Optimizacija ruta dostavnih vozila na primjeru grada Rijeke

Buratović, Emil

SVEUČILIŠTE U RIJECI

POMORSKI FAKULTET

EMIL BURATOVIĆ

OPTIMIZACIJA RUTA DOSTAVNIH VOZILA NA PRIMJERU GRADA RIJEKE

DIPLOMSKI RAD

Rijeka, 2024.

SVEUČILIŠTE U RIJECI

POMORSKI FAKULTET

OPTIMIZACIJA RUTA DOSTAVNIH VOZILA NA PRIMJERU GRADA RIJEKE

OPTIMIZATION OF DELIVERY VEHICLE ROUTES: A CASE STUDY OF THE CITY OF RIJEKA

DIPLOMSKI RAD

MASTER THESIS

Kolegij: Algoritmi i strukture podataka

Mentor: *izv. prof. dr. sc.* Marko Gulić

Student: Emil Buratović

Studijski smjer: Elektroničke i informatičke tehnologije u pomorstvu

JMBAG: 0112081263

Rijeka, rujan 2024.

Student: Emil Buratović

Studijski program: Elektroničke i informatičke tehnologije u pomorstvu

JMBAG: 0112081263

IZJAVA O SAMOSTALNOJ IZRADI DIPLOMSKOG RADA

Kojom izjavljujem da sam diplomski rad s naslovom

Optimizacija ruta dostavnih vozila na primjeru grada Rijeke

(naslov diplomskog rada)

izradio samostalno pod mentorstvom *izv. prof. dr. sc.* Marka Gulića.

U radu sam primijenio metodologiju izrade stručnog/znanstvenog rada i koristio literaturu koja je navedena na kraju diplomskog rada. Tuđe spoznaje, stavove, zaključke, teorije i zakonitosti koje sam izravno ili parafrazirajući naveo u diplomskom radu na uobičajen, standardan način citirao sam i povezao s fusnotama i korištenim bibliografskim jedinicama, te nijedan dio rada ne krši bilo čija autorska prava. Rad je pisan u duhu hrvatskoga jezika.

Student 

(potpis)

Emil Buratović

Student: Emil Buratović

Studijski program: Elektroničke i informatičke tehnologije u pomorstvu

JMBAG: 0112081263

IZJAVA STUDENTA – AUTORA
O JAVNOJ OBJAVI OBRANJENOG DIPLOMSKOG RADA

Izjavljujem da kao student – autor diplomskog rada dozvoljavam Pomorskom fakultetu u Rijeci da ga trajno javno objavi i besplatno učini dostupnim javnosti u cjelovitom tekstu u mrežnom digitalnom repozitoriju Pomorskog fakulteta.

U svrhu podržavanja otvorenog pristupa diplomskim radovima trajno objavljenim u javno dostupnom digitalnom repozitoriju Pomorskog fakulteta, ovom izjavom dajem neisključivo imovinsko pravo iskorištavanja bez sadržajnog, vremenskog i prostornog ograničenja mog diplomskog rada kao autorskog djela pod uvjetima *Creative Commons* licencije CC BY Imenovanje, prema opisu dostupnom na http://creativecommons.org/licenses/

Student – autor 

_____

(potpis)

Emil Buratović

SAŽETAK

U ovom diplomskom radu analizirana je učinkovitost različitih algoritama za optimizaciju ruta dostavnih vozila u urbanim sredinama, s posebnim naglaskom na prometne uvjete u gradu Rijeci. Prometne gužve i sve veći broj dostavnih vozila u gradskim jezgrama predstavljaju ozbiljan problem koji zahtijeva efikasna rješenja. U radu su analizirani jednostavni algoritam, pohlepni algoritam i algoritam afričkog bizona, s ciljem utvrđivanja njihove učinkovitosti u optimizaciji ruta u stvarnim uvjetima. Algoritam afričkog bizona se pokazao kao najučinkovitiji, posebno u složenim scenarijima s većim brojem dostavnih vozila. U radu su također razmatrana ograničenja algoritama te moguće prilagodbe i poboljšanja.

Ključne riječi: algoritam, algoritam afričkog bizona, pohlepni algoritam, optimizacija ruta, promet, Rijeka.

SUMMARY

This thesis analyses the efficiency of various algorithms for optimizing delivery vehicle routes in urban areas, with a particular focus on traffic conditions in the city of Rijeka. Traffic congestion and the increasing number of delivery vehicles in city centres pose serious challenges that require efficient solutions. The study examines a simple algorithm, a greedy algorithm and an African Buffalo algorithm to determine their effectiveness in route optimization under real-world conditions. The African Buffalo algorithm proved to be the most efficient, especially in complex scenarios involving a larger number of delivery vehicles. The thesis also discusses the limitations of the algorithms and potential adaptations and improvements.

Keywords: algorithm, African Buffalo, routing optimization, traffic, Rijeka.

SADRŽAJ

1. UVOD

U posljednjih nekoliko desetljeća, prometne gužve u urbanim područjima postale su ozbiljan problem koji utječe na kvalitetu života stanovnika, ekonomske aktivnosti i efikasnost transportnih sustava. S povećanjem broja stanovnika i gospodarskih aktivnosti u gradovima, dolazi do sve većeg opterećenja prometne infrastrukture. Osim negativnog utjecaja na svakodnevne aktivnosti stanovništva, prometne gužve rezultiraju povećanom potrošnjom goriva, emisijom štetnih plinova i smanjenjem ukupne produktivnosti. Posebno su pogođeni logistički sektori, gdje kašnjenja u dostavi robe mogu imati dalekosežne ekonomske posljedice. S obzirom na sve veću prisutnost dostavnih vozila u gradskim jezgrama, posebice u gradovima poput Rijeke, optimizacija ruta postaje ključni izazov. Povećani obujam e-trgovine, rast potreba za brzim dostavama te ograničeni resursi prometne infrastrukture doveli su do toga da optimizacija postane nužnost za osiguranje nesmetanog protoka prometa i smanjenje operativnih troškova. Efikasno planiranje ruta dostavnih vozila može značajno smanjiti gužve, skratiti vrijeme dostave i smanjiti negativne ekološke učinke. Cilj ovog rada je analizirati učinkovitost različitih algoritama (jednostavni algoritam, pohlepni algoritam i algoritam afričkog bizona) za optimizaciju rute dostavnih vozila, s posebnim fokusom na specifičnosti prometne situacije u Rijeci. Rijeka, kao jedan od najvećih gradova u Hrvatskoj, suočava se s brojnim prometnim izazovima, uključujući uske gradske ulice, visoku koncentraciju vozila u gradskoj jezgri te vremenske uvjete koji često otežavaju promet. Razumijevanje načina na koji različiti algoritmi reagiraju na ove izazove ključno je za poboljšanje prometne situacije. Proučavajući neke osnovne algoritme, te naprednije pristupe poput algoritma afričkog bizona, ovaj rad će pružiti uvid u to koji su algoritmi najefikasniji u rješavanju problema optimizacije ruta u stvarnim uvjetima. S obzirom na sve veći prometni kaos i zagađenje u urbanim sredinama, te sve veće zahtjeve za efikasnijim dostavnim sustavima, optimizacija rute postaje ključno pitanje za sve veće gradove. Poseban naglasak bit će stavljen na algoritam afričkog bizona te njegove prilagodbe specifičnim prometnim uvjetima grada Rijeke, uzimajući u obzir lokalne karakteristike koje mogu utjecati na izvedbu ovog algoritma. Također, bit će istražene mogućnosti daljnjih poboljšanja i hibridizacije algoritma kako bi se osigurala maksimalna efikasnost u primjeni u stvarnom svijetu.

1.1. PROBLEM, PREDMET I OBJEKTI ISTRAŽIVANJA

Problem istraživanja ovog rada leži u izazovu optimizacije ruta dostavnih vozila u urbanim sredinama, s posebnim naglaskom na grad Rijeku. S obzirom na složene prometne uvjete i sve veći broj vozila, potrebno je pronaći optimalne algoritme koji mogu smanjiti vrijeme dostave i zagušenja u prometu. Predmet istraživanja su algoritmi za optimizaciju rute, dok su objekti istraživanja stvarni podaci o prometu i dostavnim vozilima u Rijeci.

1.2. RADNA HIPOTEZA

Radna hipoteza ovog istraživanja je da algoritam afričkog bizona, zbog svoje složenosti i prilagodljivosti, pruža superiorna rješenja za optimizaciju ruta dostavnih vozila u usporedbi s jednostavnijim algoritmima, osobito u uvjetima koji uključuju veliki broj vozila i složene prometne situacije.

1.3. SVRHA I CILJEVI ISTRAŽIVANJA

Svrha ovog istraživanja je identificirati najučinkovitiji algoritam za optimizaciju rute dostavnih vozila u urbanim sredinama poput Rijeke. Ciljevi istraživanja uključuje analizu učinkovitosti tri algoritma (jednostavni algoritam, pohlepni algoritam i algoritam afričkog bizona), te testiranje njihove izvedbe u različitim scenarijima s različitim brojem vozila te ocjenu njihovog utjecaja na ukupnu prometnu situaciju.

1.4. ZNANSTVENE METODE

U ovom radu korištene su sljedeće znanstvene metode: komparativna analiza, simulacija, empirijsko istraživanje, heuristička analiza i statistička analiza podataka.

1.5. STRUKTURA RADA

Rad se sastoji od šest glavnih poglavlja. U drugom poglavlju objašnjavaju se teoretske osnove algoritama za optimizaciju ruta. Treće poglavlje analizira postojeće pristupe i algoritme u optimizaciji ruta, dok četvrto poglavlje opisuje metodologiju istraživanja. Peto poglavlje

predstavlja rezultate testiranja algoritama i analizira njihovu učinkovitost. Na kraju, u šestom poglavlju, donose se zaključci istraživanja i preporuke za buduće radove.

2. TEORIJSKA OSNOVA

Ovo poglavlje se fokusira na teoretske osnove optimizacije ruta u logistici, analizu problema u transportnoj optimizaciji te pregled različitih algoritama koji se koriste za rješavanje tih problema. Poseban naglasak je stavljen na metaheurističke pristupe i algoritme strojnog učenja, s obzirom na njihovu važnost za analizu koja se provodi u ovom radu.

2.1. OPTIMIZACIJA RUTA U LOGISTICI

Optimizacija ruta je ključni element za poboljšanje učinkovitosti i smanjenje troškova u logistici. [1]

Ovaj proces uključuje odabir najkraćih ili najbržih puteva za vozila koja opskrbljuju različite lokacije. S obzirom na složenost logističkih sustava, optimizacija ruta ne samo da smanjuje operativne troškove, već i doprinosi boljoj iskorištenosti resursa i većoj zadovoljstvu klijenata. U praktičnoj primjeni, optimizacija ruta uzima u obzir više faktora poput udaljenosti između točaka, prometa, kapaciteta vozila, te specifičnih ograničenja kao što su vremenski prozori unutar kojih isporuke moraju biti izvršene. U kontekstu dostave, optimizacija ruta nije samo tehnički izazov već i strateški proces koji može značajno utjecati na konkurentnost poduzeća na tržištu. S razvojem tehnologije i rastom složenosti logističkih sustava, potreba za naprednim metodama optimizacije ruta postaje sve važnija.

2.2. PROBLEMI OPTIMIZACIJE U TRANSPORTU

Problemi optimizacije u transportu, poput problema rasporeda vozila (eng. *Vehicle Routing Problem* - VRP), predstavljaju jedan od najvećih izazova u modernoj logistici. [2]

VRP je klasični problem u kojem je potrebno pronaći optimalne rute za vozila kako bi opskrbili više klijenata s minimalnim troškovima. Ovaj problem postaje još složeniji kada se uvedu dodatni čimbenici poput vremenskih zahtjeva, ograničenja kapaciteta vozila, ili potrebe za povratkom vozila u postaju nakon obavljene isporuke. Problem rasporeda vozila s vremenskim ograničenjima (eng. *Vehicle Routing Problem with Time Windows* - VRPTW) dodaje složenost osnovnom VRP-u tako što uvodi vremenske prozore unutar kojih svaka isporuka mora biti obavljena. Ovo ograničenje može značajno utjecati na raspodjelu resursa i povećati ukupne operativne troškove.

S obzirom na kompleksnost ovih problema, razvijeni su različiti algoritmi i metode koje omogućuju rješavanje problema optimizacije ruta u transportu na što učinkovitiji način. Ovi problemi su posebno važni u velikim gradovima kao što je Rijeka, gdje je optimizacija ruta ključna za smanjenje prometnih gužvi i poboljšanje učinkovitosti dostavnih službi.

2.3. OPIS PROBLEMA OPTIMIZACIJE DOSTAVNE RUTE U GRADSKOM SREDIŠTU GRADA RIJEKE

U urbanim sredinama poput grada Rijeke, optimizacija dostavnih ruta postaje složen problem zbog nekoliko faktora kao što su prometne gužve, uski gradski prolazi, ograničena parkirna mjesta te veliki broj dostavnih vozila koja se kreću u istom vremenskom intervalu. Budući da je fokus ovog rada optimizacija dostavnih ruta dvaju (ili više) vozila koja mogu imati identične dostavne rute unutar gradskog središta. Problem nastaje kada oba vozila pokušavaju izvršiti dostave na istim lokacijama u isto vrijeme, što može rezultirati dodatnim čekanjem te povećanjem ukupnog vremena provedenog u gradu.
Na primjer, postoje dva dostavna vozila, Vozilo 1 i Vozilo 2, koja moraju dostaviti robu na tri ista mjesta u centru Rijeke. Dostavna mjesta označena su sa M1, M2 i M3. Vrijeme iskrcaja na svakoj lokaciji je 10 minuta te se računa vrijeme putovanja između svake lokacije:

1. Prvo rješenje - ista ruta za oba vozila:
    - Vozilo 1:
        - Polazak u grad: 08:00
        - M1: dolazak 08:10, iskrcaj 10 min (08:10 - 08:20)
        - M2: put 15 min, dolazak 08:35, iskrcaj 10 min (08:35 - 08:45)
        - M3: put 20 min, dolazak 09:05, iskrcaj 10 min (09:05 - 09:15)
        - Ukupno vrijeme: 08:00 - 09:15 (75 minuta)
    - Vozilo 2:
        - Polazak u grad: 08:00
        - M1: dolazak 08:20 (čekanje zbog Vozila 1), iskrcaj 10 min (08:20 - 08:30)
        - M2: put 15 min, dolazak 08:45 (zadržavanje zbog prometa), iskrcaj 10 min (08:45 - 08:55)
        - M3: put 20 min, dolazak 09:15, iskrcaj 10 min (09:15 - 09:25)

- Ukupno vrijeme: 08:00 - 09:25 (85 minuta)

U prvom rješenju, oba vozila prate istu rutu i dolaze na iste lokacije jedno za drugim. Zbog toga, Vozilo 2 mora čekati na svakoj lokaciji dok Vozilo 1 završi s iskrcajem, što produžuje ukupno vrijeme dostave u gradu.

2. Drugo rješenje - različite rute za oba vozila:
    - Vozilo 1:
        - Dolazak u grad: 08:00
        - M1: dolazak 08:10, iskrcaj 10 min (08:10 - 08:20)
        - M2: put 15 min, dolazak 08:35, iskrcaj 10 min (08:35 - 08:45)
        - M3: put 20 min, dolazak 09:05, iskrcaj 10 min (09:05 - 09:15)
        - Ukupno vrijeme: 08:00 - 09:15 (75 minuta)
    - Vozilo 2 (alternativna ruta):
        - Dolazak u grad: 08:00
        - M3: put 20 min, dolazak 08:20, iskrcaj 10 min (08:20 - 08:30)
        - M1: put 15 min, dolazak 08:45, iskrcaj 10 min (08:45 - 08:55)
        - M2: put 10 min, dolazak 09:05, iskrcaj 10 min (09:05 - 09:15)
        - Ukupno vrijeme: 08:00 - 09:15 (75 minuta)

U drugom rješenju, Vozilo 2 koristi alternativnu rutu, počevši s M3 umjesto M1. Ovim pristupom vozila izbjegavaju susrete na dostavnim lokacijama, što rezultira kraćim ukupnim vremenom dostave za oba vozila. Umjesto čekanja na svakoj lokaciji, oba vozila optimiziraju svoju rutu tako da minimiziraju moguće prometne gužve i zastoje.
Ovaj primjer jasno ilustrira problem optimizacije ruta u urbanom prometu. Korištenjem istih ruta za oba vozila povećava se vrijeme provedeno u gradu, dok promjena rute za jedno vozilo može značajno smanjiti ukupno vrijeme dostave. Osnovni cilj optimizacijskih algoritama je pronaći rješenja koja minimiziraju ukupno vrijeme i resurse potrebne za izvršenje dostave, uzimajući u obzir specifične uvjete u gradskoj jezgri, kao što su prometne gužve i ograničena parkirna mjesta.

2.4. ALGORITMI ZA OPTIMIZACIJU RUTA

Postoje različiti algoritmi koji se koriste za optimizaciju ruta u logistici. Oni se mogu podijeliti na klasične, heurističke, metaheurističke i algoritme strojnog učenja. Svaka od ovih kategorija pruža specifične prednosti i koristi se u različitim slučajevima, ovisno o tipu problema koji se rješava.

