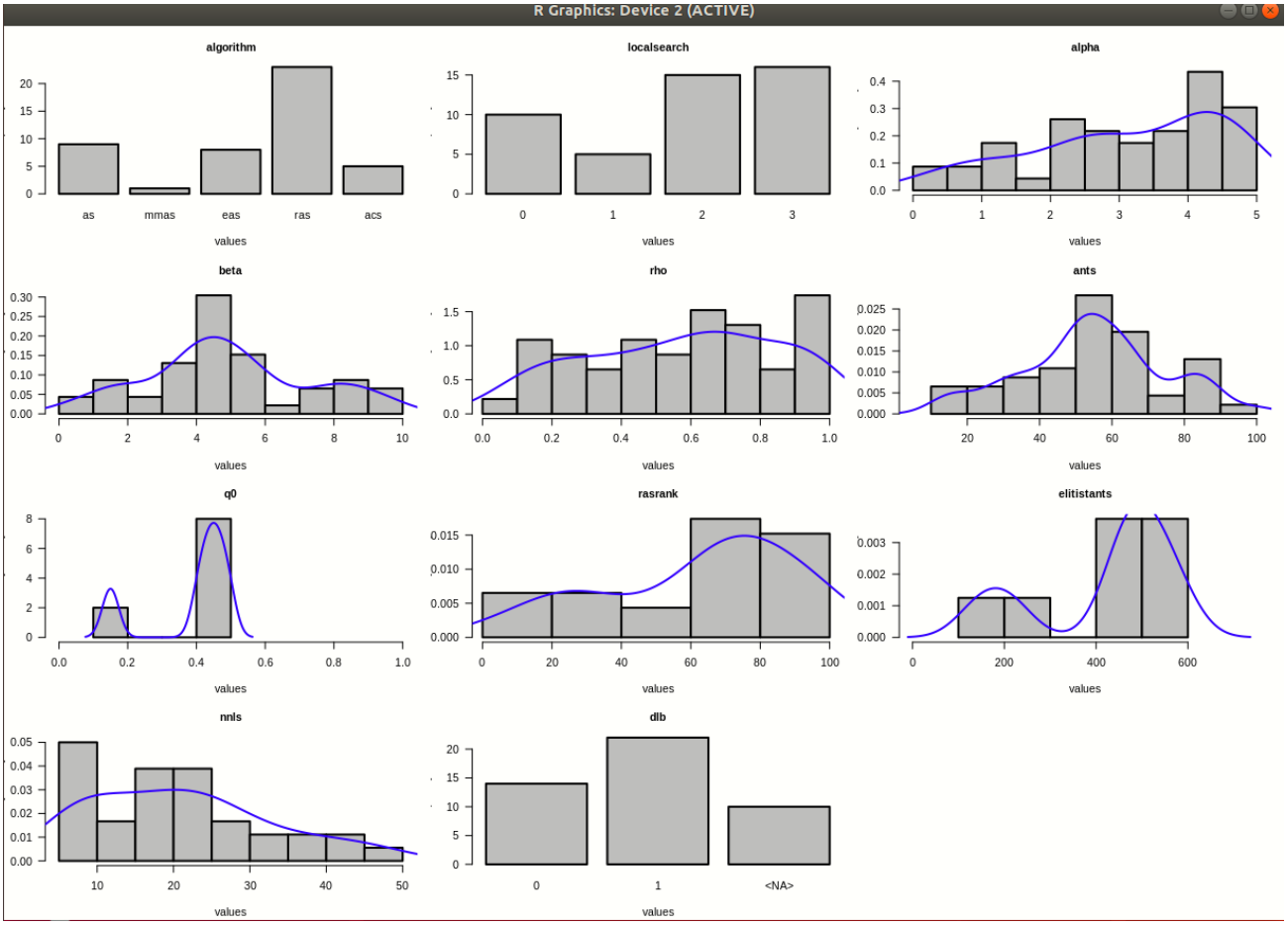
Analiza rezultata

Nakon što irace završi sa radom moguće je analizirati njegove rezultate. To možemo uraditi tako što ćemo vršiti statističke testove i generisati grafike u R-u. Prethodni deo je bio predstavljen na primeru sa manje parametara, radi lakšeg razumevanja. Kako bi analiza rezultata bila što kompletnije prikazana, analizu nećemo vršiti nad našim primerom, već ćemo koristiti primer koji je dat na putanji `$IRACE_HOME/experiments/acotsp` (gde je $IRACE_HOME putanja dobijena u sekciji "Instalacija"), a koji smo dobili prilikom instaliranja irace-a. Neće opet biti prikazivani kompletni rezultati irace-a, sve što je potrebno za analizu rezultata biće vidljivo kroz naredbe koje budu korišćene. Potrebno je samo naglasiti da ukoliko se koristi ovaj primer za anlizu rezultata, u "scenario.txt" potrebno je dodati opcije *testNbElites*, *testIterationElites*, *testInstancesDir* i *testInstancesFile*, jer bez *testInstancesDir* i *testInstancesFile* neće se sprovesti testiranje, bez *testNbElites* biće posmatrana samo krajnja najbolja konfiguracija, a bez *testIterationElites* biće posmatrane samo konfiguracije koje su bile najbolje nakon poslednje iteracije. Ovo nije neophodno za prve dve stavke, date u nastavku (dakle nije neophodno za *parameterFrequency* i paralelni prikaz vrednosti konfiguracija). Postupak za pokretanje ovog primera dat je sekciji 4.2 u [1].

Ukoliko želimo da dobijemo grafički prikaz raspodele koja će nam pokazati **koliko često je neki parametar biran iz nekog domena**, odnosno za kategoričke i ordinalne parametre, koliko često je uzimana određena vrednost, to možemo uraditi uz pomoć funkcije *parameterFrequency*, pozivom `parameterFrequency(iraceResults$allConfigurations,iraceResults$parameters)` kao na slici 21. Grafički prikaz, koji je rezultat ovog poziva, može se videti na slici 22.

```
>
> parameterFrequency(iraceResults$allConfigurations, iraceResults$parameters)
Plotting: algorithm
Plotting: localsearch
Plotting: alpha
Plotting: beta
Plotting: rho
Plotting: ants
Plotting: q0
Plotting: rasrank
Plotting: elitistants
Plotting: nnls
Plotting: dlb
>
```

