Naučni projekat

“Flight Assignment Problem”

Autori: Aleksa Tomašević i Milica Ristić

Predmet : Računarska inteligencija

Matematički fakultet Univerziteta u Beogradu

Septembar 2025.

Sažetak

Fleet Assignment Problem (FAP) je centralni zadatak u planiranju avio-saobraćaja, koji se bavi dodelom tipova aviona letovima uz cilj minimizacije troškova i maksimizacije iskorišćenosti resursa. U radu se daje pregled literature i savremenih pristupa, kao i implementacija modela na primeru Aerodroma Optimal. Metodologija se zasniva na formulisanju problema putem MILP modela, primeni softverskih solvera i analizi različitih algoritamskih tehnika. Rezultati rada imaju potencijal da doprinesu kako akademskom razumevanju problema, tako i praktičnoj optimizaciji flote u realnim uslovima.

Ovim radom je pokazano da je na malim mrežama VNS je najbrži put do najboljeg cilja (isti optimum kao GA, brže izvođenje). Na srednjim mrežama VNS je često bolji uz prihvatljivo vreme, naročito kada dopustimo veći budžet. Na velikim mrežama GA je je bolji zbog vremena izvrsavanja, a VNS vredi kad je cilj “maksimum kvaliteta”. Ovaj metod je operativno stabilan (0 nepokrivenih ruta). Ne bavi se codeshare-om, stohastikom ni posadom — ta ograničenja su sledeća logična nadogradnja, u skladu sa heuristikama i integrisanim pristupima iz literature.

1. Uvod

"Fleet Assignment Problem" (FAP) predstavlja jednu od najvažnijih faza u planiranju i optimizaciji avio-saobraćaja. Suština problema je dodela tipova aviona unapred definisanim letovima, uz minimizaciju troškova i zadovoljavanje potražnje. Rani autori poput Abara [1] pokazali su značaj formalizacije problema putem mešovito celobrojno linearnog programiranja (MILP), dok su Rushmeier i Kontogiorgis [2] razvili metode koje su omogućile praktičnu primenu na mrežama srednje veličine. Barnhart et al. [3] i Sherali et al. [4] su dali sveobuhvatan pregled algoritama i modela, dok su novija istraživanja fokusirana na integrisane i stohastičke pristupe [5–8].

Za rešavanje MILP modela koriste se industrijski solveri poput IBM ILOG CPLEX i Gurobi, koji primenjuju napredne algoritme *branch-and-bound* i *cutting planes*. Pored tačnih metoda, u upotrebi su i heuristički pristupi kao što su genetski algoritmi i simulirano kaljenje, koji su i dalje u fokusu istraživača, u cilju dobijanja približnih rešenja u kraćem vremenu. S obzirom na rast kompleksnosti kod velikih mreža, posebna pažnja u naučnim istraživanjima se posvećuje dekompozicionim metodama (Bendersova dekompozicija i generisanje kolona), koje su se pokazale uspešnim u prethodnim istraživanjima [3,7]. *Analizirani su rezultati tačnih i heurističkih metoda, kao i prednosti integrisanih modela u poređenju sa klasičnim pristupima.*

U literaturi su prikazane mogućnosti i ograničenja različitih softverskih i algoritamskih pristupa u rešavanju FAP-a. Dosta se govori o značaju MILP formalizacije [1,4], analizira se uloga dekompozicionih metoda [3,7], a poslednjih godina su istraživanja usmerena na potencijal integrisanih i stohastičkih modela [5,6,8,9].

1.1 Cilj rada i potencijalni doprinosi

Cilj ovog projekta je da se pronadje optimalno softversko rešenje algoritma koji se na jednostavan načim pružiti mogućnosti što ekonomičnijeg planiranja letova. Potencijalni doprinos ovog rada se ogleda u demonstraciji kako teorijski modeli mogu biti prilagođeni realnim operativnim uslovima, u evaluaciji performansi različitih softverskih rešenja, i u davanju preporuka za dalja istraživanja i industrijsku primenu. Na ovaj način rad povezuje akademska istraživanja sa praktičnim izazovima avio-kompanija.

2. Metodologija

Metodološki okvir rada zasniva se na formulaciji problema kroz mešovito celobrojno linearno programiranje (MILP). Osnovni model obuhvata sledeća ograničenja: pokrivanje svih letova, usklađivanje kapaciteta sedišta sa potražnjom, ograničenja raspoloživosti aviona i vreme preokretanja između letova. Ciljna funkcija modela definiše minimizaciju ukupnih troškova, koji uključuju gorivo, operativne troškove i penale za nedovoljno zadovoljenje potražnje.

2.1. Opis rešenja

Problem i kodiranje rešenja

Razmatra se problem dodele aviona iz flote letovima (Fleet Assignment): za skup ruta R={0, …,R−1} i skup tipova aviona T={0,…,T−1} potrebno je odabrati tip aviona za svaku rutu, uz ograničenja raspoloživosti flote i operativnih pravila.

Odluka se kodira binarnom matricom

$$X = \left[x_{r,t} \right]_{R \times T}, \quad x_{r,t} \in 0,1$$

gde je xr, t=1 ako je za rutu r izabran tip aviona t, u suprotnom 0. U svakom redu (ruta) važi one-hot uslov:

$$\sum_{t=0}^{T-1} x_{r,t} = 1, \quad \forall r \in 0,\ldots,R-1$$

Dodela na nivou konkretnog aviona unutar tipa t (ima ih A_t) vrši se tokom simulacije rasporeda (evaluacije), dinamički, na osnovu vremena dostupnosti.

Parametri koji se koriste (tačno kao u kodu): broj ruta R, broj tipova T, broj aviona po tipu A_t , broj putnika po ruti P_r , trošak $F_{t,r}$, kapacitet C_t , trajanje leta T_r , vreme na zemlji H_t , cena karte po ruti price r , početno vreme, udaljenost rute dist r i domet tipa range t .

Planirano je da model bude primenljiv za bilo koju avio-kompaniju koja je stacionirana na matičnom aerodromu.

2.2. Matematički model ciljne funkcije

Cilj je minimizacija modifikujućih troškova uz maksimizaciju prihoda, uz prioritet maksimalne pokrivenosti ruta. U implementaciji se prvo minimizuje broj nepokrivenih ruta, zatim vrednost ciljne funkcije. Neka je:

- Cost(X) ukupan operativni trošak,
- Rev(X) ukupan prihod,
- Wait(X) kazna zbog čekanja (definisana u simulaciji rasporeda).

Tada je

$$min\left(Uncovered\left(X\right),Z\left(X\right)\right),Z\left(X\right)=\alpha\cdot Cost\left(X\right)-\beta\cdot Rev\left(X\right)+Wait\left(X\right),$$

pri čemu:

$$Cost\left(X\right)=\sum_{r=0}^{R-1}\sum_{t=0}^{T-1}F_{t,r}\,x_{r,t}$$

$$Rev\left(X\right)=\sum_{r=0}^{R-1}\sum_{t=0}^{T-1}\left(min\,P_r,C_t\cdot price_r\right)x_{r,t}$$

Kazna Wait nastaje iz simulacije kad ruta ne može da krene u najranijem trenutku; tada se trošku rute dodaje penal wait=10×Δt (u satima), gde je Δt vreme čekanja do prvog raspoloživog konkretnog aviona iz izabranog tipa. Napomena: u trenutnoj implementaciji penal za čekanje se dodaje u *fallback* grani (kada “odmah” polazak nije moguć).