2.4.1. Klasični algoritmi

Klasični algoritmi, kao što su Dijkstrin algoritam i Bellman-Fordov algoritam, predstavljaju osnovne metode za rješavanje problema optimizacije u grafovima. Dijkstrin algoritam koristi se za pronalaženje najkraćeg puta u mreži, dok Bellman-Fordov algoritam omogućuje rješavanje problema s negativnim težinama u grafu. [3]

Dijkstrin algoritam (nazvan po autoru, E.W. Dijkstri) rješava problem pronalaženja najkraćeg puta od točke u grafu (izvora) do odredišta. Ispostavlja se da se najkraći putovi od danog izvora do svih točaka u grafu mogu pronaći u istom vremenu, pa se ovaj problem ponekad naziva problemom najkraćih putova s jednim izvorom. [4]

Bellman-Fordov algoritam je klasični algoritam za problem najkraćih puteva od jednog izvora. Algoritam radi u vremenu O(nm) na grafu sa n čvorova i m lukova. Ovo je trenutno najbolje poznato snažno polinomijalno ograničenje za ovaj problem. [5]

Ovi algoritmi pružaju osnovne alate za rješavanje problema optimizacije ruta, ali njihova učinkovitost opada s porastom složenosti i veličine problema. Iako su klasični algoritmi temelj mnogih suvremenih rješenja, njihova primjena je često ograničena zbog računalne složenosti i nedostatka fleksibilnosti u rješavanju složenijih problema, kao što su oni s velikim brojem varijabli i ograničenja.

2.4.2. Heuristički algoritmi

Heuristički algoritmi, poput algoritma k-najbližeg susjeda, nude brzo rješenje za probleme optimizacije ruta, ali ne garantiraju pronalazak globalno optimalnog rješenja. [6]

Ovi algoritmi su dizajnirani da pronađu prihvatljivo rješenje u razumnom vremenskom okviru, što ih čini idealnima za primjenu u stvarnim situacijama gdje je brzina važnija od optimalnosti. Algoritmi poput algoritma k-najbližeg susjeda i algoritma umetanja najbližeg i

najdaljeg primjenjuju jednostavna pravila za izgradnju ruta, što omogućuje brzo pronalaženje rješenja, ali uz rizik da konačno rješenje ne bude optimalno. Ovi algoritmi su korisni u situacijama kada je vrijeme za izračun ograničeno ili kada se radi o manje složenim problemima.

2.4.3. Metaheuristički algoritmi

Metaheuristički algoritmi, kao što su optimizacija rojem čestica (eng. *Particle Swarm Optimization* - PSO) i optimizacija krda afričkih bizona (eng. *African Buffalo Optimization* - ABO), omogućuju istraživanje velikih prostora rješenja i često nadmašuju klasične metode. [7]

Ovi algoritmi kombiniraju različite heurističke metode kako bi povećali vjerojatnost pronalaska globalno optimalnih rješenja. PSO koristi model ponašanja roja pčela ili jata ptica kako bi pronašao optimalno rješenje. ABO, ili optimizacija krda afričkih bizona, noviji je pristup koji se oslanja na ponašanje krda u pronalaženju optimalnih izvora hrane, primjenjujući te principe u optimizaciji kompleksnih problema. Ovi algoritmi posebno su korisni u situacijama kada se radi o velikim i složenim problemima gdje tradicionalne metode ne uspijevaju pronaći učinkovito rješenje. Njihova primjena u logistici omogućuje bolju optimizaciju ruta u uvjetima visokih zahtjeva i promjenjivih okolnosti. Postoji velik broj metaheurističkih metoda jer jedna metaheuristička metoda nije dobra za sve optimizacijske probleme pa se stoga stalno predlažu i istražuju novi algoritmi. U ovom radu evaluacija je također pokazala kako je jedan algoritam (ABO) dao najbolje ukupne rezultate, ali nije dao najbolji rezultat baš za svaki testni skup (32 testna skupa) nad kojim je izvršena evaluacija. Tu tvrdnju potvrđuje i teorem „Nema besplatnog ručka“ (eng. No Free Lunch - NFL) koji kaže da dobar rad određenog algoritma na jednom problemu očituje jednako lošim radom tog istog algoritma na nekom drugom problemu.

Odabrani algoritmi su ocijenjeni pomoću trideset i dva testna skupa, što predstavlja reprezentativnu veličinu uzorka. Evaluacija je također pokazala da, iako je jedan algoritam dao najbolje ukupne rezultate, to ne znači da je taj algoritam najbolji za rješavanje svakog skupa. Ovo također naglašava potrebu za daljnjim istraživanjima u području metaheurističkih algoritama. [8]

Intuitivno, ako algoritam dobro funkcionira na određenoj klasi problema, to nužno plaća s lošijom izvedbom na skupu svih preostalih problema. [9]

2.4.4. Algoritmi strojnog učenja

Algoritmi strojnog učenja koriste se za analizu i optimizaciju ruta na temelju povijesnih podataka i obrazaca, omogućujući prilagodbu promjenjivim uvjetima. [10]

Primjena strojnog učenja u optimizaciji ruta postaje sve popularnija, posebno s razvojem tehnologija velikih podataka i naprednih analitičkih alata. Nadzirano učenje koristi se za treniranje modela koji mogu predviđati optimalne rute na temelju povijesnih podataka, dok nenadzirano učenje omogućuje identifikaciju obrazaca i klasteriranje dostavnih točaka kako bi se poboljšala učinkovitost dostave. Strojno učenje pruža fleksibilnost i prilagodljivost koja je potrebna u dinamičkim okruženjima, omogućujući optimizaciju ruta u stvarnom vremenu te poboljšanje ukupne učinkovitosti logističkih operacija.

3. ALGORITMI ZA RJEŠAVANJE PROBLEMA OPTIMIZACIJE RUTE DOSTAVNIH VOZILA U GRADSKOM SREDIŠTU

U ovom poglavlju će se predstaviti 3 algoritma pomoću kojih će se pokušati riješiti problem optimizacije ruta za gradsko središte grada Rijeke. Prvo će se razmotriti jednostavan algoritam i sebičan algoritam te će se detaljnije razmotriti algoritam afričkog bizona (ABO), analizirati njegovi temeljni principi, te ispitati njegovu potencijalnu primjenu u optimizaciji ruta. Osim toga, istražiti će se prednosti i nedostatci ovog algoritma, te usporediti s drugim metaheurističkim pristupima u području optimizacije.

3.1. JEDNOSTAVNI ALGORITAM

Jednostavni (eng. *Basic*) algoritam služi kao polazište za optimizaciju dostavnih ruta, a koristi se za jednostavno izračunavanje ukupnog vremena potrebnog za dostavu na sve lokacije prema unaprijed zadanom redoslijedu. Njegova glavna funkcija je prolazak kroz sve definirane točke na ruti bez ikakvih optimizacija. Svako vozilo započinje dostavu iz početne točke, posjećuje sve određene lokacije redom i vraća se na početnu točku. Kod jednostavnog algoritma, za svaku dostavu izračunava se ukupno vrijeme potrebno za obavljanje dostave svih predviđenih pošiljki, pri čemu se uzima u obzir vrijeme dolaska vozila na svaku lokaciju, vrijeme čekanja na lokaciji te udaljenost između pojedinih točaka. Ovaj pristup omogućava jednostavnu implementaciju, ali ne vodi računa o optimizaciji rute, što znači da algoritam nije prilagođen za minimiziranje ukupnog vremena ili troškova dostave. Algoritam je pogodan za situacije gdje je potrebno brzo procijeniti vrijeme dostave bez obzira na eventualnu optimizaciju. Unatoč svojoj jednostavnosti, može biti koristan za inicijalno testiranje i postavljanje osnovnog okvira za složenije algoritme koji će kasnije pokušati optimizirati rutu na temelju različitih kriterija, poput smanjenja ukupne udaljenosti, vremena putovanja ili troškova.

3.2. POHLEPNI ALGORITAM

Pohlepni (eng. *Greedy*) algoritam predstavlja jednostavnu, ali značajno efikasniju metodu optimizacije u usporedbi s jednostavnim algoritmom. Ovaj algoritam djeluje prema principu

"pohlepe" gdje u svakom koraku odabire trenutno najbolju opciju bez obzira na dugoročne posljedice.

U kontekstu optimizacije ruta, pohlepni algoritam za neko dostavno vozilo prvo odabire najbližu lokaciju na koju treba izvršiti dostavu, zatim prelazi na sljedeću najbližu i tako redom, sve dok ne obiđe sve lokacije. Ključni korak u ovom algoritmu je generiranje svih mogućih kombinacija redoslijeda dostave za zadane lokacije. Algoritam zatim računa ukupno vrijeme dostave za svaku kombinaciju i odabire onu koja minimizira ukupno vrijeme. Iako ovaj pristup često pruža dobre rezultate, njegova glavna slabost je ta što ne uzima u obzir globalnu optimizaciju rute za sva vozila odjednom (nego samo za jedno po jedno vozilo) što može dovesti do lošijih rješenja. Na primjer, može se dogoditi da kombinacija više manjih koraka na kraju daje bolji ukupni rezultat nego odabir pojedinačno najboljih koraka. Unatoč tome, pohlepni algoritam je koristan zbog svoje brzine i jednostavnosti, osobito u slučajevima kada je broj lokacija manji i kad su rute manje kompleksne. Ovaj algoritam, kao i ostali razvijeni algoritmi u ovom radu, korišten je za međusobnu usporedbu kako bi se procijenila njihova učinkovitost u različitim scenarijima optimizacije ruta.

3.3. ALGORITAM AFRIČKOG BIZONA

Algoritam afričkog bizona inspiriran je kolektivnim ponašanjem afričkih bizona u njihovom prirodnom staništu. Ovaj algoritam koristi principe suradnje i grupne dinamike kako bi istražio prostor rješenja i pronašao optimalne ili približno optimalne odgovore na probleme optimizacije. [7]

Kao što ime kaže, ABO algoritam imitira ponašanje krda bizona, gdje svaka jedinka predstavlja potencijalno rješenje. Bizoni u krdu komuniciraju i međusobno dijele informacije o okruženju, što im omogućava donošenje zajedničkih odluka o smjeru kretanja prema povoljnijim izvorima hrane. S obzirom na to da su bizoni često suočeni s promjenjivim i nepredvidljivim uvjetima, ova kolektivna strategija omogućava brzu prilagodbu i efikasno pretraživanje novoga prostora za pronalazak najboljeg izvora hrane. Primjena ABO algoritma u kontekstu optimizacije ruta može donijeti inovativne pristupe rješavanju složenih problema u logistici, osobito onih koji uključuju dinamičke uvjete i vremensko opterećenje.

3.3.1. Princip rada algoritma na primjeru kretanja krda bizona

ABO algoritam zasnovan je na dva osnovna principa: istraživanju i eksploataciji. Tijekom faze istraživanja, bizoni (tj. rješenja) pretražuju prostor kako bi otkrili nove, potencijalno bolje izvore hrane. Faza eksploatacije, s druge strane, fokusira se na dovršavanje već pronađenih dobrih izvora kako bi se dodatno poboljšala njihova kvaliteta. [10]

Krdo bizona se nalazi na velikoj površini u potrazi za hranom, a ta faza se naziva faza istraživanja. Tijekom faze istraživanja pretražuje se što veći dio ukupne površine u potrazi za što boljim izvorom hrane te tijekom tog perioda svaki bizon može potencijalno naići na najbogatiji izvor hrane. Faza istraživanja prikazana je na slici 1.

Slika 1. Faza istraživanja

Izvor: izradio student

Nakon što završi faza istraživanja, tj. kada jedan od bizona nađe dobar izvor hrane na površini, cijelo krdo kreće sa eksploatacijom hrane oko mjesta gdje je pronađen, to se naziva faza eksploatacije te je prikazana na slici 2.

Slika 2. Faza eksploatacije

Izvor: izradio student

Također, tijekom procesa eksploatacije, moguće je da u blizini samog izvora neki od bizona pronađe još bogatiji izvor hrane te time taj novi bogatiji izvor postaje i novi najbolji izvor te se cijelo krdo premješta što bliže novome najboljemu izvoru.

Ovaj algoritam koristi dinamičku prilagodbu gdje bizoni koji se nalaze na boljim pozicijama preuzimaju ulogu vođe, dok ostali članovi krda prate njihovo kretanje. Kako bi izbjegli loša rješenja, uloge vođe mogu se mijenjati tijekom iteracija, što osigurava da algoritam ostaje fleksibilan i sposoban istražiti različite dijelove prostora rješenja.

Važno je napomenuti da ABO algoritam također uključuje mehanizme koji omogućavaju bizonima da se vrate na prethodne pozicije ukoliko novo rješenje ne pokazuje poboljšanje, čime se izbjegava gubljenje resursa na pretrage koje ne nude dobro rješenje.

3.3.2. Primjena algoritma u optimizaciji ruta

ABO algoritam ima značajan potencijal u primjeni za optimizaciju ruta, osobito u logističkim sustavima gdje su potrebna fleksibilna i dinamička rješenja. Njegova sposobnost prilagodbe na promjenjive uvjete i brze reakcije na promjene u stvarnom vremenu čine ga pogodnim za rješavanje složenih problema. [7]

Optimizacija ruta je problem koji zahtijeva dobru ravnotežu između globalnog istraživanja mreže i lokalnog poboljšanja ruta, može imati koristi od ABO algoritma jer on omogućuje istovremeno istraživanje i eksploataciju prostora rješenja. Primjerice, ABO može

biti korišten u scenarijima gdje su prisutni prometni zastoji, nepredviđene vremenske promjene ili iznenadne trenutne promjene slobodnih dostavljačkih mjesta. U takvim slučajevima, ABO algoritam mogao bi brzo reagirati na promjene i prilagoditi rute u skladu s novonastalim uvjetima. Međutim, da bi se u potpunosti ispitala njegova učinkovitost u stvarnim situacijama, potrebno je provesti opsežna istraživanja sa stvarnim podacima.

3.3.2.1. Teoretska implementacija algoritma afričkog bizona

Algoritam afričkog bizona je metaheuristički algoritam za optimizaciju dostavnih ruta kroz više iteracija pokušava pronaći bolje rješenje. Algoritam započinje generiranjem početne populacije rješenja. Svako rješenje u populaciji predstavlja moguću rutu za dostavu, koja se sastoji od nasumično odabranih redoslijeda posjeta različitim lokacijama. Algoritam prolazi kroz više generacija, pri čemu se svaka generacija sastoji od nekoliko rješenja (populacija). U svakoj generaciji, algoritam optimizira rješenja koristeći različite strategije. Svako rješenje se evaluira na temelju ukupnog vremena dostave. Ovo uključuje vrijeme putovanja između lokacija i vrijeme čekanja na svakoj lokaciji. Ako neko rješenje nije poboljšano u određenom broju generacija, algoritam ga prilagođava generiranjem novih rješenja nasumično, u nadi da će pronaći bolje rute. Postoje različite opcije za prilagodbu vrijednosti u rješenju, uključujući korištenje dinamičkih faktora za promjenu težine prilagodbe. Rješenja se ažuriraju na temelju njihove evaluacije. Ako novo rješenje ima bolje ukupno vrijeme od prethodnih rješenja, ono se zamjenjuje kao novo najbolje rješenje. Algoritam prati najbolja rješenja lokalno i globalno. Najbolje rješenje za svaku generaciju i najbolje rješenje ukupno se bilježe i ispravljaju ako se pronađe bolje rješenje. Nakon što su sve generacije završene, algoritam ispisuje najbolje rješenje za dostavne rute, uključujući redoslijed posjeta i ukupno vrijeme dostave. ABO algoritam koristi sofisticirane tehnike evolucijskog optimiziranja kako bi poboljšao rute dostave kroz iteracije, omogućujući pronalaženje učinkovitijih rješenja za kompleksne probleme optimizacije ruta.

3.3.3. Prednosti i nedostatci algoritma

Prednosti ABO algoritma uključuju njegovu fleksibilnost i sposobnost izbjegavanja loših rješenja, što ga čini pogodnim za složene probleme optimizacije u kojima tradicionalni algoritmi često ne uspijevaju. [7]

Prednosti:

- Fleksibilnost: ABO algoritam može biti prilagođen za razne izazove, što ga čini univerzalno primjenjivim.
- Učinkovitost u izbjegavanju loših rješenja: Dinamičko preuzimanje uloga vođe unutar krda omogućava algoritmu da istraži različite dijelove u prostoru rješenja, smanjujući rizik od opstanka na lošem rješenju.
- Brza prilagodba promjenjivim uvjetima: ABO može brzo reagirati na promjene u okruženju, čineći ga korisnim u dinamičkim i nepredvidljivim scenarijima.

Nedostaci:

- Visoki računalni zahtjevi: Kako bi postigao optimalne rezultate, ABO algoritam može zahtijevati znatne računalne resurse, što može biti izazov u situacijama s ograničenim resursima.
- Potrebna prilagođavanje parametara: Postizanje najboljih performansi često zahtijeva pažljivo podešavanje algoritamskih parametara, što može biti zahtjevno, osobito u radu sa stvarnim podacima.

Kroz daljnja istraživanja potrebno bi bilo provesti analizu performansi algoritma kako bi se ABO algoritam još dodatno mogao modificirati ili hibridizirati s drugim metodama de se uklone neki od njegovih nedostataka, te poboljšale performanse u određenim kontekstima.