Slika 21: *Poziv funkcije parameterFrequency.*

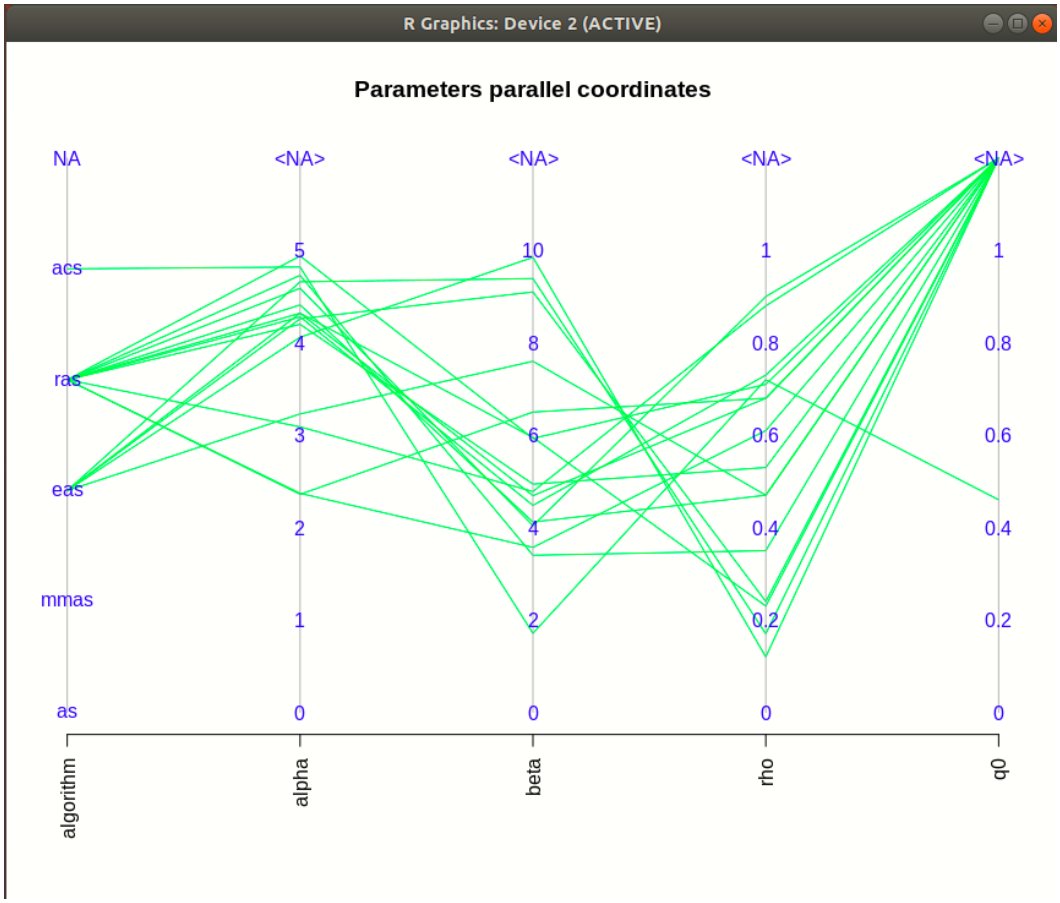


Slika 22: *Grafički prikaz koji se dobija pozivom funkcije parameterFrequency.*

Za **paralelni prikaz vrednosti konfiguracija** možemo koristiti funkciju *parallelCoordinatesPlot*. Ovo može biti korisno za analiziranje odnosa između parametara. Primer grafičkog prikaza za poslednje dve iteracije dat je na slici 24, a naredbe za njegovo dobijanje se mogu videti na slici 23. Prvom linijom komandi dobijen je ukupan broj iteracija koje je irace izvršio. U *configInLastTwoInstances* smeštene su konfiguracije za poslednje dve iteracije i na kraju je pozvana funkcija *parallelCoordinatesPlot*.

```
> lastIterationNb <- length(iraceResults$iterationElites)
> lastIterationNb
[1] 6
> configInLastTwoInstances <- getConfigurationByIteration(iraceResults = iraceResults, iterations = c(lastIterationNb - 1, lastIterationNb))
> configInLastTwoInstances
   .ID. algorithm localsearch alpha beta  rho ants   q0 rasrank elitistants
16   16       ras           3  4.28 4.48 0.73   41   NA      77          NA
29   29       ras           3  4.20 4.94 0.53   61   NA      66          NA
33   33       eas           3  3.23 7.60 0.47   52   NA      NA         503
34   34       acs           3  4.82 1.71 0.72   36 0.46      NA          NA
35   35       ras           2  4.32 4.69 0.68   32   NA      56          NA
36   36       ras           0  2.37 3.57 0.61   26   NA      67          NA
37   37       ras           3  3.09 4.78 0.88   52   NA      73          NA
38   38       ras           3  2.36 6.50 0.68   15   NA      90          NA
39   39       ras           3  4.73 3.40 0.35   83   NA      99          NA
40   40       eas           3  4.26 9.10 0.24   63   NA      NA         578
41   41       ras           3  4.94 5.93 0.71   67   NA      48          NA
42   42       eas           3  4.32 5.95 0.23   70   NA      NA         492
43   43       ras           3  4.41 4.12 0.47   60   NA      82          NA
44   44       ras           3  4.59 4.04 0.90   52   NA      69          NA
45   45       eas           3  4.06 9.85 0.12  100   NA      NA         543
46   46       eas           3  4.66 9.39 0.17   86   NA      NA         454
   nnls  dlb .PARENT.
16   25    1        1
29    9    1       16
33   12    1       16
34   25    0       16
35   24    0       16
36   NA <NA>       29
37   23    1       16
38   22    1       16
39    9    1       29
40    7    1       33
41    8    1       29
42   12    1       33
43   29    1       16
44   36    1       16
45    6    1       40
46   16    1       42
> parallelCoordinatesPlot(configInLastTwoInstances, iraceResults$parameters, param_names = c("algorithm", "alpha", "beta", "rho", "q0"), hierarchy = FALSE)
```