Dodatna operativna pravila modelovana su u simulaciji:

- Domet: ako dist r > range t, tip t nije validan za rutu r.
- Turnaround: posle povratka dodaje se 45 min obračunskog zadržavanja.
- Dnevni limit sati leta: za tipove sa dometom ≤5000 km limit je 16 h, za ostale 20 h. Pri dosezanju limita uvodi se pauza od 6 h i reset akumulisanih sati za taj avion.
- Globalni throttle: nakon “odmah” dodele rute, globalno vreme se pomera za +5 min; ovo uvodi minimalni razmak između uzastopnih odluka na nivou sistema.

Evaluacija rešenja (simulacija rasporeda)

Za zadatu matricu X vrši se deterministička simulacija:

- vodi se evidencija vremena dostupnosti svakog konkretnog aviona available_times[t][a] i akumulisanih sati flight_hours[t][a];

- rute se obrađuju redom; za svaku rutu računa se najraniji trenutak polaska kao

  earliest=max(global_time, t,a minavailable_times[t][a]).
- Prva grana (“odmah”): ako postoji konkretan avion iz izabranog tipa (xr,t=1) koji je slobodan u earliest, ruta se dodeljuje tom tipu i avionu, vremena se ažuriraju (povratak + turnaround), dnevni limit se poštuje (u slučaju prekoračenja uvodi se pauza i reset sati), a globalno vreme pomera za +5 min.
- Fallback grana (“čekanje”): ako niko nije slobodan u earliest, bira se kandidat sa najmanjim vremenom dostupnosti; tada se dodaje penal wait u trošak rute i vrše se ažuriranja kao gore.
- Ako ni u jednoj grani ruta ne može biti dodeljena, ruta se beleži kao nepokrivena.

Funkcija vraća par (#Uncovered(X),Z(X)).

Generisanje rasporeda letova

Za dobijeni najbolji X se, istom simulacionom logikom kao u evaluaciji, konstruiše i izveštava detaljan raspored: destinacija, tip, ID aviona, vreme polaska i povratka, uz sortiranje po polasku ili povratku. Ovo omogućava neposredan uvid u operativni plan i proveru ograničenja.

Napomene o ograničenjima i dizajnerskim izborima

- Pokrivenost vs. optimalnost: pokrivenost ruta ima prioritet (minimizuje se pre svega broj nepokrivenih), dok se ekonomski kriterijum optimizuje sekundarno kroz Z(X).
- Operativna ograničenja (domet, turnaround, dnevni limiti, čekanje) modelovana su proceduralno u evaluaciji; time se izbegava uvođenje velikog broja eksplicitnih linearnih ograničenja.
- Globalni throttle od +5 min u “odmah” grani implementira minimalni razmak između odluka na nivou sistema; u fallback grani globalno vreme se ne pomera.

2.3. Brute Force (egzaktna pretraga prostora dodela)

Kodiranje rešenja (genotip)

Rešenje je binarna matrica dodela

$$X=\left[x_{r,t}\right]_{R\times T},\quad x_{r,t}\in 0,1$$

uz one-hot ograničenje po ruti:

$$\sum_{t=0}^{T-1} x_{r,t}=1,\quad \forall r\in 0,\ldots,R-1$$

Dakle, po ruti se bira tačno jedan tip aviona. Dodela konkretnog primerka (indeks a unutar tipa t) nije deo genotipa, već nastaje u simulaciji tokom evaluacije (na osnovu dostupnosti aviona, radnih sati i operativnih pravila).

Evaluacija (fitnes)

Brute Force koristi istu simulacionu evaluaciju kao GA/VNS/Greedy. Fitnes je leksikografski par (u,Z):

- u=#Uncovered(X) — broj nepokrivenih ruta dobijen simulacijom,
- ciljna funkcija:

$$Z(X) = alpha \cdot \sum_{r=0}^{R-1} \sum_{t=0}^{T-1} F_{t,r}\, x_{r,t} - beta \cdot \sum_{r=0}^{R-1} \sum_{t=0}^{T-1} (min\, P_r, C_t \cdot price_r)\, x_{r,t} + Wait(X)$$

Penal čekanja je:

Wait(X)=10·Δt(sati kasnjenja polaska u fallback grani).

- Ft,r – trošak tipa aviona t na ruti r,
- Pr – broj putnika na ruti r,
- Ct – kapacitet tipa aviona t,
- pricer – cena karte na ruti r,
- Wait(X) – penalizacija zbog čekanja (izražena u zavisnosti od dužine kašnjenja).

Operativna pravila u simulaciji:

- domet: distance r ≤ range t,
- turnaround: 45 min,
- dnevni limiti: 16h (kraći domet) / 20h (veći domet),
- pauza 6h po dostizanju dnevnog limita,
- globalni throttle: posle “odmah” dodele globalno vreme se uveća za 5 min.

Kriterijum poređenja je leksikografski: minimizuje se prvo u, potom Z.

Princip rada Brute Force algoritma

Brute Force iscrpno pretražuje ceo prostor dodela po tipovima pod one-hot ograničenjem. To praktično znači: za svaku rutu r bira se jedna od T kolona, tj. prostor ima T^R kombinacija.

1. Enumeracija svih dodela: generišu se sve TR kombinacije (t0,t1,…,tR−1)
2. Za svaku kombinaciju formira se one-hot matrica X.
3. Poziva se identična simulaciona evaluacija (ista kao u GA/VNS/Greedy).

4. Pamti se najbolje rešenje po leksikografskom poretku (u,Z).

Prednost: nalazi globalno najbolje rešenje u pretraženom prostoru (pošto je pretraga iscrpna).
Ograničenje: prostor eksplodira eksponencijalno sa R.

Složenost

Brute Force obilazi sve dodele:

$$\text{Broj evaluacija}=T^R.$$

Ako jedna evaluacija (simulacija) košta Eval, ukupno:

$$BF=O(T^R \cdot \text{Eval}).$$

Za samu evaluaciju važi ista aproksimacija kao i ranije:

$$\text{Eval}\approx O(R \cdot T \cdot \bar{A}),$$

gde je $\bar{A}$ prosečan broj konkretnih aviona po tipu razmatran u simulaciji. Dakle, asimptotski:

$$BF\approx O(T^R \cdot R \cdot T \cdot \bar{A}).$$

Ovo je egzaktno ali skupo: praktično isplativo samo za male R i umerene T.

Razlozi dizajna i upotreba

- Zlatni standard (benchmark): Brute Force daje globalno optimum u pretraženom modelu, pa služi kao referenca za poređenje heuristika (Greedy, VNS, GA) na malim instancama.
- Validacija evaluacije: pošto Greedy/VNS/GA dele identičnu evaluaciju, BF pomaže da se proveri konzistentnost simulacije i penalizacija (wait, limiti).
- Kalibracija parametara: pomoću BF optimuma mogu da se podešavaju α,β ili druge težine (npr. intenzitet kazne čekanja), kako bi se heuristike usmerile ka realnim kompromisima trošak–prihod.

2.4. Greedy konstruktivna heuristika

Kodiranje rešenja (genotip)

Rešenje je binarna matrica dodela:

$$X=\left[x_{r,t}\right]_{R\times T}, \quad x_{r,t}\in 0,1$$

sa one-hot ograničenjem po ruti:

$$\sum_{t=0}^{T-1} x_{r,t}=1, \quad \forall r\in 0,\ldots,R-1$$

Na nivou tipa (kolone) se bira tačno jedan tip po ruti. Dodela konkretnog aviona (indeks a unutar tipa t) ne ulazi u genotip, već nastaje u simulaciji na osnovu dostupnosti , radnih sati i operativnih pravila.