3.3.4. Usporedba s ostalim metaheurističkim algoritmima

U usporedbi s ostalim popularnim metaheurističkim algoritmima, kao što je genetski algoritm (eng. *Genetic Algorithm* - GA), optimizacija rojem čestica i diferencijska evolucija (eng. *Differential evoluton* - DE), ABO algoritam nudi jedinstvenu kombinaciju istraživačkih i eksploatacijskih mogućnosti koje mogu biti posebno korisne u optimizaciji ruta. [10]
ABO algoritam nadmašuje GA u konvergenciji prema optimalnom rješenju, jer često zahtijeva manje iteracija zahvaljujući svojoj dinamičkoj prirodi. Nasuprot tome, PSO može biti učinkovit u ranim fazama pretrage, ali često pati od prerane konvergencije, zbog čega ABO ima prednost zahvaljujući svojim dinamičkim promjenama uloga unutar krda. U usporedbi s DE, ABO pokazuje bolju prilagodbu za diskretne probleme optimizacije, kao što su optimizacija ruta, dok je DE više usmjeren prema kontinuiranim problemima. Time se ABO predstavlja kao optimalno rješenje za kompleksne, diskretne optimizacijske izazove koji se često susreću u logistici i

transportu. Iako ABO algoritam nudi nekoliko ključnih prednosti u odnosu na druge metode, njegova stvarna primjenjivost ovisit će o kontekstu problema, dostupnim računalnim resursima i specifičnostima samih ruta. Daljnja istraživanja bit će ključna za poboljšanje algoritma kako bi ABO mogao biti učinkovitiji u različitim scenarijima optimizacije.

4. METODOLOGIJA

U ovom poglavlju će se detaljno obraditi metodološki pristup korišten u istraživanju. Poglavlje obuhvaća proces prikupljanja i odabira podataka te implementaciju tih algoritama u programskom jeziku Java. Također, bit će objašnjeni postupci validacije i testiranja algoritama. Odabrano programsko sučelje za implementaciju koda je Apache NetBeans IDE [11], koji je popularan alat za razvoj Java aplikacija zbog svoje fleksibilnosti i podrške za razne dodatke koji olakšavaju razvoj složenih algoritama.

4.1. IZBOR PODATAKA

Podaci korišteni u ovom istraživanju obuhvaćaju 21 lokaciju unutar užeg središta grada Rijeke, koje su odabrane kao ključne točke za dostavu. Lokacije su numerirane sa brojevima od 19 do 40. Ove lokacije na kojima se nalaze mjesta su u okolini različitih objekata, od poslovnih prostora do stambenih zgrada, što omogućava realističan prikaz izazova u optimizaciji ruta dostavnih vozila. Sve odabrane lokacije odabrane za potrebe ovog rada prikazane su na slici 3.

Slika 3. Odabrane lokacije

Izvor: Rijeka Plus

S obzirom na specifičnosti istraživanja, svaka je lokacija tretirana kao zasebno mjesto dostave, iako neke lokacije u stvarnosti sadrže više od jednog dostavnog mjesta.

4.1.1. Prikupljanje podataka o lokacijama

Podaci o lokacijama prikupljeni su korištenjem alata za geolokaciju, prvenstveno Google Maps, kako bi se osigurala preciznost informacija o svakoj dostavnoj točki. Svaka od 21 lokacije numerirana je prema identifikacijskim brojevima (od 19 do 40) kako bi se olakšala obrada podataka i njihovo povezivanje s vremenima putovanja između pojedinih točaka. Iako neke lokacije mogu imati više dostavnih mjesta u stvarnosti, u ovom modelu svaka je lokacija pojednostavljena da ima jedno mjesto dostave. Ovaj pristup omogućava veći fokus na optimizaciju ruta između lokacija dostave, bez dodatnih komplikacija i zahtjeva na model koje bi se morale uvesti da ih je više. Također, model je napravljen na način da nudi mogućnost kasnijeg proširivanja koda da uključuje preostala mjesta na lokacijama, tj. skalabilan je.

4.1.2. Mjerenje vremena putovanja

Kako bi se osigurao realan prikaz vremena putovanja između odabranih lokacija, korišteni su podaci prikupljeni kroz Google Maps API. Za svaku od 21 lokacije, izračunata su vremena putovanja do svih ostalih lokacija, uključujući sve moguće kombinacije ruta između njih. Mjerenje vremena putovanja obavljeno je u periodu od dva tjedna na radne dane između 7:30 i 9:00 sati ujutro, što predstavlja razdoblje jutarnje prometne gužve. Ovo vremensko razdoblje odabrano je s ciljem da se što točnije odrazi stvarna situacija na cestama u Rijeci, jer su prometne gužve u tom periodu najveće. Takav pristup omogućava da rezultati optimizacije ruta budu što bliži stvarnim uvjetima u gradu.

4.2. IMPLEMENTACIJA U JAVI

Ovaj dio detaljno će se baviti implementacijom algoritama i obradom podataka u programskom jeziku Java. Cilj je opisati kako su algoritmi razvijeni, kako funkcioniraju u praksi, te kako su podaci o lokacijama i vremenima putovanja integrirani u sustav. Kroz

konkretne primjere koda, prikazat će se ključne aspekte implementacije te objasniti logika iza svake metode i klase.
Ovaj dio podijeljen je na četiri ključna segmenta:

- Implementacija učitavanja podataka
- Implementacija jednostavnog algoritma
- Implementacija pohlepnog algoritma
- Implementacija ABO algoritma

Kroz svaku od ovih sekcija, uz prikaz koda, osvrnut će se na ključne funkcije i metode, te objasniti kako su optimizacije postignute unutar svakog algoritma. Cilj je omogućiti dublje razumijevanje implementacije i funkcionalnosti svakog dijela sustava.

4.2.1. Implementacija učitavanja podataka

Ovdje će se detaljno objasniti kako se podaci o lokacijama, udaljenostima, i dolascima učitavaju i obrađuju u Javi. U ovom dijelu koda prikazano je učitavanje podataka iz Excel tablice i tekstualne datoteke. Ovi podaci su ključni za kasniju optimizaciju dostavnih ruta, pa je važno razumjeti kako se pravilno pohranjuju i organiziraju unutar programa.

4.2.1.1. Učitavanje Excel tablice

Prvi korak u obradi podataka je učitavanje Excel tablice koja sadrži informacije o udaljenostima između različitih lokacija. Za to koristimo klasu *FileInputStream* koja omogućava čitanje podataka iz Excel datoteke te je prikazan u ovom isječku koda:

```
FileInputStream fins = new FileInputStream(new File("Lokacije za
diplomski.xlsx"));
 XSSFWorkbook wb = new XSSFWorkbook(fins);
 XSSFSheet sheet = wb.getSheetAt(0);
```

Klasa *FileInputStream* omogućuje otvaranje datoteke "Lokacije za diplomski.xlsx". Importirana programska biblioteka *XSSFWorkbook* predstavlja cijelu Excel datoteku, dok *XSSFSheet* predstavlja jedan specifični radni list unutar datoteke.

Nakon što se Excel tablica učita, koristi se programska biblioteka *FormulaEvaluator* za evaluaciju ćelija koje mogu sadržavati formule, radi sigurnosti da se uvijek dobivaju ispravne vrijednosti:

```
FormulaEvaluator fmEval = wb.getCreationHelper().createFormulaEvaluator();
```

4.2.1.2. Popunjavanje polja udaljenosti

Podaci o udaljenostima između lokacija pohranjuju se u dvodimenzionalno polje *udaljenost* koje sadrži 41x41 elemenata. Elementi polja predstavljaju udaljenosti između lokacija numeriranih od 19 do 40:

```
int[][] udaljenost = new int[41][41];
```

Popunjavanje polja udaljenostima iz Excel tablice zahtijeva iteraciju kroz redove i stupce kako bi se pronašle odgovarajuće vrijednosti, u nastavku je opisan princip rada i pridodan kod:

1. Pronalaženje redova za lokaciju: Pretražujemo redove u Excel tablici kako bi se našla odgovarajuća lokacija *i*. Kad se pronađe vrijednost, pohranjuje se indeks u varijablu *indexP*.
2. Pronalaženje stupaca za lokaciju: Slično pretražuju se stupci kako bi se našla lokacija *j*, pohranjujući indeks u varijablu *indexD*.
3. Postavljanje vrijednosti u polje: Ako su obje lokacije pronađene, tada se izračunava udaljenost između njih i pohranjuje se u polje *udaljenost*.

```
if (i == j) {
    udaljenost[i][j] = 0;
} else if (indexP != -1 && indexD != -1) {
    Cell cell = sheet.getRow(indexD).getCell(indexP);
      if (cell != null && fmEval.evaluateInCell(cell).getCellType() ==
CellType.NUMERIC) {
       udaljenost[j][i] = (int) cell.getNumericCellValue();
    }
}
```

4.2.1.3. Stvaranje mape mjesta

Nakon što su udaljenosti definirane, u daljnjem kodu je prikazano stvaranje mape *mapMjesta* koja povezuje svaku lokaciju s inicijalnim vrijednostima za dostavna mjesta. Svaka lokacija ima pridružen *ArrayList* od 1000 elemenata u kojemu svaki element predstavlja jednu minutu te može imati vrijednost 0 ili 1, gdje 0 predstavlja da je taj vremenski interval (1 min) slobodan, a 1 da je zauzet . Na početku su svih 1000 elemenata inicijalizirani na 0:

```
Map<Integer, ArrayList<Integer>> mapMjesta = new HashMap<>();
for (int i = 19; i <= 40; i++) {
    ArrayList<Integer> a = new ArrayList<>();
    for (int j = 0; j < 1000; j++) {
        a.add(0);
    } mapMjesta.put(i, a);
}
```

4.2.1.4. Učitavanje podataka iz tekstualne datoteke

Podaci o dolascima učitavaju se iz tekstualne datoteke "dostava.txt". U sljedećem isječku koda prikazana je funkcija *ucitajDolaske* koji čita svaku liniju datoteke, dijeli je na dijelove prema zarezu i dodaje u *ArrayList*:

```
public static ArrayList<ArrayList<Integer>> ucitajDolaske(File file) throws
IOException {
        ArrayList<ArrayList<Integer>> dolasci = new ArrayList<>();
        try (BufferedReader br = new BufferedReader(new FileReader(file)))
    {
            String linija;
            while ((linija = br.readLine()) != null) {
                String[] dijelovi = linija.split(",");
                ArrayList<Integer> dolazak = new ArrayList<>();
                for (int i = 0; i < dijelovi.length; i++) {
                    dolazak.add(Integer.valueOf(dijelovi[i]));
                }
                dolasci.add(dolazak);
            }
        }
```

```
        return dolasci;
    }
```

Programska biblioteka *BufferedReader* koristi se za čitanje datoteke red po red. Funkcija *split()* dijeli redak datoteke na elemente prema zarezu. *ArrayList* dodaje svaki redak u listu *dolasci*, gdje svaki element predstavlja dolazak na određenu lokaciju.

Sveukupno gledano, ova implementacija omogućava učinkovito učitavanje i organizaciju podataka potrebnih za optimizaciju dostavnih ruta. Podaci o udaljenostima i dolascima pohranjuju se u odgovarajuće strukture podataka, koje će kasnije koristiti algoritmi za optimizaciju.

4.2.2. Implementacija jednostavnog algoritma

U ovom dijelu će se detaljno objasniti kako je implementiran jednostavan algoritam za optimizaciju dostavnih ruta. Ovaj algoritam predstavlja jednostavan pristup optimizaciji, fokusirajući se na izračun ukupnog vremena dostave za svako vozilo na temelju zadanih lokacija i vremena dolaska. Ključne funkcionalnosti uključuju učitavanje podataka, računanje vremena puta, i označavanje iskrcavanja.

4.2.2.1. Računanje ukupnog vremena puta

Nakon učitavanja podataka o dolascima iz tekstualne datoteke, u daljnjem isječku koda, algoritam prolazi kroz svaku rutu, računajući ukupno vrijeme dostave za svako vozilo. Ovaj proces se odvija unutar glavne petlje koja iterira kroz sve dolaske vozila:

```
    for (ArrayList<Integer> dolazak : dolasci) {
        int voziloBroj = dolazak.get(0);
        int voziloDolazak = dolazak.get(1);
        dolazak.remove(0);
        dolazak.remove(0);

int ukupnoVrijeme = RacunajUkupnoVrijeme(dolazak, udaljenost, mapMjesta,
voziloDolazak);
```

```
System.out.println("Ukupno vrijeme dostave za vozilo " + voziloBroj + ": " +
ukupnoVrijeme + " min");

        OznaciIskrcavanje(dolazak, udaljenost, mapMjesta, voziloDolazak);
        ukupnoVrijemeSvi += ukupnoVrijeme;
    }
```

Funkcija *RacunajUkupnoVrijeme* računa vrijeme potrebno za izvršenje dostave za zadanu rutu. Nakon što se izračuna vrijeme dostave, funkcija *OznaciIskrcavanje* ažurira mapu *mapMjesta* s informacijama o vremenu iskrcavanja za svaku lokaciju.

4.2.2.2. Funkcija za računanje ukupnog vremena dostave

Funkcija *RacunajUkupnoVrijeme* je ključna za izračun ukupnog vremena čekanja i dostave za svaku rutu. Ona prolazi kroz sve lokacije koje vozilo treba posjetiti, računajući vrijeme dolaska i čekanja, u ovome isječku koda je rad funkcije u cijelosti prikazan:

```
public static int RacunajUkupnoVrijeme(ArrayList<Integer> svaMjesta, int[][]
udaljenost, Map<Integer, ArrayList<Integer>> mapMjesta, int dolazakPocetak)
{
        int UkupnoVrijemeCekanja = 0;
        int dolazakSljedeci;

        for (int i = 0; i < svaMjesta.size(); i++) {
            int mjesto = svaMjesta.get(i);
            if (i == 0) {
                dolazakSljedeci = dolazakPocetak;
            } else {
                 dolazakSljedeci = dolazakPocetak + UkupnoVrijemeCekanja +
           udaljenost[svaMjesta.get(i - 1)][svaMjesta.get(i)];
UkupnoVrijemeCekanja += udaljenost[svaMjesta.get(i - 1)][svaMjesta.get(i)];
            }
           int vrijemeCekanja = findVrijemeCekanja(mapMjesta.get(mjesto),
           dolazakSljedeci);
           UkupnoVrijemeCekanja += vrijemeCekanja + 10;
        }
        return UkupnoVrijemeCekanja;
    }
```

Varijabla *dolazakSljedeci* označava vrijeme dolaska na trenutnu lokaciju, koje se računa na temelju prethodnog dolaska i udaljenosti. Funkcija *findVrijemeCekanja* traži slobodni vremenski interval na lokaciji, gdje vozilo može izvršiti dostavu. Varijabla *vrijemeCekanja* dodaje se na ukupno vrijeme čekanja zajedno s fiksnim vremenom iskrcavanja te za potrebe ovog modela uzeto je 10 minuta.

4.2.2.3. Ažuriranje mape s oznakama iskrcavanja

Nakon izračuna vremena dostave, funkcija *OznaciIskrcavanje* ažurira mapu *mapMjesta*, koja prati kada je svaka lokacija zauzeta, funkcija je u cijelosti prikazana ovom isječku koda:

```
public static void OznaciIskrcavanje(ArrayList<Integer> svaMjesta, int[][]
udaljenost, Map<Integer, ArrayList<Integer>> mapMjesta, int dolazakPocetak)
{
        int UkupnoVrijemeCekanja = 0;
        int dolazakSljedeci;

        for (int i = 0; i < svaMjesta.size(); i++) {
            int mjesto = svaMjesta.get(i);
            if (i == 0) {
                dolazakSljedeci = dolazakPocetak;
            } else {
           dolazakSljedeci = dolazakPocetak + UkupnoVrijemeCekanja +
           udaljenost[svaMjesta.get(i - 1)][svaMjesta.get(i)];
UkupnoVrijemeCekanja += udaljenost[svaMjesta.get(i - 1)][svaMjesta.get(i)];
            }
           int vrijemeCekanja = findVrijemeCekanja(mapMjesta.get(mjesto),
      dolazakSljedeci);
            UkupnoVrijemeCekanja += vrijemeCekanja + 10;

          for (int j = dolazakSljedeci + vrijemeCekanja; j < dolazakSljedeci
     + vrijemeCekanja + 10; j++) {
            mapMjesta.get(mjesto).set(j, 1);
            }
        }
    }
```

Mapa *mapMjesta* prati zauzetost svake lokacije tijekom dana. Vrijeme dostave označeno je kao zauzeto kako bi se izbjegli konflikti s drugim vozilima. Varijabla *UkupnoVrijemeCekanja* prati ukupno vrijeme čekanja tokom putovanja (uključujući i vrijeme provedeno na lokacijama).

4.2.2.4. Pronalazak slobodnog vremena za iskrcavanje

Funkcija *findVrijemeCekanja* traži slobodan vremenski interval za iskrcavanje na određenoj lokaciji, ova funkcija uzima u obzir listu dostupnih vremena za tu lokaciju i vrijeme dolaska na lokaciju, a kaže koliko dugo treba čekati prije nego što se može izvršiti iskrcavanje. Funkcija je prikazana u cijelosti u sljedećem isječku koda:

```
public static int findVrijemeCekanja(ArrayList<Integer> polje, int dolazak)
{
        int vrijemeCekanja = 0;
        int pocetakDostave = -1;

        for (int i = dolazak; i <= polje.size() - 10; i++) {
            boolean found = true;
            for (int j = i; j < i + 10; j++) {
                if (polje.get(j) != 0) {
                    found = false;
                    break;
                }
            }
            if (found) {
                pocetakDostave = i;
                break;
            }
        }

        if (pocetakDostave != -1) {
            vrijemeCekanja = pocetakDostave - dolazak;
        }
        return vrijemeCekanja;
    }
```

Varijabla *pocetakDostave* pronalazi prvi slobodan interval od 10 minuta, dok varijabla *vrijemeCekanja* vraća razliku između željenog dolaska i stvarnog vremena početka dostave.

U suštini, jednostavni algoritam predstavlja osnovni pristup optimizaciji dostava, gdje se svako vozilo obrađuje pojedinačno, a vrijeme dolaska i čekanja izračunava se na temelju redoslijeda lokacija i njihove zauzetosti. Ovaj algoritam je jednostavno izvediv te pruža osnovu za daljnje optimizacije koje se mogu ostvariti u složenijim algoritmima.