Slika 23: *Naredbe za paralelni prikaz vrednosti konfiguracija.*



Slika 24: *Paralelni prikaz vrednosti konfiguracija.*

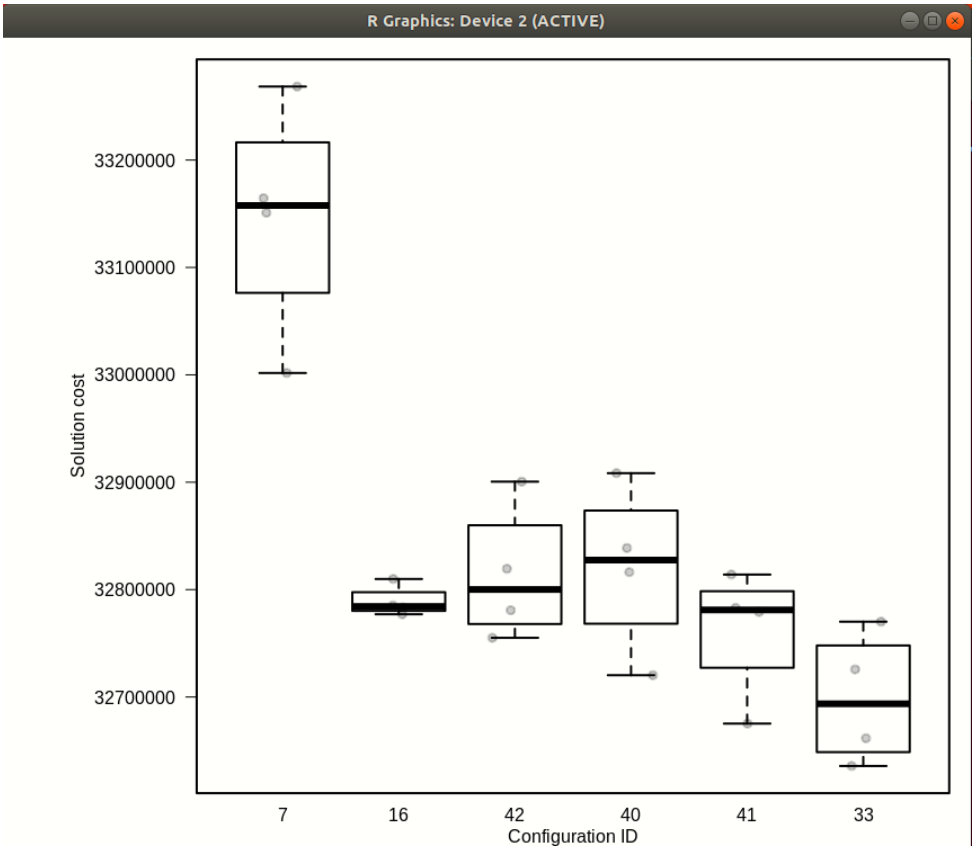
Ukliko se vršilo testiranje (tj. ukoliko smo bili zadali instance za testiranje u "scenario.txt") moguće je raditi analizu najboljih konfiguracija na osnovu njihovih performansi u fazi testiranja.

Možemo izvršiti neki **statistički test** radi poređenja. Primer je dat na slici 25. U *results* su smešteni rezultati testiranja najboljih konfiguarcija. U *conf* smešteni su takozvani nivoi, koji nam služe da grupišemo vrednosti. Prvi argument funkcije *gl* predstavlja broj nivoa (uzet je broj kolona matrice *results*, a to je zapravo broj najboljih konfiguracija), durgi predstavlja broj ponavljanja svakog nivao (uzet je broj vrsta matrice *results*, a to je zapravo broj test instanci), i treći se odnosi na oznake koje će biti dodeljene nivoima (uzeti su nazivi kolona matrice *resutls*, odnosno oznake najboljih konfiguracija). Nakon toga izvršen je Wilkoksonov test.

```
> results <- iraceResults$testing$experiments
> results
          7       16       42       40       41       33
1t 33001707 32785295 32780785 32720301 32675227 32635749
2t 33164407 32777041 32819389 32908414 32779156 32770122
3t 33150942 32809892 32900510 32838749 32783025 32725754
4t 33268451 32783336 32755115 32816222 32813995 32661621
> conf <- gl(ncol(results), nrow(results), labels = colnames(results))
> conf
 [1] 7  7  7  7  16 16 16 16 42 42 42 42 40 40 40 40 41 41 41 41 33 33 33 33
Levels: 7 16 42 40 41 33
> pairwise.wilcox.test(as.vector(results), conf, paired = TRUE, p.adj = "bonf")

        Pairwise comparisons using Wilcoxon signed rank test

data:  as.vector(results) and conf

   7 16 42 40 41
16 1 -  -  -  -
42 1 1  -  -  -
40 1 1  1  -  -
41 1 1  1  1  -
33 1 1  1  1  1

P value adjustment method: bonferroni
>
```

Slika 25: *Primer primene statističkog testa nad najboljim konfiguracijama, gledajući njihove rezultate prilikom faze testiranja.*

Na slici 26 dat je **boxplot** rezultata faze testiranja najboljih konfiguracija.