Evaluacija (fitnes)

Greedy i GA/VNS dele istu simulacionu evaluaciju. Fitnes je leksikografski par (u,Z):

- u=#Uncovered(X) – broj nepokrivenih ruta posle simulacije
- ciljna funkcija:

$$Z(X) = alpha \cdot \sum_{r=0}^{R-1} \sum_{t=0}^{T-1} F_{t,r}\, x_{r,t} - beta \cdot \sum_{r=0}^{R-1} \sum_{t=0}^{T-1} \left(min\, P_r, C_t \cdot price_r\right) x_{r,t} + Wait(X)$$

Wait(X) je penal za čekanje kada ruta ne može da krene u najranijem mogućem terminu i računa se kao:

$$Wait(X) = 10 \cdot \Delta t(\text{sati čekanja}).$$

Minimizacija je leksikografska: najpre minimizujemo u, zatim Z.

Princip rada Greedy algoritma

Greedy je konstruktivna heuristika: gradi matricu X red po red (rutu po rutu) u jednom prolazu.

1. Planiranje u najranijem terminu

Za trenutnu rutu r izračuna se

$$earliest\_time = \max(global\_time, \min \text{ po } t \text{ i } a \text{ za } available\_times)$$

Traži se među dozvoljenim tipovima (domet) i avionima onaj slobodan do earliest_time sa najmanjim troškom Ft,r.

- Vreme kada ce avion ponovo biti dostupan,
- Ukupno vreme leta (uz dnevne limite i eventualnu 6h pauzu),
- Globalno vreme polaska za sve avione (+ 5min).

2. Fallback: najraniji mogući polazak

Ako niko nije slobodan u earliest_time, bira se avion sa najranijim kandidatskim polaskom; ako je izjednačeno, uzima se manji trošak.
Trošak uključuje i wait penal.

3. One-hot održavanje

Za svaku rutu postavlja se jedna jedinica u koloni izabranog tipa. Time se dobija potpuno validna one-hot matrica X.

Složenost

Greedy obilazi rute jednom i po ruti proverava sve tipove i njihove avione:

$$Greedy \approx O\left(R \cdot T \cdot \overline{A}\right)$$

gde je $\bar{A}$ prosečan broj aviona po tipu. Operacije ažuriranja vremena i sati su O(1), pa ne utiču na složenost.

Razlozi dizajna

- Jednoprohodna konstrukcija daje brzo (i često razumno) polazno rešenje.
- Ista evaluacija kao GA/VNS omogućava da se Greedy koristi kao seed ili kao baseline za poređenje.
- Ugrađena operativna pravila (domet, turnaround, dnevni limiti, pauze) čine rešenje operativno validnim već u fazi konstrukcije.

Preporuke za upotrebu

- Kao inicijalni seed u GA (par komada za ubrzanje konvergencije).
- Kao baseline heuristika za poređenje.

2.5. Genetski algoritam

Kodiranje rešenja (genotip)

Rešenje se kodira binarnom matricom

$$X = [x_{r,t}]_{R \times T}, \quad x_{r,t} \in 0,1$$

uz one-hot uslov po ruti:

$$\sum_{t=0}^{T-1} x_{r,t} = 1, \quad \forall r \in 0, \ldots, R-1$$

Ovime se na nivou tipa aviona bira tačno jedan tip po ruti. Dodela konkretnog aviona (indeks a u okviru tipa t) nije deo genotipa već se vrši tokom evaluacije na osnovu trenutne dostupnosti.

Evaluacija (fitnes)

Fitnes jedinke je par (u,Z) gde je:

- u=#Uncovered(X) – broj nepokrivenih ruta dobijen simulacijom rasporeda (domet, dostupnost, turnaround, dnevni limiti, čekanje),

$$Z(X) = alpha \cdot \sum_{r=0}^{R-1} \sum_{t=0}^{T-1} F_{t,r}\, x_{r,t} - beta \cdot \sum_{r=0}^{R-1} \sum_{t=0}^{T-1} (min\, P_r, C_t \cdot price_r)\, x_{r,t} + Wait(X)$$

Minimizacija je leksikografska: prvo minimizujemo u, pa zatim Z. Penal za čekanje se računa samo kada ruta ne može da krene u najranijem trenutku i iznosi 10×Δt (u satima).

Inicijalizacija populacije

Polazna populacija kombinije:

- nasumične jedinke (za svaku rutu nasumično izabrati tip t),
- heurističke (greedy) jedinke dobijene konstrukcijom rasporeda po najnižem trošku u najranijem terminu.

Ovakva kombinacija daje balans između diverziteta i kvaliteta početnih rešenja.

Selekcija

Koristi se turnirska selekcija veličine k (u kodu k=5): slučajno se izdvoji k jedinki i izabere se najbolja po leksikografskom poretku fitnesa (u,Z). Turnirska selekcija ima nekoliko praktičnih osobina:

- robusna je na skaliranje fitnesa (nije potrebna normalizacija),
- lako se podešava pritisak selekcije preko k (veći k → jači selekcioni pritisak, brža, ali rizičnija konvergencija),
- jednostavna je za implementaciju.

Ukrštanje (crossover)

Ukrštanje je jedno-tačkasto po rutama. Neka su roditelji X(1) i X(2). Izabere se tačka preseka $c \in \{1,\ldots,R-1\}$ i formiraju se potomci:

$$Y^{(1)} = \left[X_{0:c^{(1)}} \quad X_{c:R^{(2)}} \right], \quad Y^{(2)} = \left[X_{0:c^{(2)}} \quad X_{c:R^{(1)}} \right]$$

Pošto je kodiranje po rutama nezavisno (svaki red je one-hot vektor dužine T), jedno-tačkasto ukrštanje čuva semantiku rešenja i ne narušava ograničenja, uz jednu praktičnu meru:

Popravka (repair)

Ako ukrštanje proizvede red koji nije one-hot (npr. zbog zaštite tipova ili slučajne degradacije), primenjuje se repair operator: postavi se nula na svim pozicijama, pa jedna jedinica na nasumičnom tipu t. U praksi, pošto roditelji zadovoljavaju one-hot, ukrštanje po celim redovima održava validnost, ali repair ostaje korisna zaštita i za mutaciju.

Mutacija

Mutacija je tačkastog tipa nad rutom: sa malom verovatnoćom p_m (u kodu ~0.05) svakoj ruti se nezavisno može promeniti tip:

- pronađe se trenutni tip $t_{old,}$
- izabere se $t_{new} \neq t_{old}$ uniformno iz $\{0,\ldots,T-1\} \setminus \{t_{old}\}$,
- red se ažurira tako da ostane one-hot.

Ovakva mutacija obezbeđuje lokalne skokove po prostoru rešenja bez narušavanja kodiranja. Vrednost pm utiče na diverzitet: veće $p_m \rightarrow$ veća raznolikost, ali i rizik od razbijanja dobrih obrazaca.

Elitizam

Najboljih E jedinki (u kodu $E=\lfloor POP/10 \rfloor$ i još prilagođavanje parnosti) se direktno kopira u narednu generaciju. Elitizam:

- garantuje monoton rast najbolje vrednosti kroz generacije (ne gubimo najbolje rešenje),
- skraćuje vreme konvergencije,
- uz prevelik E može smanjiti diverzitet; zato je tipičan opseg 5–15% populacije.