4.2.3. Implementacija sebičnog algoritma

Sebičan algoritam funkcionira tako da u svakom koraku bira lokalno optimalno rješenje s ciljem minimiziranja ukupnog vremena dostave. U ovom dijelu će biti detaljno objašnjena implementacija ovog algoritma, uz prikaz relevantnog koda i objašnjenje ključnih pojmova.

4.2.3.1. Priprema podataka i generiranje kombinacija

Generiraju se sve moguće kombinacije redoslijeda obilaska dostavnih mjesta koje vozilo može napraviti. U sjedećem isječku koda kombinacije su generirane pomoću funkcije *generateCombinations*, a potom su uklonjene sve kombinacije koje sadrže duplikate koristeći funkciju *removeCombinationsWithDuplicates*:

```
List<int[]> combinations = generateCombinations(new int[svaMjesta.size()], 0, svaMjesta.size());
    removeCombinationsWithDuplicates(combinations);
```

4.2.3.2. Kreiranje i sortiranje rješenja

Za svaku kombinaciju kreira se *HashMap* koja povezuje mjesto s redoslijedom. Zatim se ova mapa sortira po vrijednostima, čime dobijemo redoslijed mjesta koji ćemo koristiti za izračunavanje vremena dostave. Ovo je prikazano u sljedećem isječku koda:

```
HashMap<Integer, Integer> hm = new HashMap<>();
for (int i = 0; i < svaMjesta.size(); i++) {
```

```
int mjesto = svaMjesta.get(i);
hm.put(svaMjesta.get(i), combination[i]);
}
      List<Map.Entry<Integer,     Integer>>     entryList     =     new
ArrayList<>(hm.entrySet());
entryList.sort(Map.Entry.comparingByValue());

LinkedHashMap<Integer, Integer> sortedMap = new LinkedHashMap<>();
for (Map.Entry<Integer, Integer> entry : entryList) {
sortedMap.put(entry.getKey(), entry.getValue());
}
```

Mapa *HashMap<Integer, Integer> hm* povezuje mjesto dostave sa slučajnim brojem iz trenutne kombinacije. Funkcija *entryList.sort(Map.Entry.comparingByValue())* sortira mapu prema vrijednostima, što rezultira redoslijedom obilaska mjesta dostave. Na kraju *LinkedHashMap<Integer, Integer>* sortedMap je sortirana mapa spremna za uporabu.

4.2.3.3. Računanje ukupnog vremena za svaku kombinaciju

Nakon što se dobije sortirani redoslijed, koristi se funkcija *RacunajUkupnoVrijeme* koja izračunava ukupno vrijeme dostave za trenutni redoslijed. Ova funkcija, prikazana u nadolazećem isječku koda, uzima u obzir udaljenosti između mjesta i vrijeme čekanja na slobodna mjesta za dostavu:

```
int ukupnoVrijeme = RacunajUkupnoVrijeme(svaMjestaLokalno, udaljenost,
mapMjesta, dolazakPocetak);
```

Funkcija *RacunajUkupnoVrijeme* izračunava ukupno vrijeme dostave uzimajući u obzir sve relevantne faktore (udaljenost, čekanje).

4.2.3.4. Pronalaženje najboljeg rješenja

Ako trenutno vrijeme dostave (za trenutnu kombinaciju) bude bolje od dosadašnjeg najboljeg vremena, tada se ažuriraju varijable *najOptimalnijeVrijeme* i *najOptimalnijiPut* kako bi zabilježili ovo rješenje. Ovo je prikazano u sljedećem isječku koda:

```
if (ukupnoVrijeme < najOptimalnijeVrijeme) {
    najOptimalnijeVrijeme = ukupnoVrijeme;
    najOptimalnijiPut = new ArrayList<>(svaMjestaLokalno);
}
```

Varijabla *najOptimalnijeVrijeme* pohranjuje najmanje vrijeme dostave koje je pronađeno do sada, dok *najOptimalnijiPut* pohranjuje redoslijed mjesta koji daje najoptimalnije vrijeme dostave.

4.2.3.5. Ažuriranje stanja dostave

Nakon što se nađe optimalni redoslijed obilaska, potrebno je ažurirati stanje dostave koji je prikazan u sljedeće isječku koda. Ovaj korak uključuje označavanje iskrcavanja robe na mjestima dostave pomoću funkcije *OznaciIskrcavanje*.

```
OznaciIskrcavanje(svaMjestaOznacavanje, udaljenost, mapMjesta,
dolazakPocetak);
```

Funkcija *OznaciIskrcavanje* ažurira stanje dostave, uključujući slobodne intervale na svakom mjestu.

4.2.3.6. Konačni ispis rezultata

Na kraju te na zadnjem prikazanom isječku koda, algoritam ispisuje najoptimalniji redoslijed obilaska i najoptimalnije vrijeme dostave za svako vozilo, kao i ukupno optimalno vrijeme dostave za sva vozila.

```
System.out.println("Najoptimalniji redoslijed obilaska za vozilo " +
voziloBroj + ": " + najOptimalnijiPut);
System.out.println("Najoptimalnije ukupno vrijeme dostave za vozilo " +
voziloBroj + ": " + najOptimalnijeVrijeme + " min");
System.out.println("Ukupno optimalno vrijeme dostave za sva vozila: " +
ukupnoVrijemeSvi + " min");
```

Varijabla *najOptimalnijiPut* prikazuje najefikasniji redoslijed mjesta koji vozilo treba posjetiti, *najOptimalnijeVrijeme* prikazuje najmanje vrijeme potrebno za dostavu, *ukupnoVrijemeSvi* prikazuje sumu ukupnog vremena svih vozila za kompletnu dostavu.

Prema dizajnu, sebičan algoritam koristi jednostavan pristup optimizaciji, gdje se u svakom trenutku donosi najbolja lokalna odluka. To ga čini brzim i jednostavnim za implementaciju, ali može biti manje efikasan u složenijim slučajevima gdje bi globalna optimizacija zahtijevala drugačiji pristup.

4.2.4. Implementacija algoritma afričkog bizona

Algoritam afričkog bizona (ABO algoritam) je optimizacijski algoritam inspiriran ponašanjem bizona u krdu. Ovaj algoritam koristi kolektivno ponašanje jedinki unutar grupe kako bi pronašao optimalno rješenje za složene probleme. U nastavku će se opisati ključne komponente ABO algoritma, kako ga primijeniti u rješavanju problema optimizacije, te pružiti kôd za implementaciju.

4.2.4.1. Inicijalizacija parametara

U fazi pripreme postavljaju se potrebni parametri za izvedbu algoritma. Parametri kao što su broj početnih rješenja, broj iteracija koje algoritam te parametri za prilagodbu rješenja prikazani su u sljedećem isječku koda:

```
int populacija = 3000;
int numberOfGeneration = 10000;
double ld0 = 1.2;
double ld1 = 0.3;
double ld2 = 0.5;
int nepromijenjenjeGeneracije = 10;
```

Varijabla *populacija* definira koliko nasumičnih rješenja se generira na početku, *numberOfGeneration* indicira ukupan broj iteracija kroz koje algoritam prolazi. Koeficijenti za težinske parametre pod varijablama *ld0, ld1,* i *ld2* određuju utjecaj lokalno i globalno najboljih

rješenja na trenutna rješenja. Broj generacija kroz koje se najbolje rješenje nije mijenjalo je određeno varijablom *nepromijenjenjeGeneracije*.

4.2.4.2. Generiranje nasumičnih rješenja

U sljedećem isječku kod generira početnu populaciju nasumičnih rješenja koja se pohranjuje u strukturirane liste, ovdje su definirane glavne strukture podataka:

```
    Random random = new Random();
ArrayList<ArrayList<ArrayList<Integer>>> viseRjesenja = new ArrayList<>();
ArrayList<ArrayList<ArrayList<Integer>>> viseRjesenjaLokalnoNajbolje = new
ArrayList<>();
          ArrayList<Integer>      najboljeVrijemePopulacije      =      new
    ArrayList<>(Collections.nCopies(populacija, Integer.MAX_VALUE));
          ArrayList<Integer>   brojNepromijenjenihBoljihRjesenja   =   new
    ArrayList<>(Collections.nCopies(populacija, Integer.MAX_VALUE));
```

Lista *viseRjesenja* sadrži sva nasumična rješenja za sve kamione u svim dolascima, *viseRjesenjaLokalnoNajbolje* sadrži trenutno najbolje rješenje za svaki kamion u svakoj iteraciji. Varijabla *najboljeVrijemePopulacije* pohranjuje najbolje ukupno vrijeme dostave za svako rješenje u populaciji. Kako bi se izbjegao lokalni minimum, dodaje se popis *brojNepromijenjenihBoljihRjesenja* koji prati koliko dugo određeno rješenje nije poboljšano, što se koristi za resetiranje rješenja nakon određenog broja neuspjeha. Ova modifikacija omogućuje prevenciju preoptimizacije oko istog najboljeg rješenja na način da tjera „krdo bizona“ da počne negdje drugdje u prostoru rješenja tražiti moguća bolja rješenja.

U ovom sljedećem djelu koda svaki kamion dobije nasumične brojeve (od 1 do 30) koji određuju redoslijed obilaska lokacija, na temelju nasumično dobivenog broja određuje se poredak lokacija koju kamion obilazi. Lokacije se slažu od nižeg nasumično dobivenog broja prema višem:

```
    for (int p = 0; p < populacija; p++) {
        ArrayList<ArrayList<Integer>>    randomBrojeviZaKamione    =    new
ArrayList<>();
        for (int i = 0; i < dolasci.size(); i++) {
```

```
            ArrayList<Integer> dolazak = dolasci.get(i);
            int brojLokacija = dolazak.size() - 2;
            ArrayList<Integer> randomBrojevi = new ArrayList<>();
            for (int j = 2; j < brojLokacija + 2; j++) {
                randomBrojevi.add(random.nextInt(30) + 1);
            }
            randomBrojeviZaKamione.add(randomBrojevi);
        }
        viseRjesenja.add(randomBrojeviZaKamione);

    ArrayList<ArrayList<Integer>>  randomBrojeviZaKamioneLokalnoNajbolje  =
new ArrayList<>();
        for (ArrayList<Integer> kamion : randomBrojeviZaKamione) {
            ArrayList<Integer>      kamionLokalnoNajbolje      =      new
ArrayList<>(Collections.nCopies(kamion.size(), 0));

randomBrojeviZaKamioneLokalnoNajbolje.add(kamionLokalnoNajbolje);
        }

viseRjesenjaLokalnoNajbolje.add(randomBrojeviZaKamioneLokalnoNajbolje);
    }
```

RrandomBrojeviZaKamione pohranjuje nasumične brojeve za svaku lokaciju posjećenu od strane kamiona, *viseRjesenja* sadrži nasumične brojeve za svaki kamion za sve rješenja u populaciji. Prazne liste koje će kasnije sadržavati najbolje lokalno rješenje za svaki kamion inicijalizira *randomBrojeviZaKamioneLokalnoNajbolje*. Za početak, varijabla *viseRjesenjaLokalnoNajbolje* postavlja lokalno najbolja rješenja na nule za svako rješenje. Ovo će se ažurirati kroz iteracije.

4.2.4.3. Iterativni pristup

U ovoj fazi, bizoni (rješenja) se kreću prema najboljem rješenju u krdu. Parametri *ld1* i *ld2* kontroliraju koliko se bizon kreće prema lokalnom i globalnom najboljem rješenju. Algoritam traži rješenja kroz određeni broj iteracija te je prikazan u sljedećem isječku koda:

```
    for (int nog = 1; nog < numberOfGeneration; nog++) {
        System.out.println("GENERACIJA " + nog);
```

```
        if (nog > 1) {
            for (int p = 0; p < populacija; p++) {
if    (brojNepromijenjenihBoljihRjesenja.get(p) < nepromijenjenjeGeneracije)
{
     ArrayList<ArrayList<Integer>> trenutnaRjesenja = viseRjesenja.get(p);
     ArrayList<ArrayList<Integer>>lokalnoNajboljaRjesenja=viseRjesenjaLokal
     noNajbolje.get(p);
      ArrayList<Integer>globalnoNajboljaRjesenjaKamion                    =
     najboljeRjesenje.get(i);
```

Ova sljedeća petlja upravlja glavnim iterativnim procesom. U svakoj generaciji algoritam provjerava hoće li koristiti trenutna rješenja za mutacije ili resetirati rješenja ako su se predugo zadržavala bez poboljšanja. U drugoj i sljedećim generacijama (*'nog > 1'*), algoritam ažurira rješenja i koristi dinamičke ili statičke koeficijente za promjenu rješenja:

```
     for (int i = 0; i < trenutnaRjesenja.size(); i++) {
       ArrayList<Integer> trenutnaRjesenjaKamion = trenutnaRjesenja.get(i);
       ArrayList<Integer>          lokalnoNajboljaRjesenjaKamion          =
lokalnoNajboljaRjesenja.get(i);
      ArrayList<Integer>          globalnoNajboljaRjesenjaKamion          =
najboljeRjesenje.get(i);

     for (int j = 0; j < trenutnaRjesenjaKamion.size(); j++) {
       int trenutnaVrijednost = trenutnaRjesenjaKamion.get(j);
       int lokalnoNajboljaVrijednost = lokalnoNajboljaRjesenjaKamion.get(j);
     int globalnoNajboljaVrijednost = globalnoNajboljaRjesenjaKamion.get(j);

             ld1 = random.nextInt(100) / (double) 100;
             ld2 = 1 - ld1;
     int  novaVrijednost  =  (int)  ((trenutnaVrijednost  +  (int)  (ld1  *
lokalnoNajboljaVrijednost) + (int) (ld2 * globalnoNajboljaVrijednost)) * 0.5)
% 30 + 1;
            trenutnaRjesenjaKamion.set(j, novaVrijednost);
                      }
                   }
          } else {
ArrayList<ArrayList<Integer>> trenutnaRjesenja = viseRjesenja.get(p);
            for (int i = 0; i < trenutnaRjesenja.size(); i++) {
            ArrayList<Integer> randomBrojevi = new ArrayList<>();
            for (int j = 2; j < dolasci.size() - 2; j++) {
```

```
                randomBrojevi.add(random.nextInt(30) + 1);
                          }
                trenutnaRjesenja.set(i, randomBrojevi);
                      }
                   brojNepromijenjenihBoljihRjesenja.set(p, 0);
                  }
              }
          }
     }
```

Svaka iteracija predstavlja novi skup rješenja unutar kojeg algoritam pokušava pronaći bolje rješenje od prethodno pronađenog najboljeg rješenja. Ako se rješenje nije poboljšalo kroz određeni broj generacija (definirano varijablom *nepromijenjenjeGeneracije*), nasumično se generiraju nova rješenja za taj set. Dinamički se prilagođavaju *ld1* i *ld2* kako bi omogućili fleksibilniju optimizaciju. Nova vrijednost za određenu poziciju u rješenju je kombinacija trenutne vrijednosti, lokalno najboljih vrijednosti i globalno najboljih vrijednosti. Ako rješenje nije poboljšano u zadanim iteracijama, stvara se potpuno nova kombinacija nasumičnih brojeva za to rješenje.

4.2.4.4. Računanje vremena dostave

Funkcijom *RacunajUkupnoVrijeme* računa ukupno vrijeme putovanja kamiona između lokacija uzimajući u obzir udaljenosti između mjesta i vrijeme čekanja na svakoj lokaciji. Implementacija funkcije je prikazana u sljedećem isječku koda:

```
int ukupnoVrijemeSviNovo = 0;
for (int i = 0; i < randomBrojeviZaKamione.size(); i++) {
      ArrayList<Integer>            randomBrojeviZaJedanKamion            =
      randomBrojeviZaKamione.get(i);
    ArrayList<Integer> dolazak = dolasci.get(i);
    int voziloBroj = dolazak.get(0);
    int voziloDolazak = dolazak.get(1);
    ArrayList<Integer> svaMjesta = new ArrayList<>();
    for (int j = 2; j < dolazak.size(); j++) {
        svaMjesta.add(dolazak.get(j));
    }
    HashMap<Integer, Integer> hm = new HashMap<>();
```

```
    for (int j = 0; j < svaMjesta.size(); j++) {
        int mjesto = svaMjesta.get(j);
        int randomBroj = randomBrojeviZaJedanKamion.get(j);
        hm.put(randomBroj, mjesto);
    }

    ArrayList<Integer> zamjena = new ArrayList<>();
    TreeMap<Integer, Integer> sorted = new TreeMap<>(hm);
    for (Map.Entry<Integer, Integer> entry : sorted.entrySet()) {
        zamjena.add(entry.getValue());
    }

      int ukupnoVrijeme = RacunajUkupnoVrijeme(svaMjesta, udaljenost,
      mapPuta, voziloDolazak);
    ukupnoVrijemeSviNovo += ukupnoVrijeme;
}
```

Funkcija *RacunajUkupnoVrijeme* računa ukupno vrijeme dostave za sva mjesta koje kamion posjećuje, a *HashMap* i *TreeMap* koriste se za sortiranje lokacija prema nasumičnim brojevima, simulirajući nasumičan redoslijed dostave.