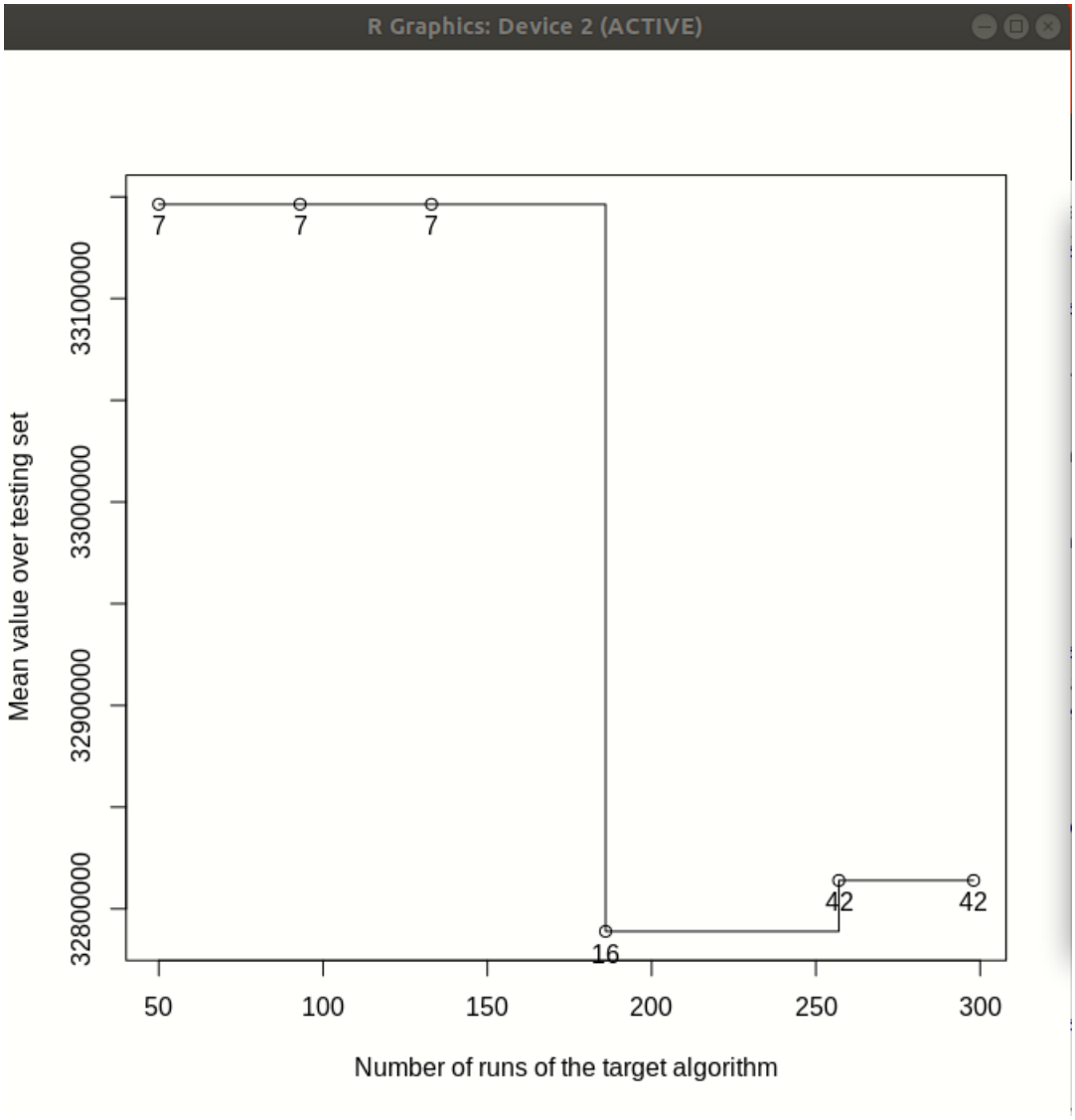


Slika 26: *Boxplot rezultata testiranja najboljih konfiguracija.*

Moguće je takođe grafički predstaviti **performanse najboljih konfiguracija po iteracijama** (dakle posmatramo po jednu najbolju konfiguraciju iz svake iteracije) nad test instancama. Na slici 27 mogu se videti naredbe koje je potrebno izvršiti. U *best* su izdvojene najbolje konfiguracije iz svake iteracije. U *values* su sačuvane srednje vrednosti rezultata pri testiranju tih konfiguracija. Dakle, za jednu konfiguraciju sabrani su svi rezultati dobijeni nad test instancama za tu konfiguraciju, i podeljeni sa brojem rezultata (ovaj broj je jednak broju test instanci, jer se u fazi testiranja targetRunner poziva za svaku konfiguraciju po jednom nad svakom instancom, za razliku od faze treninga). Pozvana je funkcija *plot* koja će generisati odgovarajući grafički prikaz. Kako bi prikaz bio jasniji dodate su određenje oznake uz pomoć funkcija *point* i *text*. Na slici 28 dat je grafički prikaz. Posmatrajmo npr. prvi čvor kod koga piše 7. Broj 7 označava ID konfiguracije koja je na kraju prve iteracije bila najbolja. Prva iteracija završena je nakon 50 eksperimenata, što je x koordinata posmatrane tačke, dok je y koordinata srednja vrednost rezultata testiranja konfiguracije čiji je ID 7 (ovo možemo proveriti tako što ćemo sabrati sve vrednosti iz prve kolone matrice *results* na slici 25 i podeliti sa 4). Slično, drugi čvor predstavlja odgovarajuće podatke za konfiguraciju koja je bila najbolje nakon druge iteracije itd.

```
> best <- as.character(iraceResults$iterationElites)
> best
[1] "7"  "7"  "7"  "16" "42" "42"
> values <- colMeans(iraceResults$testing$experiments[,elites])
> values
       7        7        7       16       42       42
33146377 33146377 33146377 32788891 32813950 32813950
> plot(fes, values, type="s", xlab="Number of runs of the target algorithm", ylab="Mean value over testing set")
> points(fes, values)
> text(fes, values, best, pos=1)
```

Slika 27: *Naredbe za generisanje grafičkog prikaza performansi najboljih konfiguracija po iteracijama, pri testiranju.*



Slika 28: *Grafički prikaz performansi najboljih konfiguracija po iteracijama, pri testiranju.*

LITERATURA