Kriterijum zaustavljanja

Primena je sa fiksnim brojem generacija G (npr. 500–1500).

Kontrola raznolikosti

Diverzitet obezbeđuju turnirska selekciju i elitizam, mutacija i nasumične jedinke u inicijalnoj populaciji,

Složenost i praktične napomene

Evaluacija jedne jedinke simulacijom je približno $O(R \cdot T \cdot A^{-})$ u najgorem slučaju (kandidati po ruti), gde je A^{-} prosečan broj konkretnih aviona po tipu koji se proveravaju; praktično je brže jer se preseče čim se nađe validan avion u “odmah” grani.

- Jedna generacija: $O(POP \cdot Eval)$.
- Ukupno: $O(G \cdot POP \cdot Eval)$.

Stabilnost: Pošto se koristi „globalni throttle“ (+5 min) samo u “odmah” grani, simulacija je deterministična za dati X. Za reprodukovljivost, postaviti seme generatora slučajnih brojeva pre inicijalizacije i mutacije.

Podešavanje parametara (smernice)

- Populacija (POP): 50–200 za srednje probleme; veća populacija daje više diverziteta ali linearno diže vreme.
- Generacije (G): 300–1500 u zavisnosti od željenog kvaliteta i vremena izvođenja.
- Turnir k: 3–7; veći k ubrzava konvergenciju uz manji diverzitet.
- Mutacija pm: 0.03–0.08; povećati ako primećuješ preranu konvergenciju.
- Elitizam E: 5–15% populacije, po mogućnosti paran (da se zadrži uparenost u petlji ukrštanja).

Razlozi dizajna

- Kodiranje po rutama omogućava jednostavno ukrštanje po redovima bez narušavanja one-hot ograničenja, a ocena rešenja preuzima složene realistične uslove (domet, turnaround, dnevni limiti, čekanje) kroz simulaciju.
- Leksikografski fitnes daje jasan prioritet pokrivenosti (operativna validnost), a tek potom ekonomskoj optimizaciji.
- Turnirska selekcija + elitizam su robusni i jednostavni, što je praktično za eksperimente i kasnije poređenje varijanti (npr. različitih mutacija ili dodatnih operatora).

2.6. Variable Neighborhood Search (VNS)

Kodiranje rešenja (fenotip)

Raspored se kodira binarnom matricom

$$X=[xr,t\ ]R\times T\ ,xr,t\ \in 0,1,$$

uz one-hot uslov po ruti:

$$\sum_{t=0}^{T-1} x_{r,t}=1,\quad \forall r\in 0,\dots,R-1$$

Dodela konkretnog aviona (indeks a unutar tipa t) nije deo genotipa, već se vrši u evaluaciji (simulaciji) na osnovu dostupnosti.

Evaluacija (fitnes)

Fitnes rešenja definisan je kao leksikografski par (u,Z):

- u=#Uncovered(X) – broj nepokrivenih ruta dobijen simulacijom,

$$Z(X) = alpha \cdot \sum_{r=0}^{R-1} \sum_{t=0}^{T-1} F_{t,r}\, x_{r,t} - beta \cdot \sum_{r=0}^{R-1} \sum_{t=0}^{T-1} (min\, P_r, C_t \cdot price_r)\, x_{r,t} + Wait(X)$$

Penal Wait(X) nastaje u fallback grani kada ruta ne može da poleti u najranijem terminu: dodaje se 10×Δt (sati čekanja).

Inicijalizacija

Generiše se nasumično one-hot rešenje: svaka ruta dobija slučajan tip t. Ovo pruža raznolik polazni plan koji VNS dalje poboljšava.

Okoline (neighborhoods)

Okolina je definisana brojem ruta koje se menjaju odjednom. U implementaciji se koristi lista veličina K=[1,2,3], što znači da posmatramo susede nastale promenom 1, 2 ili 3 rute u jednoj iteraciji. Promena rute je zamena izabranog tipa t nekim drugim $t' \neq = t$ (zadržava se one-hot).

Shaking

Shaking nasumično bira k različitih ruta i za svaku postavlja novi tip $t' \neq t$. Cilj je da se rešenje izbaci iz trenutnog lokalnog minimuma i uvede u novu oblast prostora rešenja pre lokalne pretrage. Shaking ne menja ništa van one-hot strukture.

Lokalna pretraga (intenzifikacija)

Lokalna pretraga sprovodi determinističu poboljšavajuću pretragu:

- prolazi se kroz sve rute r i sve alternative tipova $t \neq t_{tren}$,
- ocenjuje se nova jedinka sa jednom izmenom (move $r{:}t_{tren} \rightarrow t$),
- prihvata se najbolji poboljšavajući potez (po leksikografskom poretku (u,Z)),
- proces se ponavlja dok nema daljeg poboljšanja (1-opt lokalni optimum).

Ovo je klasičan „first/improving best move” pristup: garantuje monotono poboljšavanje fitnesa unutar jedne lokalne pretrage.

Kriterijum prihvatanja i raspored okolina

Glavna petlja vns radi sledeće:

1. Za svaku veličinu okoline $k \in K$:
    - Primenjuje se shaking (promeni k ruta),
    - evaluira i zatim se primeni lokalna pretraga na dobijenom rešenju,
    - dobijeni kandidat se uporedi sa trenutnim rešenjem po leksikografiji (u,Z).
2. Prihvatanje:
    - Ako je kandidat strogo bolji, prihvata se i prekida for-petlja po okolini (vraća se na najmanju okolinu u sledećoj iteraciji),

    - Ako je jednak po fitnesu, prihvata se sa verovatnoćom move_prob (u kodu 0.1) radi izlaska iz „platoa”.
3. Ako nijedna okolina ne dovede do (strogog ili probabilističkog) prihvatanja, iteracija se završava bez promene rešenja.

Ovo je standardni VNS raspored okolina: kreće se od najmanje (1) ka većoj (2, 3), uz restart na najmanju čim se nađe poboljšanje.

Složenost i praktične napomene

Evaluacija i lokalna pretraga:

- Evaluacija jedne jedinke (simulacija): približno

$$O(R \cdot T \cdot \overline{A})$$

  gde je A^- prosečan broj konkretnih aviona po tipu koje proveravamo. U praksi je brže jer se prekida čim se nađe validan avion u “odmah” grani.

- Lokalna pretraga (1-opt) u najgorem slučaju ispituje sve poteze $r \times (T-1)$ i može se ponavljati dok ima poboljšanja. Jedna „runda“ LS je oko

$$O(R \cdot (T-1) \cdot Eval) \approx O(R \cdot T \cdot R \cdot T \cdot \overline{A})$$

  ako se svaka kandidatura evaluira od nule. U praksi je bitno povoljnije jer:

    - poboljšanje se često nađe rano (prvo-prekid),
    - LS se zaustavlja čim nema poboljšavajućeg poteza,
    - male okoline (1, 2, 3 promene) ograničavaju „udaljenost“ od polaznog rešenja.

Shaking

Shaking radi na nivou kodiranja: promeni k ruta → O(k) manipulacija redova (konstantno u odnosu na evaluaciju).