4.2.4.5. Traženje najboljeg rješenja

U zadnjem prikazanom danom isječku koda, ako novo generirano rješenje ima bolje ukupno vrijeme dostave od prethodnih, ažurira se najbolje vrijeme za to rješenje, a broj iteracija bez promjene resetira se na 0. Ako je ovo rješenje globalno najbolje do sada, ono se pohranjuje kao najbolje rješenje:

```
    if (ukupnoVrijemeSviNovo < najboljeVrijemePopulacije.get(p)) {
        brojNepromijenjenihBoljihRjesenja.set(p, 0);
        najboljeVrijemePopulacije.set(p, ukupnoVrijemeSviNovo);
        viseRjesenjaLokalnoNajbolje.set(p, new ArrayList<>(zamjena));
    } else {
    brojNepromijenjenihBoljihRjesenja.set(p,(brojNepromijenjenihBoljihRjes
enja.get(p) + 1));
    }
    if (ukupnoVrijemeSviNovo < najmanjeVrijeme) {
        najmanjeVrijeme = ukupnoVrijemeSviNovo;
```

```
            najboljeRjesenje = zamjena;
        }
```

Izraz *ukupnoVrijemeSviNovo* < *najboljeVrijemePopulacije.get(p)* uvjetuje da novo rješenje ima kraće vrijeme dostave, te ažurira se kao najbolje rješenje, a varijabla *brojNepromijenjenihBoljihRjesenja* uvjetuje da ako rješenje nije poboljšano, povećava se brojač neuspjelih iteracija. Nakon dovoljnog broja neuspjeha, rješenje se ponovno generira nasumično.

ABO algoritam se pokazuje učinkovitim u optimizaciji problema raspodjele resursa i vremena, gdje je potreban kompromis između više faktora (npr. udaljenosti i vremena čekanja). Ova metoda koristi kolektivno ponašanje za postupno poboljšanje rješenja, osiguravajući da se algoritam ne zaglavi u lokalnim optimumima, što je čest problem u drugim optimizacijskim metodama. Kroz iterativne korake ažuriranja i evaluacije, ABO algoritam omogućava pronalazak optimalnog rješenja u zadatom vremenskom okviru, uz smanjenje ukupnih troškova i povećanje efikasnosti sistema dostave.

4.3. VALIDACIJA I TESTIRANJE

U ovom poglavlju objašnjava se proces validacije i testiranja rješenja algoritma. Kako bi se osigurala pouzdanost rezultata, razvijen je sveobuhvatan testni okvir koji pokriva različite scenarije. Kroz definirane testne scenarije i provedbu testiranja analiziraju se performanse algoritma i uspoređuju se različiti pristupi.

4.3.1. Definicija testnih scenarija

Da bi se procijenila učinkovitost algoritma, osmišljeni su različiti testni scenariji koji pokrivaju širok raspon mogućih situacija u stvarnim uvjetima. Testni scenariji su definirani pomoću nekoliko ključnih parametara koji variraju kako bi se simulirale različite operativne okolnosti.

Ključni parametri testnih scenarija su:

- *brojKamiona*: Varira između 10, 20, 50 i 100. Ovo simulira različite veličine broja kamiona koji istovremeno obavljaju dostavu i omogućuje testiranje algoritma pod različitim opterećenjima.
- *maxBrojDostavnihMjesta*: Postavljen na 5 ili 10. Ovaj parametar definira maksimalan broj dostavnih mjesta koje svaki kamion može posjetiti u jednom dolasku.
- *zavrsnoMjesto*: Postavljen na 30 ili 40. Ovaj parametar definira raspon dostupnih dostavnih mjesta (od 19 do 30 (problem veće gustoće) ili od 19 do 40 (problem manje gustoće)) koje kamion može posjetiti.
- *vrijemeDolaska*: Generira se nasumično s granicom od 30 ili 60 minuta. Ovaj parametar simulira različite polaske kamiona na prema prvoj lokaciji, a početno vrijeme bi bilo jutarnje vrijeme u 8:00 ujutro kao vrijeme kad je promet u Rijeci najgušći.

Kombinacijom ovih parametara formirano je 32 različite kombinacije testnih scenarija. Za svaki scenarij generirano je 5 testnih skupova, čime je ukupan broj testova dosegao 160. Na ovaj način pokrivene su različite varijante koje se mogu pojaviti u stvarnom okruženju.

Primjer jednog testnog skupa, označenog kao 10_10_40_60, sadrži sljedeće podatke:

1,20,32,31,33,34,22,35,39,40,30,20
2,48,37,25
3,51,24,40,39,34,38,27,22,33
4,59,40,30,27
5,45,39,30,21,29
6,58,30
7,53,24,31,36,21,25,26,28
8,48,33,22,25,24,35
9,39,39,38,32,20,23
10,3,23,24,27,36,38,25

Ovaj testni skup sadrži 10 kamiona, svaki sa svojim vremenom polaska i nizom dostavnih mjesta. Na primjer, prvi kamion dolazi u 20. minuti i posjećuje dostavna mjesta 32, 31, 33, itd. U scenariju su uključeni parametri:

- 10 kamiona

- maksimalno 10 dostavnih mjesta po kamionu
- dostavna mjesta u rasponu od 19 do 40
- polazak kamiona u rasponu od 0 do 60 minuta.

4.3.2. Provedba testiranja

Za generiranje testnih podataka i provođenje testiranja razvijen je jednostavan program koji simulira različite scenarije. Ovaj program generira nasumične podatke koji simuliraju dolaske kamiona i raspored dostavnih mjesta. Svaki testni skup pokrenut je u sva tri algoritma, a rezultati su uspoređivani kako bi se identificirao najefikasniji pristup. Ukupni kod za generator testnih skupova je prikazan u nastavku:

```
public class DataGenerator {
    public static void main(String[] args) {
        Random randNumber = new Random();
        int brojKamiona = 10;
        int maxBrojDostavnihMjesta = 10;
        int pocetnoMjesto = 19;
        int zavrsnoMjesto = 40;
        for (int i = 1; i <= brojKamiona; i++) {
        int vrijemeDolaska = randNumber.nextInt(60);
int brojDostavnihMjestaKamion = randNumber.nextInt(maxBrojDostavnihMjesta) +
1;
 ArrayList<Integer> dostavnaMjestaKamion = new ArrayList<>();
for (int j = 0; j < brojDostavnihMjestaKamion; j++) {
int dostavnoMjestoTrenutno = randNumber.nextInt((zavrsnoMjesto -
pocetnoMjesto + 1)) + pocetnoMjesto;

if(!dostavnaMjestaKamion.contains(dostavnoMjestoTrenutno)){
dostavnaMjestaKamion.add(dostavnoMjestoTrenutno);
                }else{
                    j--;
                }
            }
            System.out.print(i + "," + vrijemeDolaska);
            for (Integer integer : dostavnaMjestaKamion) {
                System.out.print("," + integer);
            }
```

```
                System.out.println("");
            }
        }
    }
```

1. Definiranje parametara:
   Parametar *brojKamiona* postavlja broj kamiona koji će se simulirati u ovom testu, *maxBrojDostavnihMjesta* postavlja maksimalan broj dostavnih mjesta po kamionu (5 ili 10). Parametri *pocetnoMjesto* i *zavrsnoMjesto* definiraju raspon dostavnih mjesta (npr. 19-40 ili 19-30) te *vrijemeDolaska'* nasumično generira vrijeme dolaska kamiona (0-60 ili 0-30 minuta).

2. Generiranje dostavnih mjesta:
   Za svaki kamion se nasumično generira broj dostavnih mjesta unutar definiranog raspona. Pomaže se funkcijom *randNumber.nextInt()* kako bi se osigurala raznolikost u testnim skupovima. Duplikati dostavnih mjesta se izbjegavaju tako da se provjerava je li dostavno mjesto već dodano u listu.

3. Ispis rezultata:
   Generirani podaci se ispisuju u obliku CSV (eng. Comma Separated Values) formata, gdje svaki redak predstavlja jedan kamion i njegove dostavne točke.

Nakon generiranja testnih podataka, oni se prosljeđuju svakom od tri algoritma. Svaki algoritam optimizira rutu dostave na temelju ulaznih podataka, a zatim se uspoređuju rezultati. Cilj je pronaći algoritam koji daje najkraće ukupno vrijeme dostave. Na ovaj način osiguravamo rigorozno testiranje algoritama kroz različite scenarije, omogućujući nam da precizno odredimo najbolji pristup za optimizaciju dostave. Kroz analizu rezultata iz svih 160 testnih skupova možemo donijeti zaključke o performansama algoritma u različitim uvjetima.

5. ANALIZA REZULTATA

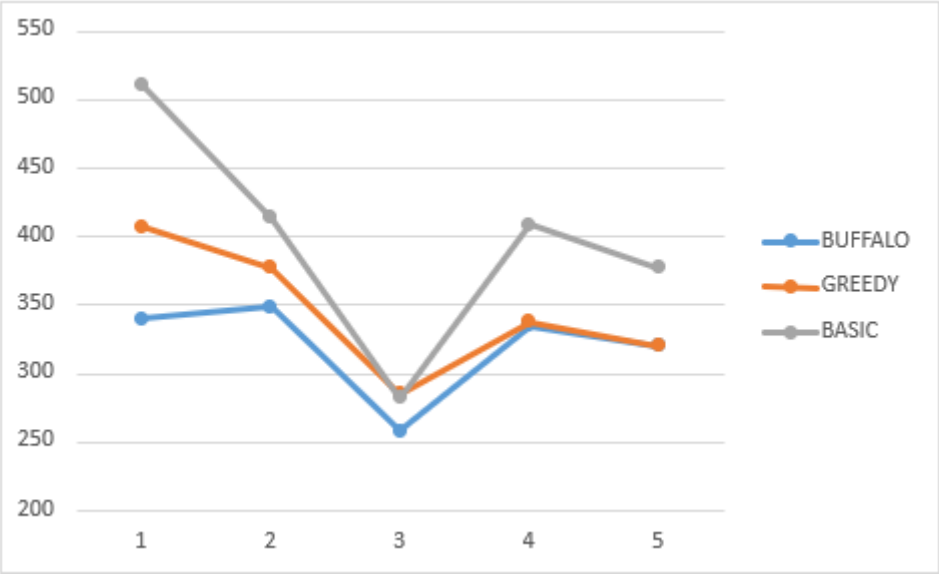
U ovom poglavlju provodi se analiza rezultata dobivenih iz primjene razvijenih algoritama na različitim testnim skupovima podataka. Cilj analize je usporediti učinkovitost algoritama na temelju različitih scenarija, kao i evaluirati njihovu primjenjivost u stvarnim prometnim uvjetima. Također se razmatraju potencijalna ograničenja algoritama te predlažu moguća poboljšanja.

5.1. USPOREDBA REZULTATA

Za potrebe analize rezultata, testirani su skupovi podataka s različitim brojem kamiona, proći će se sve kombinacije parametara za 10, 20, 50 i 100 kamiona za sva tri algoritma. Ukupno ima 480 rezultata za 3 algoritma, tj. 160 rezultata po algoritmu koji grafički prikazani na 32 kombinacije od 4 različita parametara, svaka kombinacija ima 5 testnih skupova. Rezultati dobiveni iz ovih testova omogućuju dublje razumijevanje performansi algoritama ovisno o opterećenju i složenosti prometne mreže, te će biti prikazani grafički.

5.1.1. Rezultati za testne skupove s 10 kamiona

Prva kombinacija ima 10 kamiona koji idu na maksimalno 5 dostavnih mjesta. Zadnja lokacija koja može bit uključena u dostavljačku rutu je označena brojem 30 (11 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 8:30 sati.
Grafički, na y osi označeno je ukupno vrijeme putovanja dostavnog vozila u minutama, a na x osi označene su instance testnih skupova. Rezultati algoritama za svih 5 instanci testnog skupa 10_5_30_30 prikazani su na slici 4.

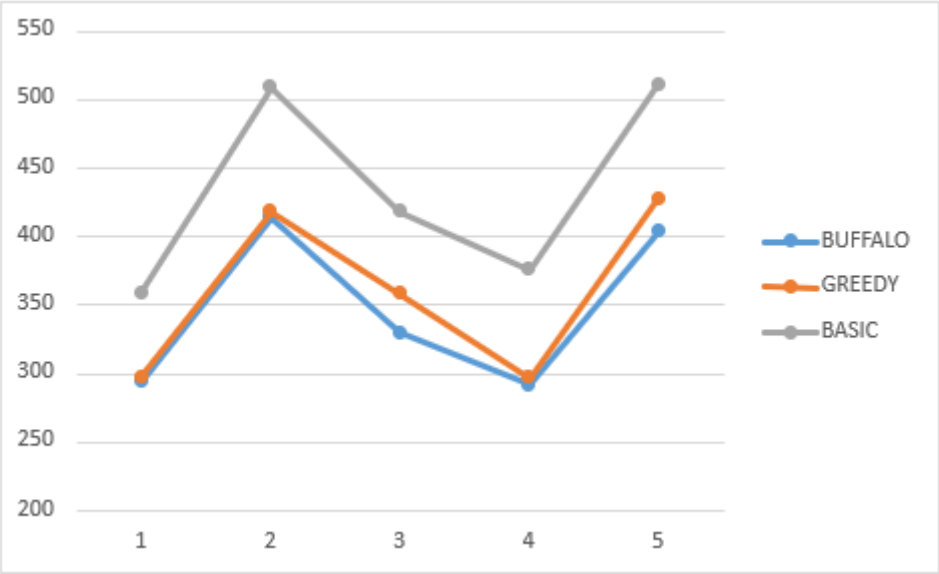


Slika 4. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 10_5_30_30

Izvor: izradio student

Iz slike je vidljivo da, s obzirom na mali broj lokacija i raznolikosti parametara, ABO algoritam je pokazao najbolje vrijeme u svih 5 slučajeva. Sebičan algoritam je u 5. skupu imao jednako rješenje kao i ABO algoritam što upućuje da je ta nasumična kombinacija bila jednako dobra i za sebičan algoritam, međutim, jednako tako je u 3. testnom skupu jednostavan algoritam bio od sebičnog algoritma za 3 minute, što može ukazivati da ako se u testnom skupu dobro poklopi redoslijed lokacija, da i najjednostavniji algoritam može pokazati dobro vrijeme.

Druga kombinacija ima 10 kamiona koji idu na maksimalno 5 dostavnih mjesta. Zadnja lokacija koja može biti uključena u dostavljačku rutu je označena brojem 30 (11 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 9:00 sati. Rezultati algoritama za svih 5 instanci testnog skupa 10_5_30_60 prikazani su na slici 5.

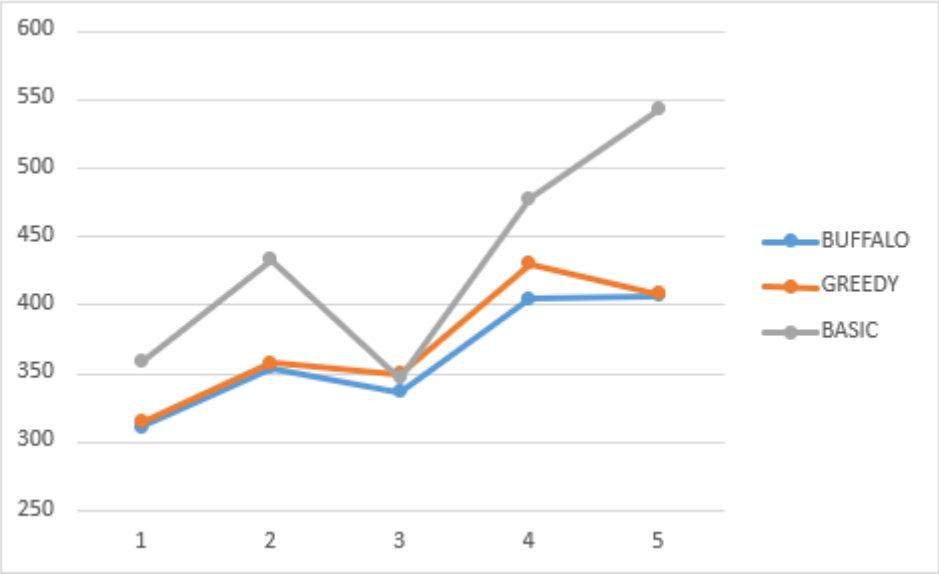


Slika 5. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 10_5_30_60

Izvor: izradio student

Iz slike je opet vidljivo je ABO algoritam postizao najniža vremena ukupnih dostava, s time što je sebičan algoritam pokazivao skoro jednako dobra vremena (4-5 min razlike) u većini slučajeva. Jednostavan algoritam je imao najgora vremena u svim slučajevima.

Treća kombinacija ima 10 kamiona koji idu na maksimalno 5 dostavnih mjesta. Zadnja lokacija koja može biti uključena u dostavljačku rutu je označena brojem 40 (21 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 8:30 sati. Rezultati algoritama za svih 5 instanci testnog skupa 10_5_40_30 prikazani su na slici 6.

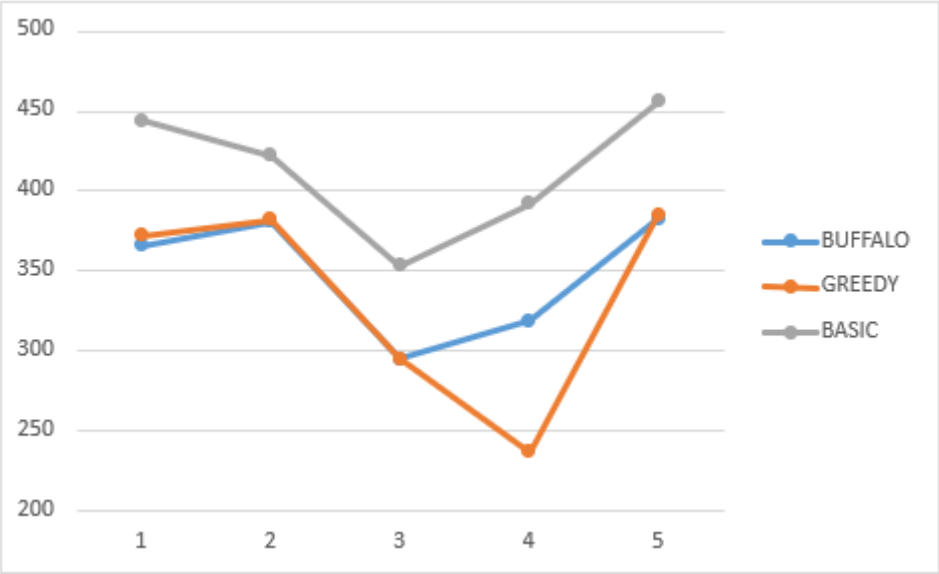


Slika 6. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 10_5_40_30

Izvor: izradio student

Isto kao i u prethodnom slučaju, vidi se da je ABO algoritam dobio najbolja vremena, s time da je sebičan algoritam opet jako dobra sekundarna opcija jer vremena su mu između 1-10 minuta lošija od ABO algoritma. Jednostavan algoritam je u 3. testnom skupu pokazao 3 minute bolje vrijeme od sebičnog algoritma, ali je gori u svim ostalim slučajevima.