[1] The irace Package: User Guide. Manuel López-Ibáñez, Leslie Pérez Cáceres, Jérémie Dubois-Lacoste, Thomas Stützle and Mauro Birattari. IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium

[2] https://mlopez-ibanez.github.io/irace/index.html

[3] https://groups.google.com/g/irace-package

[4] M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, and M. Birattari. The irace package, iterated racing for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium, 2011. URL http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf.

[5] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, T. Stützle, and M. Birattari. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives,* 3:43–58, 2016. doi: 10.1016/j.orp.2016.09.002.

[6] https://cran.r-project.org/doc/manuals/r-release/R-admin.html

[7] Hutter, F., Hoos, H.H., Leyton-Brown, K., St¨utzle, T.: ParamILS: an automatic algorithm configuration framework. J. Artif. Intell. Res. 36, 267–306 (2009)

[8] L. Pérez Cáceres, M. López-Ibáñez, H. H. Hoos, and T. Stützle. An experimental study of adaptive capping in irace. In R. Battiti, D. E. Kvasov, and Y. D. Sergeyev, editors, *Learning and Intelligent Optimization, 11th International Conference, LION* 11, volume 10556 of Lecture Notes in Computer Science, pages 235–250. Springer, Cham, Switzerland, 2017. doi: 10.1007/978-3-319-69404-7_17.

[9] https://github.com/automl/GenericWrapper4AC/

[10] Brimberg J., Mladenović N., Todosijević R. et al. A basic variable neighborhood search heuristic for the uncapacitated multiple allocation p-hub center problem. Optim Lett 11, 313–327 (2017). https://doi.org/10.1007/s11590-015-0973-5

[11] http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/

[12] M. Schneider and H. H. Hoos. Quantifying homogeneity of instance sets for algorithm configuration. In Y. Hamadi and M. Schoenauer, editors, *Learning and Intelligent Optimization, 6th International Conference*, *LION* 6, volume 7219 of *Lecture Notes in Computer Science*, pages 190–204. Springer, Heidelberg, Germany, 2012. doi: 10.1007/978-3-642-34413-8_14.

[13] Birattari, Mauro & Yuan, Zhi & Balaprakash, Prasanna & Stützle, Thomas. (2009). Iterated F-race An Overview. Experimental Methods for the Analysis of Optimization Algorithms. 10.1007/978-3-642-02538-9_13.