Jedna VNS iteracija

Za svaku $k \in \{1,2,3\}$ radimo Eval+LS. Dakle, grubo:

$$O\left(\sum_{k \in K} (Eval + LS)\right)$$

s tim da je LS dominantan deo u najgorem slučaju. Kako se u ovom kodu prihvatanje dešava čim se nađe (strogo ili probabilističko) poboljšanje, prosečna složenost je mnogo manja od gornje granice.

Ukupno

Za num_iters = N:

$$O\left(N \cdot \sum_{k \in K} \left(Eval + LS\right)\right)$$

Na praktičnim instancama, zahvaljujući ranom prihvatanju i malim okolinama, VNS je efikasan i dobro balansira eksploraciju (shaking, veće okoline) i eksploataciju (lokalna pretraga).

Podešavanje parametara (smernice)

- Okoline K: krenuti od 1-promene; dodavanje 2 i 3 povećava sposobnost izlaska iz dubljih lokalnih minimuma. Prevelike okoline dižu vreme.
- Broj iteracija num_iters: 200–1000 je razuman raspon; više iteracija donosi stabilniji kvalitet.
- move_prob (npr. 0.05–0.2): veća vrednost pomaže da se preskoče platoi, ali može izazvati „šetanje“; 0.1 je dobar kompromis.
- Inicijalno rešenje: čista nasumična one-hot dodela je ok; alternativno, može se koristiti greedy seed ili više nasumičnih startova (multi-start VNS) za veću robusnost.

Razlozi dizajna

- Odvajanje evaluacije (koja modeluje realna operativna pravila) od operatora pretrage (koji rade na čistom one-hot kodiranju) čini metod modularnim i lakim za proširenje.
- Raspored okolina ($1 \rightarrow 2 \rightarrow 3$) s resetovanjem na 1 čim se nađe poboljšanje je klasičan VNS princip koji obezbeđuje balans između pretraživanja i lokalnog doterivanja.
- Probabilističko prihvatanje kada su rešenja jednaka po fitnesu pomaže u izbegavanju ciklusa i „zarobljavanja“ na platoima.

3. Rezultati

Postavka eksperimenata

U ovom odeljku porede se četiri pristupa: genetski algoritam (GA), varijabilna pretraga okoline (VNS), greedy i (po potrebi) brute-force (za vrlo male instance). Analize su izvedene na instancama: *small_balanced_s1*, *small_tight_fleet_s2*, *small_range_constrained_s3*, *small_turnaround_stress_s4*, *medium_hdly_m1*, *medium_ldhy_m2*, *condor*, *albatross_large* i *goliath_l110*.

Korišćene su dve konfiguracije (“budžeta” računanja):

- Mali budžet: GA 500 generacija (populacija 100), VNS 150 iteracija (veličina okoline 3).
- Veliki budžet: GA 1500 generacija (populacija 100), VNS 300 iteracija (okolina 3). Negativnija vrednost funkcije cilja je bolja. Sve dobijene solucije imaju 0 nepokrivenih ruta.

Tabela 1. Prikaz test primera

Naziv skupa	Veličina (R, T)	Flota (po tipu: broj aviona)	Kratak opis / namena	Ključna ograničenja / svojstva
small_balanced_s1	(8, 4)	ATR 72×2, Boeing 737×2, Airbus A330×1, Boeing 787×1	Mali, „uravnotežen“ miks kratkih i dugih ruta; dovoljna flota.	Umerena raspoloživost (2–2–1–1); rasponi pokrivaju sve udaljenosti; bazna validacija metoda.
small_tight_fleet_s2	(8, 4)	ATR 72×1, Boeing 737×1, Airbus A330×1, Boeing 787×1	Mali skup sa oskudnom flotom (po 1 avion).	Primarno ograničenje je kapacitet flote (1–1–1–1); stresira dodelu tipova pod „tesnom flotom“.
small_range_constrained_s3	(8, 4)	ATR 72×2, Boeing 737×1, Airbus A330×1, Boeing 787×1	Mali skup sa ograničenjima doleta po tipu.	Neke rute na granici/van dometa slabijih tipova; stresira range_t vs. distance_r.
small_turnaround_stress_s4	(8, 4)	ATR 72×2, Boeing 737×2, Airbus A330×1, Boeing	Stres na turnaround (zemaljske operacije).	Veći H_t (1.0–2.2 h) i zategnuti slotovi; testira

Naziv skupa	Veličina (R, T)	Flota (po tipu: broj aviona)	Kratak opis / namena	Ključna ograničenja / svojstva
		787×1		sabijanje rotacija.
medium_hdly_m1	(20, 5)	ATR 72×3, Boeing 737×4, Airbus A330×2, Boeing 787×2, Airbus A350×2	Srednji, high-density/long-haul & yield miks.	Više širokotrupaca; duži T_r i veće cene; kompromis kvalitet ↔ vreme.
medium_ldhy_m2	(20, 5)	ATR 72×3, Boeing 737×3, Airbus A330×2, Boeing 787×2, Airbus A350×1	Srednji, low-demand/high-yield scenario.	Niže P_r, povišene cene na delovima mreže; fokus na profitabilnost uz manju tražnju.
condor	(50, 4)	Airbus A320×12, Airbus A321×16, Airbus A321neo×8, Airbus A330-900neo×12	Realističan scenario jedne kompanije (EU + transatlantske).	Rasponi i kapaciteti uskotrupaca vs. širokotrupca; skaliranje na 50 ruta.
albatross_large	(60, 5)	ATR 72×6, Boeing 737×8, Airbus A330×4, Boeing 787×3, Airbus A350×3	Veliki, mešoviti globalni skup.	Raznolike distance/cene; mnogo konflikata pri izboru tipa/sati; stresira globalni kompromis.
goliath_l110	(110, 6)	ATR 72×8, Airbus A320×12, Airbus A321×10, Boeing 787-9×8, Airbus A350-900×8, Boeing 777-300ER×6	Veoma veliki stres-test skalabilnosti.	Postepen rast troškova i distanci; najjači računski pritisak (110 ruta, 6 tipova).

Tabela 2. Veliki budžet — najbolji postignuti “cilj”

dataset	GA	VNS	GREEDY	Best_overall
small_balanced_s1	-907.200000	-907.200000	-609.866667	-907.200000
small_tight_fleet_s2	-105.400000	-105.400000	-52.166667	-105.400000
small_range_constrained_s3	-538.700000	-538.700000	-199.566667	-538.700000
small_turnaround_stress_s4	-322.100000	-322.100000	-121.833333	-322.100000
medium_hdly_m1	-112.433333	-117.233333	382.066667	-117.233333
medium_ldhy_m2	-2305.166667	-2305.166667	-1863.166667	-2305.166667
albatross_large	-9284.266667	-9319.000000	-4712.400000	-9319.000000
condor	-8076.166667	-8076.166667	-6998.166667	-8076.166667
goliath_l110	-24607.113333	-24755.976667	-20527.346667	-24755.976667

U tabeli 2. je prikazano da:

- *Small* instance: i dalje GA = VNS po najboljem cilju, greedy zaostaje.
- *Medium_hdly_m1*: VNS preuzima vođstvo (najbolji cilj).
- *Medium_ldhy_m2*: GA = VNS (izjednačeno najbolje).
- *Albatross_large*: VNS prelazi GA (za razliku od malog budžeta).
- *Goliath_l110*: VNS ostaje najbolji (i sa većim budžetom).
- *Condor*: GA = VNS (isti optimum).

Implikacije.