Četvrta kombinacija ima 10 kamiona koji idu na maksimalno 5 dostavnih mjesta. Zadnja lokacija koja može biti uključena u dostavljačku rutu je označena brojem 40 (21 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 9:00 sati. Rezultati algoritama za svih 5 instanci testnog skupa 10_5_40_60 prikazani su na slici 7.

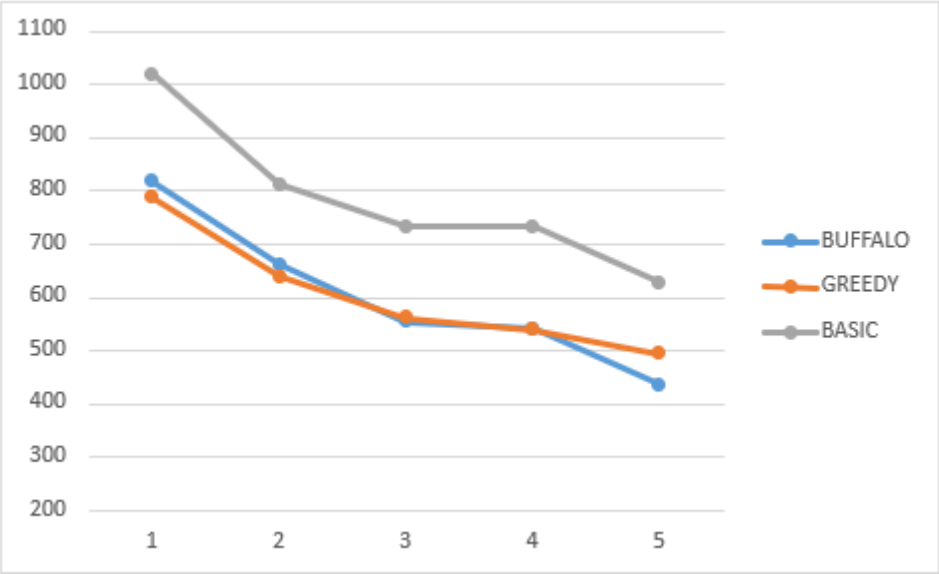


Slika 7. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 10_5_40_60

Izvor: izradio student

Za razliku od prijašnjih rezultata, ABO algoritam pokazuje najbolje vrijeme u 4 testna skupa među kojima je 3. jednak kao i kod sebičnog algoritma. U 4. testnom skupu je sebičan algoritam pokazao najbolje vrijeme dok je jednostavan algoritam najlošiji u svim testnim skupovima.

Peta kombinacija ima 10 kamiona koji idu na maksimalno 10 dostavnih mjesta. Zadnja lokacija koja može biti uključena u dostavljačku rutu je označena brojem 30 (11 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 8:30 sati. Rezultati algoritama za svih 5 instanci testnog skupa 10_10_30_30 prikazani su na slici 8.

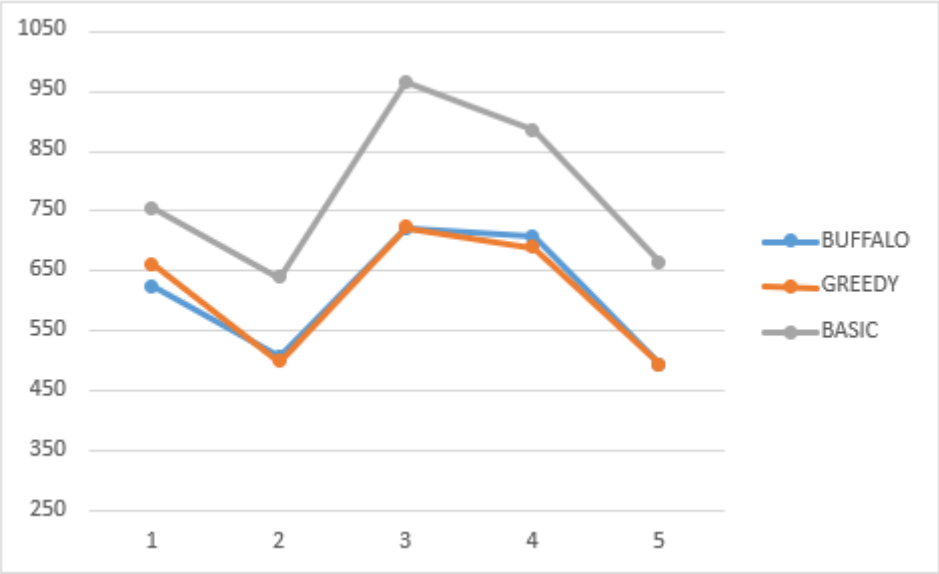


Slika 8. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 10_10_30_30

Izvor: izradio student

U ovoj kombinaciji, sebičan algoritam je imao najbolje rezultate u 3 testna skupa, dok je ABO algoritam imao u 2 testna skupa. Svejedno, razlika između njihovih rezultata je minimalna. Jednostavan algoritam je opet uvjerljivo imao najgore rezultate.

Šesta kombinacija ima 10 kamiona koji idu na maksimalno 10 dostavnih mjesta. Zadnja lokacija koja može biti uključena u dostavljačku rutu je označena brojem 30 (11 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 9:00 sati Rezultati algoritama za svih 5 instanci testnog skupa 10_10_30_60 prikazani su na slici 9.

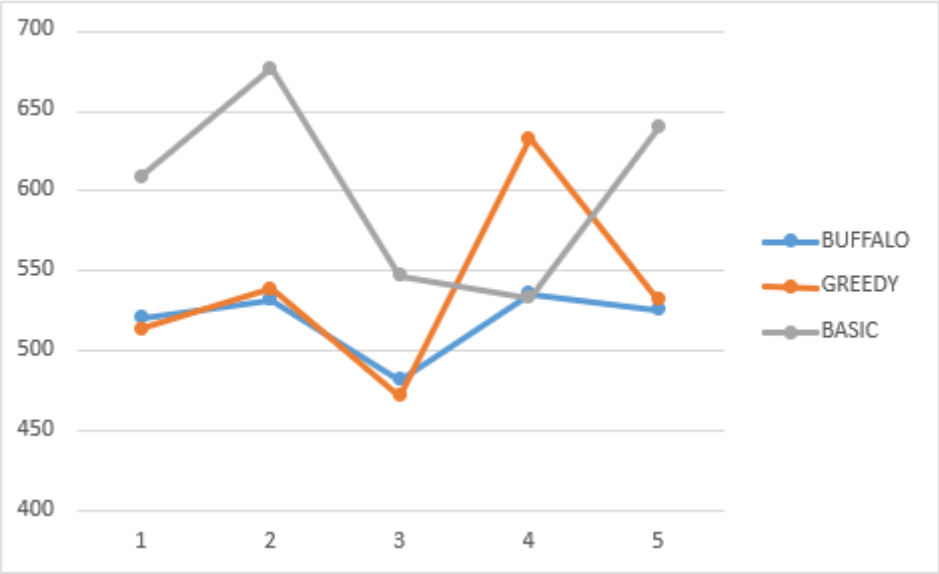


Slika 9. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 10_10_30_60

Izvor: izradio student

U ovoj kombinaciji su ABO algoritam i sebičan algoritam oboje imali 3 najbolja rezultata, 1. i 3. pripadaju ABO algoritmu a 2. i 4. pripadaju sebičnom algoritmu, dok su oboje imali jednaki rezultat 5. testnog skupa. Jednostavan algoritam je i dalje imao najlošija vremena.

Sedma kombinacija ima 10 kamiona koji idu na maksimalno 10 dostavnih mjesta. Zadnja lokacija koja može biti uključena u dostavljačku rutu je označena brojem 40 (21 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 8:30 sati. Rezultati algoritama za svih 5 instanci testnog skupa 10_10_40_30 prikazani su na slici 10.

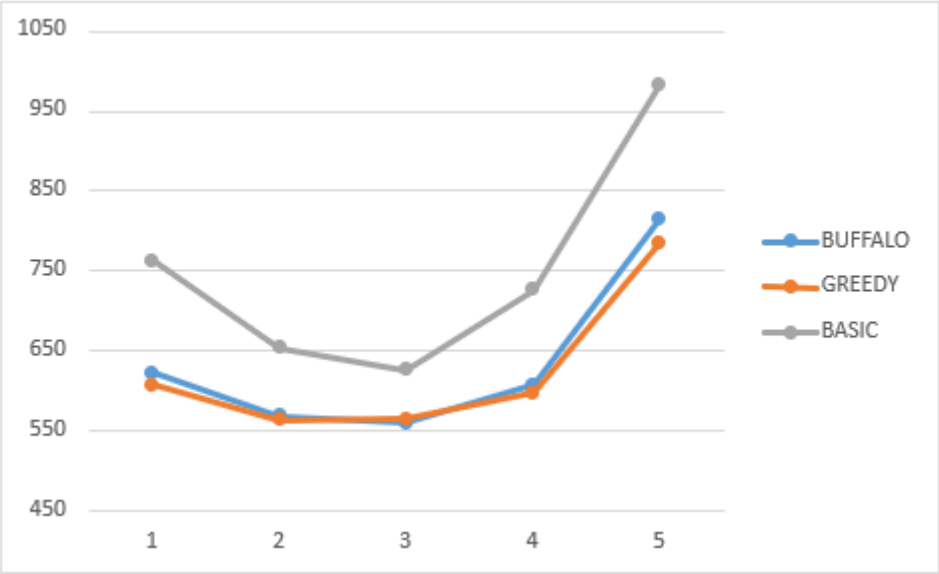


Slika 10 Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 10_10_40_30

Izvor: izradio student

U ovom slučaju ABO algoritam je imao najbolja vremena 3 puta dok je sebičan algoritam imao 2 puta. Međutim, u 4. testnom skupu je jednostavan algoritam bio bolji za 100 minuta te samo za 3 minute lošiji od ABO algoritma. Velika je vjerojatnost da su nasumični brojevi 4. testnog skupa jako godili načinu rada jednostavnog algoritma.

Osma kombinacija ima 10 kamiona koji idu na maksimalno 10 dostavnih mjesta. Zadnja lokacija koja može biti uključena u dostavljačku rutu je označena brojem 40 (21 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 9:00 sati. Rezultati algoritama za svih 5 instanci testnog skupa 10_10_40_60 prikazani su na slici 11.



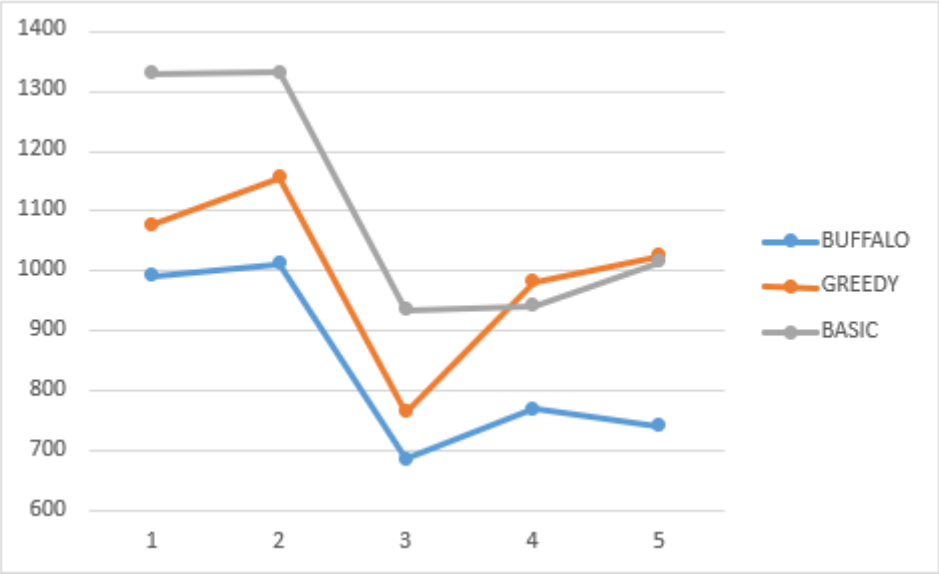
Slika 11. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 10_10_40_60

Izvor: izradio student

U ovom testnom skupu sebičan algoritam je pokazao najbolje rezultate u 4 testna skupa, jedino u 3. testnom skupu je bio za 5 minuta lošiji od ABO algoritma. ABO algoritam je imao između 5 minuta i 14 minuta lošije rezultate od sebičnog algoritma. Jednostavan algoritam je po običaju imao najlošije rezultate.

5.1.2. Rezultati za testne skupove s 20 kamiona

Deveta kombinacija ima 20 kamiona koji idu na maksimalno 5 dostavnih mjesta. Zadnja lokacija koja može bit uključena u dostavljačku rutu je označena brojem 30 (11 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 8:30 sati. Rezultati algoritama za svih 5 instanci testnog skupa 20_5_30_30 prikazani su na slici 12.

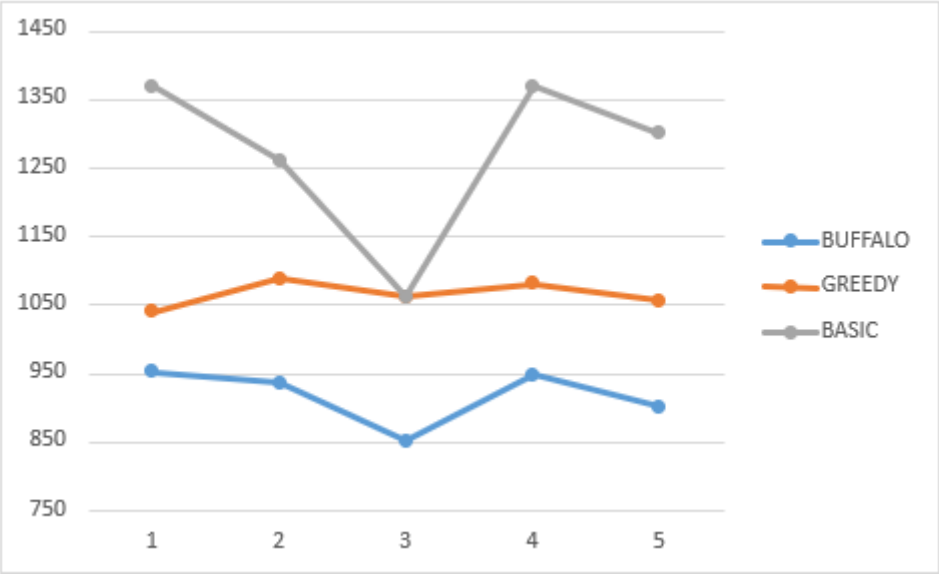


Slika 12. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 20_5_30_30

Izvor: izradio student

U prvoj kombinaciji sa 20 kamiona ABO algoritam se pokazao najbolji, te je otprilike između 70 minuta i 100 minuta bio bolji od sebičnog algoritma i jednostavnog algoritma. S druge strane, jednostavan algoritam je u zadnja 3 slučaja imao bolja vremena od sebičnog algoritma.

Deseta kombinacija ima 20 kamiona koji idu na maksimalno 5 dostavnih mjesta. Zadnja lokacija koja može bit uključena u dostavljačku rutu je označena brojem 30 (11 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 9:00 sati. Rezultati algoritama za svih 5 instanci testnog skupa 20_5_30_60 prikazani su na slici 13.

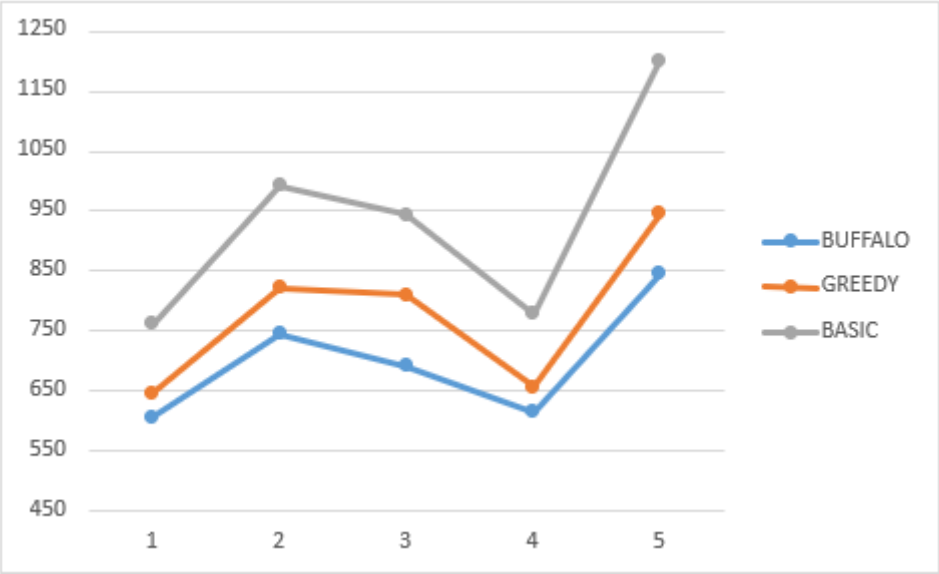


Slika 13. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 20_5_30_60

Izvor: izradio student

Rezultati ove kombinacije kažu da je ABO algoritam imao najbolja vremena u svim testnim skupovima, također su jednostavan algoritam i sebičan algoritam imali isto vrijeme u 3. testnom skupu.

Jedanaesta kombinacija ima 20 kamiona koji idu na maksimalno 5 dostavnih mjesta. Zadnja lokacija koja može bit uključena u dostavljačku rutu je označena brojem 40 (21 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 8:30 sati. Rezultati algoritama za svih 5 instanci testnog skupa 20_5_40_30 prikazani su na slici 14.

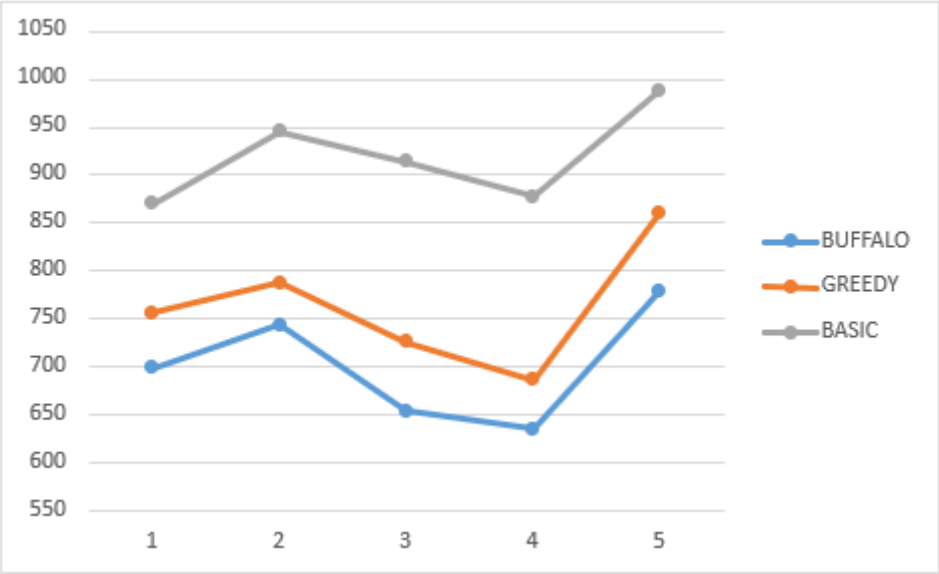


Slika 14. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 20_5_40_30

Izvor: izradio student

U ovoj kombinaciji ABO algoritam ima vidljivo najbolje rezultate, sebičan algoritam ima nešto lošije rezultate, a jednostavan algoritam ima najlošije rezultate.

Dvanaesta kombinacija ima 20 kamiona koji idu na maksimalno 5 dostavnih mjesta. Zadnja lokacija koja može bit uključena u dostavljačku rutu je označena brojem 40 (21 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 9:00 sati. Rezultati algoritama za svih 5 instanci testnog skupa 20_5_40_60 prikazani su na slici 15.

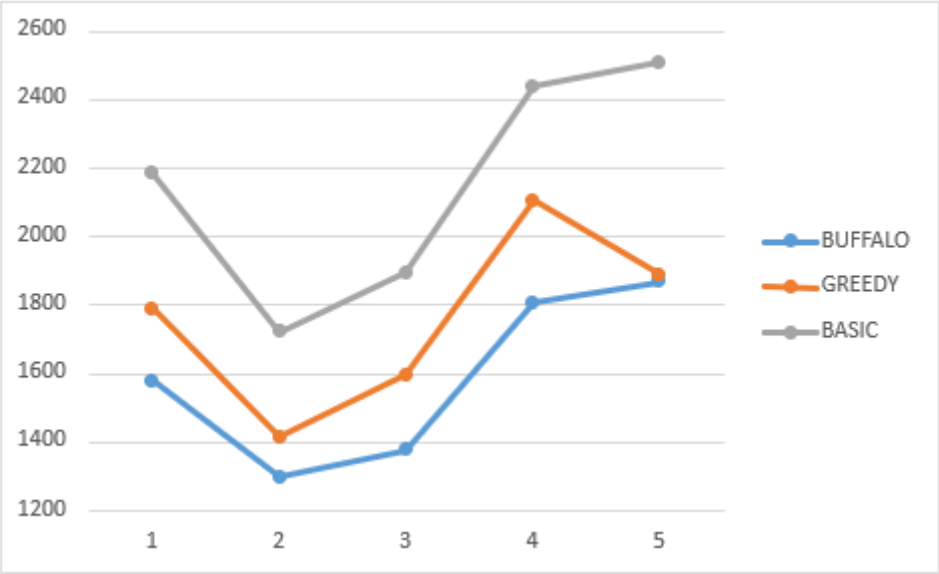


Slika 15. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 20_5_40_60

Izvor: izradio student

U ovoj kombinaciji je opet vidljivo da je ABO algoritam imao najbolje rezultate. Sebičan algoritam nije daleko od ABO algoritma glede ukupnih vremena, međutim jednostavan algoritam ima sve gore i gore rezultate u usporedbi sa prijašnjim kombinacijama. Razlika između kompliciranijih algoritama i jednostavnog algoritma zbog sve većeg i kompliciranijeg testnog skupa.

Trinaesta kombinacija ima 20 kamiona koji idu na maksimalno 10 dostavnih mjesta. Zadnja lokacija koja može bit uključena u dostavljačku rutu je označena brojem 30 (11 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 8:30 sati. Rezultati algoritama za svih 5 instanci testnog skupa 20_10_30_30 prikazani su na slici 16.

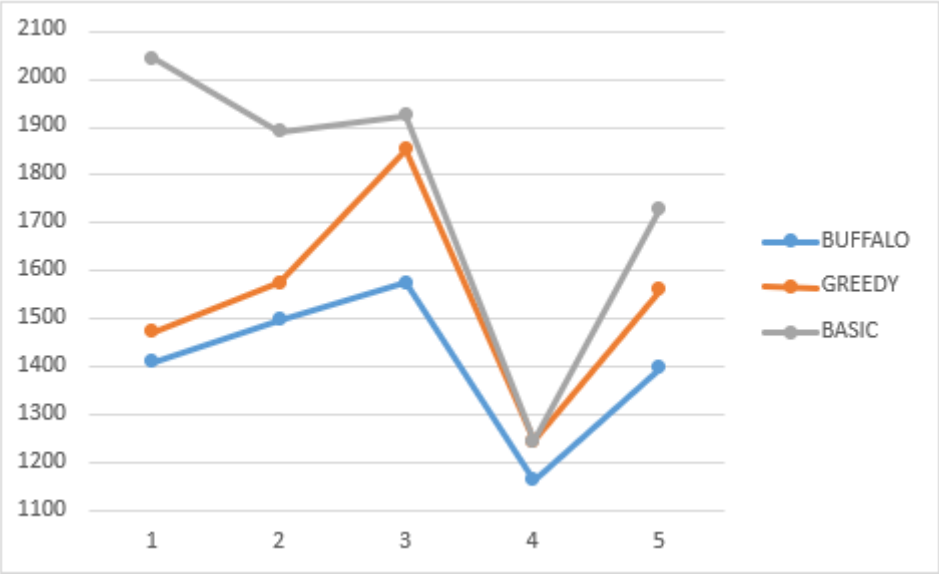


Slika 16. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 20_10_30_30

Izvor: izradio student

U ovoj kombinaciji ABO algoritam ima najbolje rezultate međutim u 5. testnom skupu sebičan algoritam ima za samo 19 minuta lošiji rezultat. To može značiti da ovaj testni skup se potrefio da odgovara sebičnom algoritmu jednako dobro kao i ABO algoritmu. Jednostavan algoritam je i dalje najlošiji.

Četrnaesta kombinacija ima 20 kamiona koji idu na maksimalno 10 dostavnih mjesta. Zadnja lokacija koja može bit uključena u dostavljačku rutu je označena brojem 30 (11 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 9:00 sati. Rezultati algoritama za svih 5 instanci testnog skupa 20_10_30_60 prikazani su na slici 17.

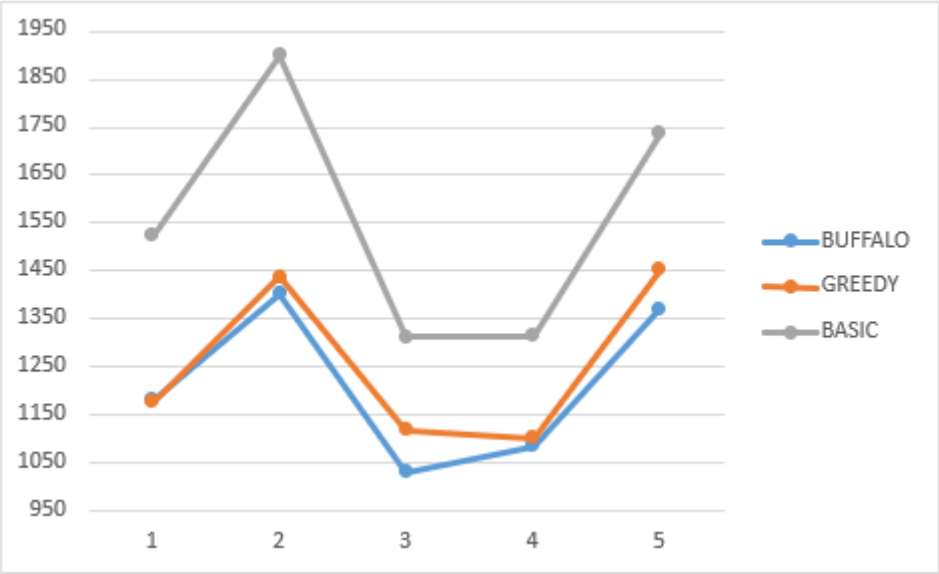


Slika 17. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 20_10_30_60

Izvor: izradio student

U ovoj kombinaciji ABO algoritam opet ima najbolje rezultate dok sebičan algoritam ima nešto lošije pogotovo u 3. testnom skupu, te u 4. testnom skupu sebičan algoritam i jednostavan imaju isto vrijeme, pojava koja je sve rijeđa kako postaju kompleksnije kombinacije.

Petnaesta kombinacija ima 20 kamiona koji idu na maksimalno 10 dostavnih mjesta. Zadnja lokacija koja može bit uključena u dostavljačku rutu je označena brojem 40 (21 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 8:30 sati. Rezultati algoritama za svih 5 instanci testnog skupa 20_10_40_30 prikazani su na slici 18.

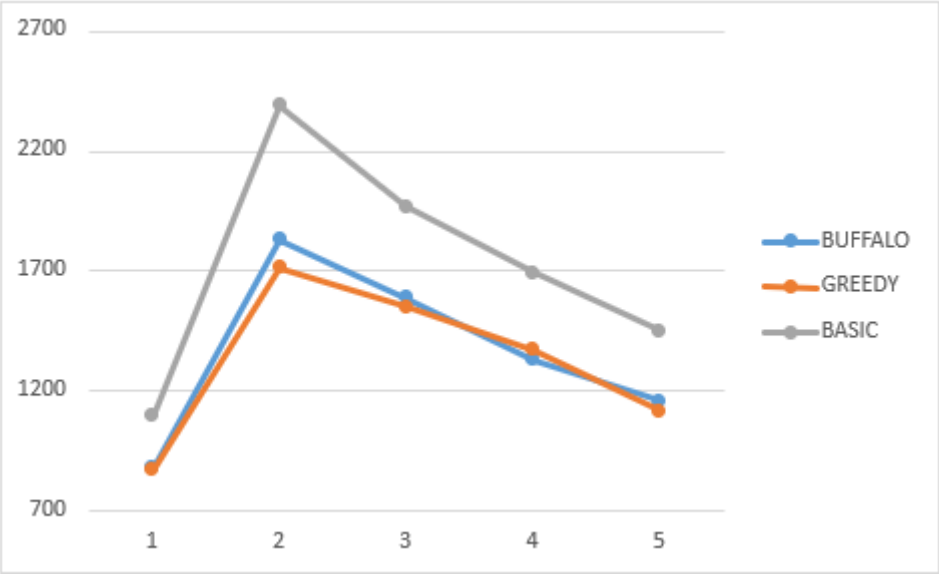


Slika 18. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 20_10_40_30

Izvor: izradio student

U ovoj kombinaciji ABO je imao najbolje rezultate u 4 testna skupa, te u sva 4 je imao relativno blisko vrijeme sebičnom algoritmu, koji je imao najbolji rezultat u 1. testnom skupu za 5 minuta razlike. Jednostavan algoritam je imao najlošije rezultate

Šesnaesta kombinacija ima 20 kamiona koji idu na maksimalno 10 dostavnih mjesta. Zadnja lokacija koja može bit uključena u dostavljačku rutu je označena brojem 40 (21 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 9:00 sati. Rezultati algoritama za svih 5 instanci testnog skupa 20_10_40_60 prikazani su na slici 19.



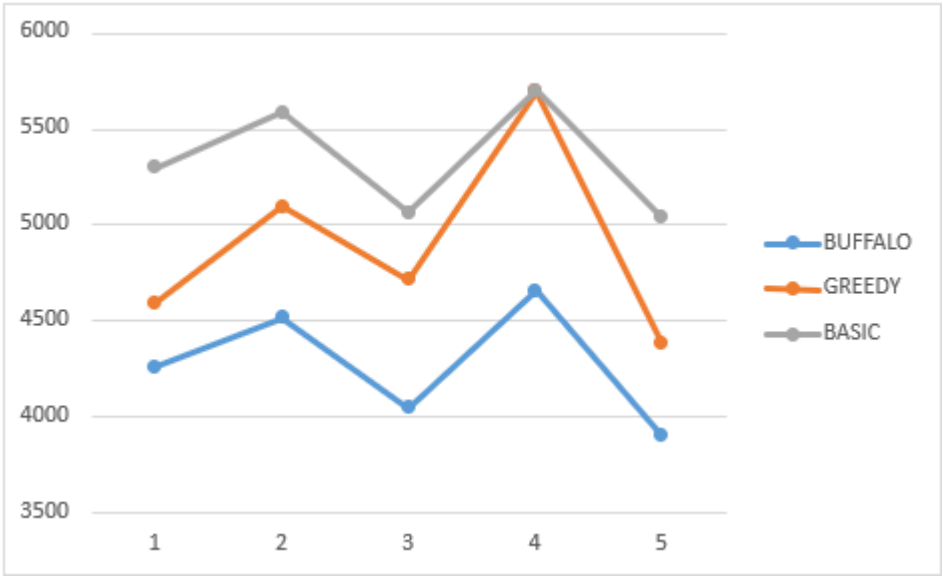
Slika 19. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 20_10_40_60

Izvor: izradio student

Ovo su zanimljivi rezultati. U ovoj kombinaciji sebičan algoritam je imao najbolji rezultat u 4 testna skupa dok je ABO algoritam imao najbolji rezultat samo u 4. testnom skupu. To znači da su rezultati za ovu kombinaciju gotovo identični istoj ovoj kombinaciji ali sa 10 kamiona. Jednostavan algoritam je imao najlošije rezultate.

5.1.3. Rezultati za testne skupove sa 50 kamiona

Sedamnaesta kombinacija ima 50 kamiona koji idu na maksimalno 5 dostavnih mjesta. Zadnja lokacija koja može bit uključena u dostavljačku rutu je označena brojem 30 (11 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 8:30 sati. Rezultati algoritama za svih 5 instanci testnog skupa 50_5_30_30 prikazani su na slici 20.

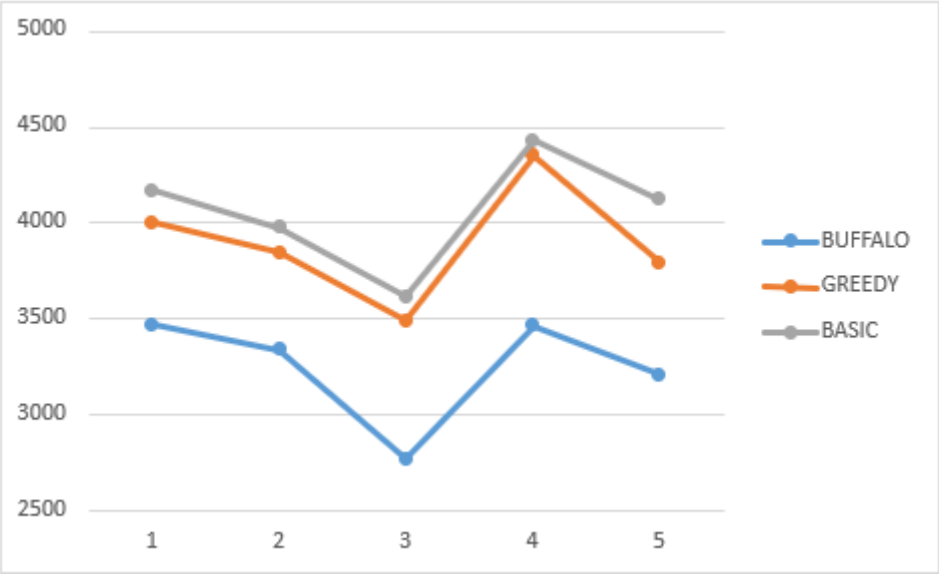


Slika 20. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 50_5_30_30

Izvor: izradio student

U usporedbi sa kombinacijama sa 20 kamiona, ukupne minute su se u prosjeku povećale za par tisuća minuta, to također rezultira i većim razlikama između dobivenih rezultata algoritama. U ovoj kombinaciji, ABO algoritam je dao daleko najbolje rezultate. Sebičan algoritam otprilike ima jednaku razliku od jednostavnog algoritma kao i od ABO algoritma, s tim što u 4. testnom skupu se potrefilo da je razlika između sebičnog algoritma i jednostavnog algoritma samo 5 minuta.