Povećanje budžeta učvršćuje poziciju VNS na *m1*, *albatross_large* i *goliath_l110*. Na ostalim skupovima nema promene poretka (ili je nerešeno).

Tabela 3. Veliki budžet — mediana vremena izvrsavanja (s)

dataset	GA	GREEDY	VNS
albatross_large	68.828	0.001	193.000
condor	49.899	0.001	80.241
goliath_l110	137.414	0.002	851.098
medium_hdly_m1	23.406	0.001	18.630
medium_ldhy_m2	23.024	0.001	17.965
small_balanced_s1	12.028	0.000	2.632
small_range_constrained_s3	12.048	0.000	2.401
small_tight_fleet_s2	11.769	0.000	2.450
small_turnaround_stress_s4	11.940	0.000	2.265

U tabeli 3. je prikazano da:

- *Small*: VNS i dalje 4–5× brži od GA, pri istom cilju (≈2.3–2.7 s vs. ≈12 s).
- *Medium*: na *m1* VNS postaje i brži i bolji (≈18.6 s vs. ≈23.4 s). Na *m2*: VNS blago brži, kvalitet isti kao GA.
- *Large*: GA ostaje višestruko brži (npr. *goliath_l110*: ≈137 s GA vs. ≈851 s VNS; *albatross_large*: ≈69 s vs. ≈193 s).
-

Implikacije.

Za veliki budžet, VNS pokazuje najbolji balans na *m1* (i kvalitet i vreme), dok je na velikim instancama GA i dalje znatno efikasniji vremenski, uz povremeni gubitak malog dela kvaliteta (goliath, albatross).

Tabela 4. Mali budžet — najbolji postignuti “cilj”

dataset	GA	VNS	GREEDY	Best_overall
small_balanced_s1	-907.200000	-907.200000	-609.866667	-907.200000
small_tight_fleet_s2	-105.400000	-105.400000	-52.166667	-105.400000
small_range_constrained_s3	-538.700000	-538.700000	-199.566667	-538.700000
small_turnaround_stress_s4	-322.100000	-322.100000	-121.833333	-322.100000
medium_hdly_m1	-112.433333	-117.233333	382.066667	-117.233333
medium_ldhy_m2	-2305.166667	-2305.166667	-1863.166667	-2305.166667
albatross_large	-9344.233333	-9319.000000	-4712.400000	-9344.233333
condor	-8076.166667	-8076.166667	-6998.166667	-8076.166667
goliath_l110	-24108.900000	-24724.296667	-20527.346667	-24724.296667

U tabeli 4. je prikazano da:

- Na svim *small* instancama (s1–s4) GA i VNS dostižu isti optimum, a greedy primetno zaostaje (veći cilj).
- Na *medium_hdly_m1* VNS je najbolji, dok je na *medium_ldhy_m2* GA = VNS (izjednačeno najbolje).
- Na *albatross_large* GA je ekskluzivno najbolji pri malom budžetu; na *condor* je GA = VNS.
- Na *goliath_l110* VNS je najbolji (GA je blizu, greedy daleko slabiji).

Impikacije.

Za kratke vremenske budžete, VNS je sigurniji izbor kad je fokus na kvalitetu na *m1* i *goliath_l110*, dok GA ima prednost na *albatross_large*. Na *small* instancama oba metaheuristika su podjednako dobra po cilju.

Tabela 5. Mali budžet — mediana vremena izvršavanja (s)

dataset	GA	GREEDY	VNS
albatross_large	22.016	0.001	117.662
condor	16.113	0.001	47.101
goliath_l110	43.191	0.002	486.307
medium_hdly_m1	7.936	0.001	11.338
medium_ldhy_m2	7.804	0.001	10.821
small_balanced_s1	3.930	0.000	1.562
small_range_constrained_s3	4.011	0.000	1.517
small_tight_fleet_s2	4.052	0.000	1.573
small_turnaround_stress_s4	3.982	0.000	1.432

U tabeli 5. je prikazano da:

- Greedy je praktično “trenutan”, ali kvalitetno slab (videti prethodnu tabelu).
- Na *small* instancama VNS je ~2–3× brži od GA (≈1.5 s vs. ≈4 s), pri istom najboljem cilju.
- Na *medium* instancama GA je brži (npr. *m1*: ≈7.9 s GA vs. ≈11.3 s VNS), uz to da je VNS bolji po cilju na *m1*.
- Na *large* instancama GA je znatno brži (npr. *goliath_l110*: ≈43 s GA vs. ≈486 s VNS).

Impikacije.

Za ograničeno vreme na *small* instancama isplati se VNS (brži, isti kvalitet). Na *medium* je često kompromis: GA brži, VNS kvalitetniji (na m1). Na *large*, GA je jasna opcija ako vreme dominira kao ograničenje.

Sumarno poređenje i preporuke

Ovaj algoritam daje sledece kvalitete rešenja:

• *Small*: VNS = GA (najbolji cilj), greedy slab.

• *Medium*: VNS je najčešće bolji ili izjednačen, naročito na *m1*; *m2* je nerešeno.

• *Large*: ponekad VNS daje najbolji cilj (*goliath_l110*), ponekad GA (*albatross_large* pri malom budžetu), a često su nerešeni (*condor*).

Vremena izvođenja su sledeća:

• *Small*: VNS je brži uz isti cilj tako da je on preporučeni izbor.

• *Medium*: za apsolutni kvalitet na *m1* uz veći budžet najbolji je VNS, a za bržu procenu pri malom budžetu GA.

• *Large*: GA je 2.8×–6.2× brži, a birati VNS samo kada je cilj "maksimum kvaliteta" i vreme nije kritično.

• Greedy.
Praktično trenutan, ali sa velikim jazom u cilju. Najbolja upotreba je kao inicijalizacija GA/VNS.

Kada razmatramo stabilnost i robusnost može se istaći sledeće:

- Na svim pokretanjima nema nepokrivenih ruta.
- Za većinu instanci najbolji ciljevi se ponavljaju kroz tri pokretanja (mala varijansa). Manja odstupanja postoje (npr. *medium_hdly_m1* i *albatross_large* kod GA), ali ne menjaju zaključke.

Efekat povećanja budžeta

- GA: vreme raste ≈3× (500 → 1500 gen), a kvalitet se retko značajno menja.
- VNS: vreme značajno raste (posebno na *large*), ali se kvalitet povremeno poboljša (npr. *albatross_large* prelazi u korist VNS).

Preporučeni izbor po tipu instance

Small: VNS (isto najbolji cilj kao GA, 4–5× brži).

- Medium: *m1* ⇒ VNS (veći budžet) / GA (mali budžet ako je vreme kritično); *m2* ⇒ svejedno (blaga prednost VNS u vremenu).
- Large: GA za vremensku efikasnost; VNS samo kada se juri svaki procent kvaliteta (npr. *goliath_l110*).

Tabela 6. Primer optimalnog rasporeda generisanog za instancu *albatross_large*

Destination	Plane Type	Plane ID	Departure Time	Return Time
Berlin	ATR 72	T1-1	2025-01-20 05:30	2025-01-20 07:30
Paris	ATR 72	T1-2	2025-01-20 05:35	2025-01-20 07:47
Rome	ATR 72	T1-3	2025-01-20 05:40	2025-01-20 08:40
Vienna	ATR 72	T1-4	2025-01-20 05:45	2025-01-20 08:09
Zurich	ATR 72	T1-5	2025-01-20 05:50	2025-01-20 07:38
Amsterdam	ATR 72	T1-6	2025-01-20 05:55	2025-01-20 07:55
London	Boeing 737	T2-1	2025-01-20 06:00	2025-01-20 08:48
Copenhagen	Boeing 737	T2-2	2025-01-20 06:05	2025-01-20 09:05
Hamburg	Boeing 737	T2-3	2025-01-20 06:10	2025-01-20 08:28
Athens	Boeing 737	T2-4	2025-01-20 06:15	2025-01-20 10:51
Lisbon	Boeing 737	T2-5	2025-01-20 06:20	2025-01-20 11:08
Marrakesh	Boeing 737	T2-6	2025-01-20 06:25	2025-01-20 12:25
Canary Islands	Airbus A330	T3-1	2025-01-20 06:30	2025-01-20 11:24
Tel Aviv	Boeing 737	T2-7	2025-01-20 06:35	2025-01-20 11:47
Cairo	Boeing 737	T2-8	2025-01-20 06:40	2025-01-20 13:16
Doha	Airbus A330	T3-2	2025-01-20 06:45	2025-01-20 17:15
Dubai	Airbus A330	T3-3	2025-01-20 06:50	2025-01-20 17:56
Abu Dhabi	Airbus A350	T5-1	2025-01-20 06:55	2025-01-20 18:43
Jeddah	Boeing 787	T4-1	2025-01-20 07:00	2025-01-20 17:48
Riyadh	Boeing 787	T4-2	2025-01-20 07:05	2025-01-20 18:05
Delhi	Airbus A350	T5-2	2025-01-20 07:10	2025-01-20 21:10
Mumbai	Airbus A330	T3-4	2025-01-20 07:15	2025-01-20 21:45
Boston	Boeing 787	T4-3	2025-01-20 07:20	2025-01-20 20:08
New York	Airbus A350	T5-3	2025-01-20 07:25	2025-01-20 21:25

Destination	Plane Type	Plane ID	Departure Time	Return Time
Madrid	ATR 72	T1-1	2025-01-20 08:15	2025-01-20 11:51
Barcelona	ATR 72	T1-5	2025-01-20 08:23	2025-01-20 11:47
Prague	ATR 72	T1-2	2025-01-20 08:32	2025-01-20 10:32
Munich	ATR 72	T1-6	2025-01-20 08:40	2025-01-20 10:28
Brussels	ATR 72	T1-4	2025-01-20 08:54	2025-01-20 10:48
Amman	Boeing 737	T2-3	2025-01-20 09:13	2025-01-20 16:25
Milan	ATR 72	T1-3	2025-01-20 09:25	2025-01-20 11:49
Tunis	Boeing 737	T2-1	2025-01-20 09:33	2025-01-20 13:57
Algiers	Boeing 737	T2-2	2025-01-20 09:50	2025-01-20 14:26
Geneva	ATR 72	T1-6	2025-01-20 11:13	2025-01-20 13:25
Budapest	ATR 72	T1-2	2025-01-20 11:17	2025-01-20 13:53
Warsaw	ATR 72	T1-4	2025-01-20 11:33	2025-01-20 14:21
Rabat	Boeing 737	T2-4	2025-01-20 11:36	2025-01-20 16:48
Baku	Boeing 737	T2-5	2025-01-20 11:53	2025-01-20 19:17
Toronto	Airbus A330	T3-1	2025-01-20 12:09	2025-01-21 02:09
Tbilisi	Boeing 737	T2-7	2025-01-20 12:32	2025-01-20 19:32
Yerevan	Boeing 737	T2-6	2025-01-20 13:10	2025-01-20 20:34
Madeira	Boeing 737	T2-8	2025-01-20 14:01	2025-01-20 20:49
Larnaca	Boeing 737	T2-1	2025-01-20 14:42	2025-01-20 21:18
Heraklion	Boeing 737	T2-2	2025-01-20 15:11	2025-01-20 20:35
Malta	Boeing 737	T2-3	2025-01-20 17:10	2025-01-20 21:58
Reykjavik	Boeing 737	T2-4	2025-01-20 17:33	2025-01-20 23:21
Montreal	Airbus A330	T3-2	2025-01-20 18:00	2025-01-21 07:18
Singapore	Boeing 787	T4-1	2025-01-20 18:33	2025-01-21 11:51
Chicago	Airbus A330	T3-3	2025-01-20 18:41	2025-01-21 09:41

Destination	Plane Type	Plane ID	Departure Time	Return Time
Shanghai	Boeing 787	T4-2	2025-01-20 18:50	2025-01-21 10:14
Tokyo	Airbus A350	T5-1	2025-01-20 19:28	2025-01-21 11:28
Bangkok	Boeing 787	T4-3	2025-01-20 20:53	2025-01-21 12:29
Hong Kong	Airbus A350	T5-2	2025-01-20 21:55	2025-01-21 14:07
Los Angeles	Airbus A350	T5-3	2025-01-20 22:10	2025-01-21 14:40
Washington	Airbus A330	T3-4	2025-01-20 22:30	2025-01-21 12:12
Philadelphia	Airbus A330	T3-1	2025-01-21 02:54	2025-01-21 15:54
Johannesburg	Airbus A330	T3-2	2025-01-21 13:18	2025-01-22 04:48
San Francisco	Airbus A350	T5-1	2025-01-21 17:28	2025-01-22 09:28
Mexico City	Airbus A350	T5-2	2025-01-21 20:07	2025-01-22 12:55
Cape Town	Airbus A350	T5-3	2025-01-21 20:40	2025-01-22 13:40

3.1. Eksperimentalno okruženje

Eksperimenti su sprovedeni na računaru tipa HP Pavilion Gaming Laptop 17-c, opremljenom procesorom Intel Core i7-10750H sa dvanaest logičkih jezgara i radnim taktom do 5.0 GHz. Računar poseduje 16 GB radne memorije, kao i dve grafičke jedinice: integrisanu Intel CometLake-H UHD Graphics i diskretnu NVIDIA GeForce GTX 1650 Mobile. Operativni sistem korišćen tokom testiranja bio je Ubuntu 24.04.3 LTS (x86_64) sa kernelom verzije 6.14.0-29-generic. Za implementaciju i pokretanje algoritama korišćen je Python 3 interpretator, uz upotrebu standardnih biblioteka za numeričke izračune (pre svega *NumPy*) i pomoćnih modula za generisanje slučajnih brojeva i vizuelizaciju rezultata.

Genetski algoritam (GA) je evaluiran sa veličinom populacije od 100 jedinki, dok je broj generacija varirao u dva režima – 500 i 1500 iteracija. U implementaciji je primenjen operator mutacije, kao i elitistička selekcija, čime je obezbeđeno da najbolje jedinke budu prenete u naredne generacije. Pored njega, testiran je i algoritam VNS (Variable Neighborhood Search), pri čemu je veličina okoline bila postavljena na 3, a broj iteracija na 150 i 300.

Svaka instanca problema rešavana je tri puta, kako bi se smanjio uticaj slučajnosti u rezultatima. U izveštaju su prikazane vrednosti najboljeg postignutog cilja (objective function) i medijana vremena izvršavanja (median runtime), što omogućava pouzdaniju procenu performansi oba algoritma.

3.2. Diskusija rezultata u odnosu na literaturu

Krenuli smo od jednostavne, operativno utemeljene postavke: prvo obezbediti pokrivenost svih ruta (0 uncovered), pa tek onda peglati ekonomiju ciljem Z(X). Na malim instancama (s1–s4) i GA i VNS postižu isti najbolji cilj, dok je greedy slabiji; pritom je VNS 2–3× brži od GA. Ovakav obrazac — "metaheuristika daje brzo kvalitetno rešenje na manjim mrežama" — dobro se uklapa u nalaze Khanmirza i sar. [10] koji PMS-GA koriste za integrisani dizajn reda letenja i dodelu flote i pokazuju da su odstupanja od MILP optimuma ~1.8–3.0% uz višestruko kraći *runtime* na srednjim i velikim instancama. Naš VNS/GA rezultat na malim skupovima, gde je cilj optimalan ili blizu optimuma, praktično ilustruje istu poruku: heuristike su dobar "*first best*" za brz fokus na kvalitet bez eksplozije vremena.

Na srednjim instancama razlika se jasnije vidi: na m1 VNS daje bolji cilj i to uz vreme koje je čak i manje od GA kada povećamo budžet (≈18.6 s VNS prema ≈23.4 s GA), dok je na m2 kvalitet izjednačen, a VNS blago brži. Ovakav odnos između kvaliteta i vremena visoko je srodan zaključku iz rada o heurističkom pristupu (Simulated Annealing/Cuckoo Search + Monte Karlo) za integrisano planiranje, gde autori kažu da klasični optimizacioni softver "postaje neadekvatan kako broj letova raste", pa se prelazi na metaheuristike da bi se u prihvatljivom vremenu dobila robusna rešenja [6]. Naš deterministički simulator (sa penalom čekanja i turnaround pravilima) je jednostavniji od njihovog stohastičkog okvira, ali empirijski postiže isti cilj — stabilna izvodljivost i dobar balans kvaliteta i vremena kada se problem uveća.

Na velikim instancama slika je nijansirana: GA je 2.8–6.2× brži (npr. goliath_l110 ≈137 s GA vs. ≈851 s VNS), ali VNS povremeno izvuče nešto bolji cilj (goliath_l110), dok GA vodi na albatross_large pri malom budžetu. Ovo je u duhu Khanmirza i sar. [6], gde je ključna poruka predvidljivosti vremena metaheuristika naspram MILP-a i mogućnosti da se targetira "suboptimalno ali brzo" rešenje na velikim primerima . Naš rezultat praktično sugeriše isti savet za praksu: za "large" mreže GA je bolji izbor ako je vreme ključni resurs, a VNS ima smisla kada se juri još koji procenat cilja i vreme nije presudno.

U odnosu na integrisane modele sa zajedničkim odlučivanjem (raspored + flota + routing + codeshare), naš rad je svesno ostao uži: držimo se FAP-a sa operativnim pravilima u simulatoru (domet, turnaround, dnevni sati, penal čekanja), bez kodšeringa i bez eksplicitne stohastike. Kızıloglu i sar. [6] upravo naglašavaju da razdvajanje potproblema lako vodi u lokalne optimume i da je kod realnih mreža korisno integrisati raspored, flotu i rutiranje te modelovati neizvesnost (potražnja, vreme na zemlji); zato predlažu nelinearni stohastički MIP i metaheuristike za veće domete. Naš pristup je lakši za implementaciju u studentskom okruženju i lako se manipuliše parametrima, ali je i ograničen: deo dobitka koji integrisani modeli hvataju (npr. codeshare rasterećenje i robusnost) u ovom radu ostaje izvan dometa.

Još jedan ugao je posade. Wang i sar. zbijaju flotu i crew pairing baš zato što su "avion i posada dva glavna resursa", a sparivanje često pravi uska grla [7]. U našem simulatoru posade nema — dnevne limite sati letenja modelujemo na nivou aviona, ne na nivou posade — što je svesno uprošćavanje. To objašnjava zašto su kod nas razlike GA/VNS najvidljivije tek na mrežama gde dominiraju flota i rotacije, a ne kad "*crew"* pravila dominiraju — za taj deo bi integrisani pristup dao realniju sliku.

4. Zaključak

U ovom projektu razvijen je praktičan okvir za Fleet Assignment Problem koji spaja jasnu MILP motivaciju sa jednostavnom, operativno vernom simulacijom za evaluaciju rešenja. Ključni doprinos je postavljanje prioriteta na potpunu pokrivenost letova (0 uncovered), a zatim optimizaciju ekonomske funkcije sa penalom čekanja; time se dobijaju rasporedi koji su odmah operativno izvodljivi (domet, turnaround, dnevni limiti) bez uvođenja teških dodatnih ograničenja u model. Drugi doprinos je poređenje tri algoritamska pristupa na raznolikom skupu instanci: greedy, VNS i GA, uz "zlatni standard" brute-force na malim primerima za validaciju. Rezultati su jasni: na malim skupovima VNS i GA dosežu isti najbolji cilj, ali je VNS 2–3× brži; na srednjim instancama VNS često daje bolji cilj (naročito m1) uz razumno vreme; na velikim mrežama GA je višestruko brži, dok VNS povremeno donosi još koji procenat kvaliteta. Treći doprinos je praktična smernica kako birati metod: VNS za "small" i kada je kvalitet prioritet uz srednji budžet, GA za "large" i brza izvođenja, greedy kao trenutan baseline/seed. Na svim eksperimentima održana je izvodljivost i stabilnost cilja, što potvrđuje konzistentnost evaluacije. Predloženi okvir je i lako proširiv: može se nadograditi integracijom posada, codeshare odluka i stohastičkih scenarija potražnje, ali i bez toga već pruža upotrebljiv alat za brzu i pouzdanu dodelu flote u realnim uslovima.

5. Literatura

1. Abara, J. Applying integer linear programming to the fleet assignment problem. Interfaces. 1989;22(3):4–20.

2. Rushmeier, R., Kontogiorgis, S. Advances in the Optimization of Airline Fleet Assignment. Transportation Science. 1997;31(2):159–169.

3. Barnhart, C, Boland NL, Clarke LW et al. Flight string models for aircraft fleeting and routing. Transportation Science. 1998;32(3):208–220.

4. Sherali, H. D., et al. Airline Fleet Assignment Concepts, Models, and Algorithms. Eur J Oper Res. 2006;172(1):1–30.

5. Unal, Y. Z., Sevkli M, Uysal,O Turkyilmaz Al. A new approach to fleet assignment and aircraft routing problems. Transportation Research Procedia 2021; 59: 67-75.

6. Kızıloglu, K., Sakallı, U. S. Integrating Flight Scheduling, Fleet Assignment, and Aircraft Routing Problems with Codesharing Agreements under Stochastic Environment. Aerospace. 2023;10(12):1031.

7. Wang, D., Guo, S., Wang, W., Liang, Z. Tactical fleet assignment and crew pairing problem with crew flight time allocation. 2022.

8. Xu Y. Perspectives on Modelling Airline Integrated Scheduling Problem: a Review on State-of-the-Art Methodologies, Journal of the Air Transport Research Society, Volume 3, 2024.

9. Yan, C., Barnhart, C., Vaze, V. Choice-Based Airline Schedule Design and Fleet Assignment: A Decomposition Approach. Transport Sci. 2022;56(6):1410–31

10. Khanmirza E, Nazarahari M, Haghbeigi M. A heuristic approach for optimal integrated airline schedule design and fleet assignment with demand recapture. Appl Soft Comput. 2020;96:106681.