Osamnaesta kombinacija ima 50 kamiona koji idu na maksimalno 5 dostavnih mjesta. Zadnja lokacija koja može bit uključena u dostavljačku rutu je označena brojem 30 (11 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 9:00 sati. Rezultati algoritama za svih 5 instanci testnog skupa 50_5_30_60 prikazani su na slici 21.

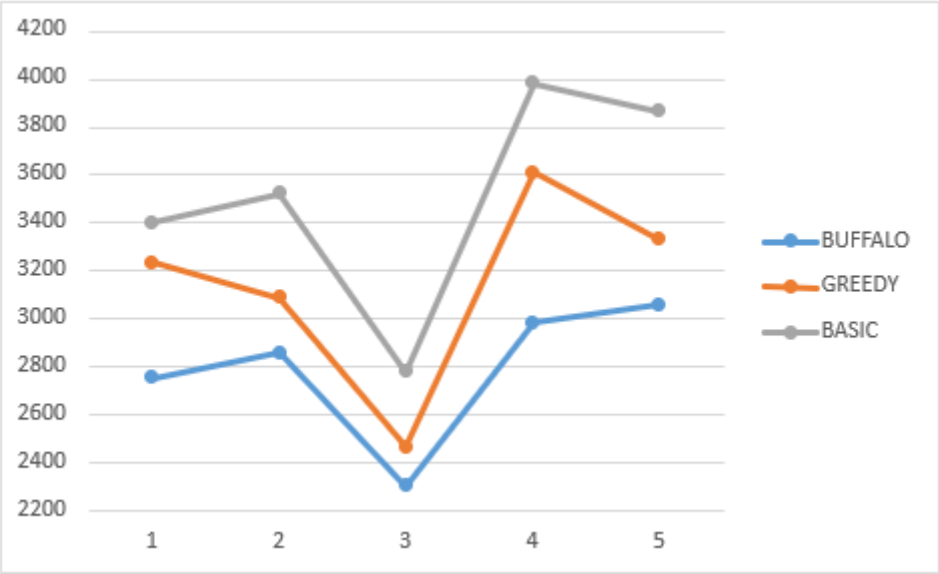


Slika 21. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 50_5_30_60

Izvor: izradio student

U ovoj kombinaciji ABO algoritam je davao daleko najbolje rezultate doku su sebičan algoritam i jednostavan algoritam imali nešto bliže rezultate naspram prijašnjoj kombinaciji. Međutim, u usporedbi sa ABO algoritmom, njihovi rezultati su dosta lošiji.

Devetnaesta kombinacija ima 50 kamiona koji idu na maksimalno 5 dostavnih mjesta. Zadnja lokacija koja može bit uključena u dostavljačku rutu je označena brojem 40 (21 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 8:30 sati. Rezultati algoritama za svih 5 instanci testnog skupa 50_5_40_30 prikazani su na slici 22.

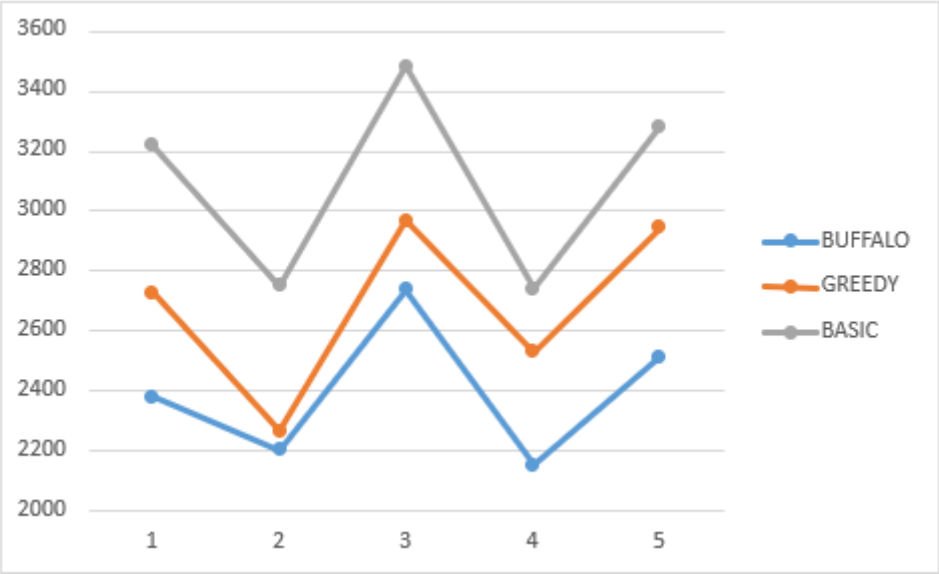


Slika 22. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 50_5_40_30

Izvor: izradio student

Jednako kao i do sada, u ovoj kombinaciji ABO algoritam daje nabolje rješenje, dok sebičan algoritam daje lošija rješenja, a jednostavan algoritam daje najgora rješenja.

Dvadeseta kombinacija ima 50 kamiona koji idu na maksimalno 5 dostavnih mjesta. Zadnja lokacija koja može bit uključena u dostavljačku rutu je označena brojem 40 (21 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 9:00 sati. Rezultati algoritama za svih 5 instanci testnog skupa 50_5_40_60 prikazani su na slici 23.

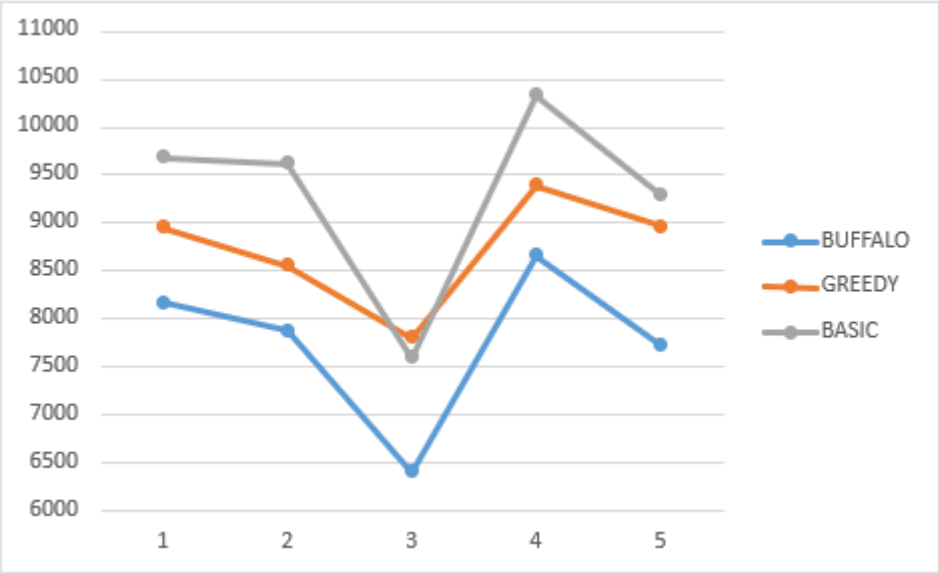


Slika 23. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 50_5_40_60

Izvor: izradio student

U ovoj kombinaciji ABO algoritam daje najbolje rezultate, jedino što mu je u 2. testnom skupu za vremenski blizu (63 min) rezultat od sebičnog algoritma. Ostatak je identičan kao i prije.

Dvadeset prva kombinacija ima 50 kamiona koji idu na maksimalno 10 dostavnih mjesta. Zadnja lokacija koja može bit uključena u dostavljačku rutu je označena brojem 30 (11 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 8:30 sati. Rezultati algoritama za svih 5 instanci testnog skupa 50_10_30_30 prikazani su na slici 24.

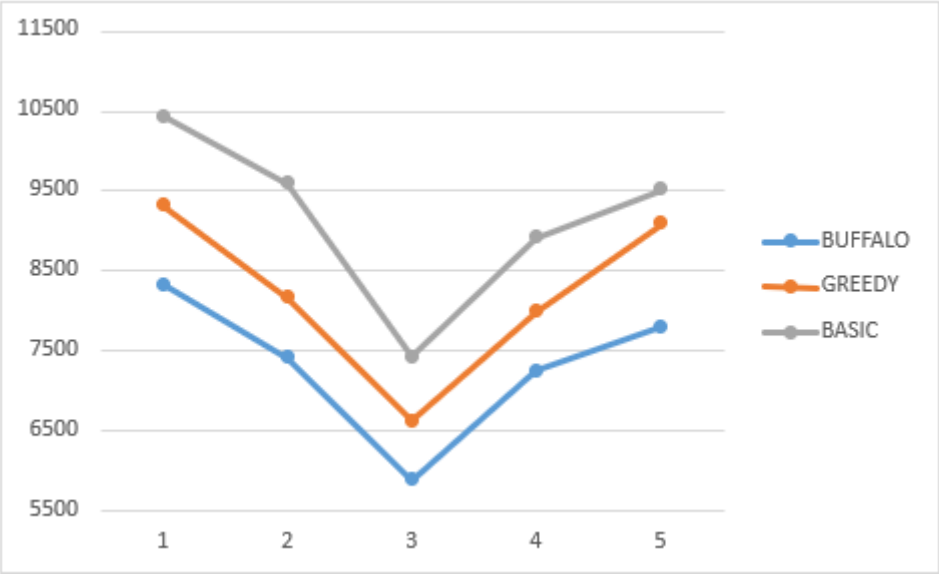


Slika 24. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 50_10_30_30

Izvor: izradio student

U ovoj kombinaciji ABO algoritam je daje daleko najbolje rezultate. Sebičan algoritam je drugi najbolji algoritam po rezultatima osim u 3. testom skupu gdje je jednostavan algoritam bolji za 300 minuta.

Dvadeset druga kombinacija ima 50 kamiona koji idu na maksimalno 10 dostavnih mjesta. Zadnja lokacija koja može bit uključena u dostavljačku rutu je označena brojem 30 (11 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 9:00 sati. Rezultati algoritama za svih 5 instanci testnog skupa 50_10_30_60 prikazani su na slici 25.

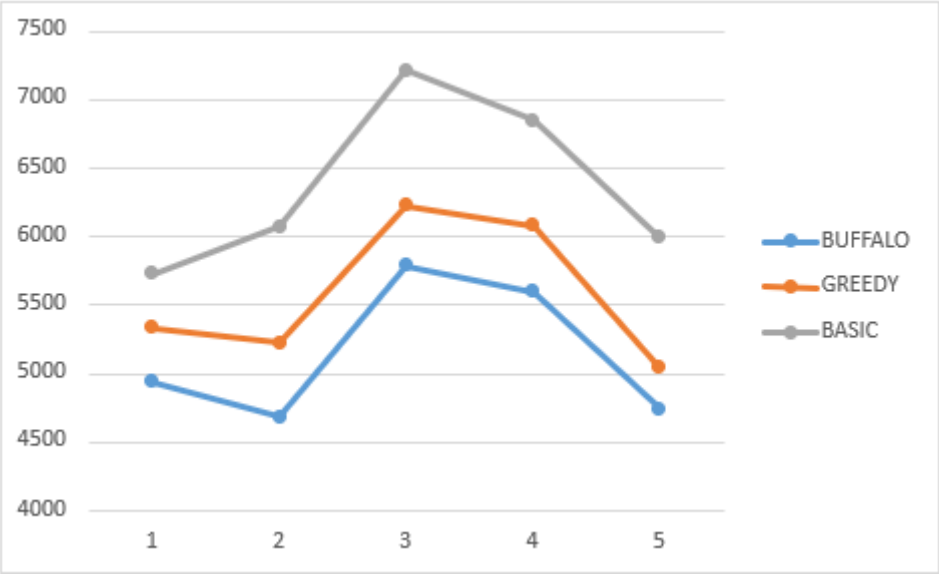


Slika 25. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 50_10_30_60

Izvor: izradio student

Identična situacija kao i prije, ABO algoritam daje najbolje rezultate, sebičan algoritam ima nešto lošije rezultate a jednostavan algoritam ima najgore rezultate.

Dvadeset treća kombinacija ima 50 kamiona koji idu na maksimalno 10 dostavnih mjesta. Zadnja lokacija koja može bit uključena u dostavljačku rutu je označena brojem 40 (21 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 8:30 sati. Rezultati algoritama za svih 5 instanci testnog skupa 50_10_40_30 prikazani su na slici 26.

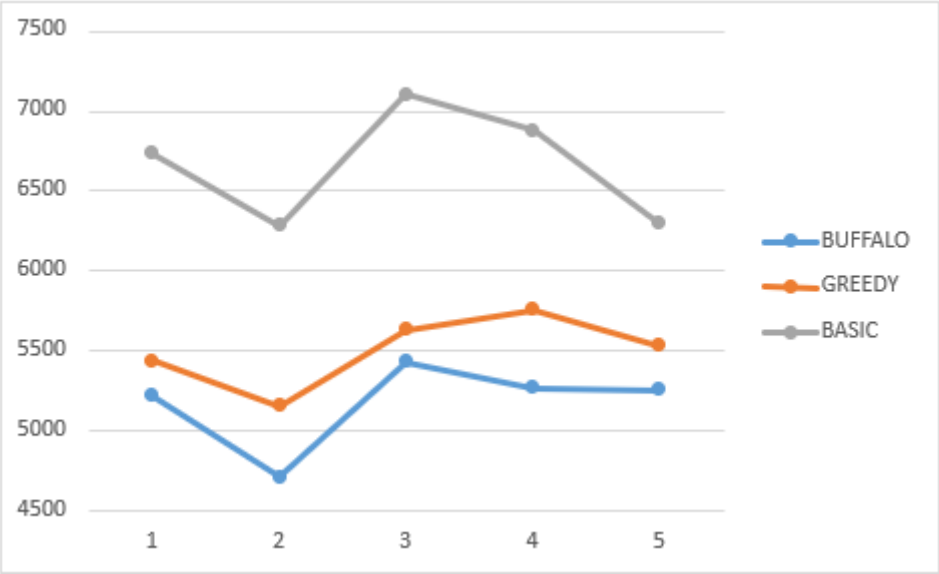


Slika 26. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 50_10_40_30

Izvor: izradio student

Isti rezultati za ovu kombinaciju kao i do sada, ABO algoritam daje najbolje rezultate, sebičan algoritam ima nešto lošije rezultate a jednostavan algoritam ima najgore rezultate.

Dvadeset četvrta kombinacija ima 50 kamiona koji idu na maksimalno 10 dostavnih mjesta. Zadnja lokacija koja može bit uključena u dostavljačku rutu je označena brojem 40 (21 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 9:00 sati. Rezultati algoritama za svih 5 instanci testnog skupa 50_10_40_60 prikazani su na slici 27.



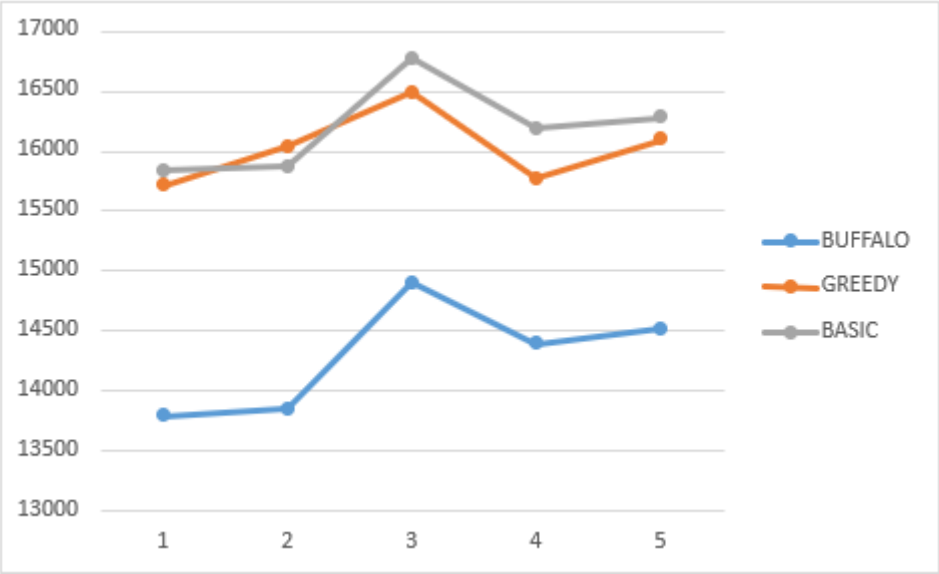
Slika 27. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 50_10_40_60

Izvor: izradio student

Slični rezultati kao i do sada. ABO algoritam daje najbolje rezultate, sebičan algoritam daje nešto lošije rezultate, ali jednostavan algoritam je dao mnogo gore rezultate u ovoj kombinaciji u usporedbi sa prijašnjim razlikama između jednostavnog algoritma i sebičnog algoritma.

5.1.4. Rezultati za testne skupove sa 100 kamiona

Dvadeset peta kombinacija ima 100 kamiona koji idu na maksimalno 5 dostavnih mjesta. Zadnja lokacija koja može bit uključena u dostavljačku rutu je označena brojem 30 (11 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 8:30 sati. Rezultati algoritama za svih 5 instanci testnog skupa 100_5_30_30 prikazani su na slici 28.

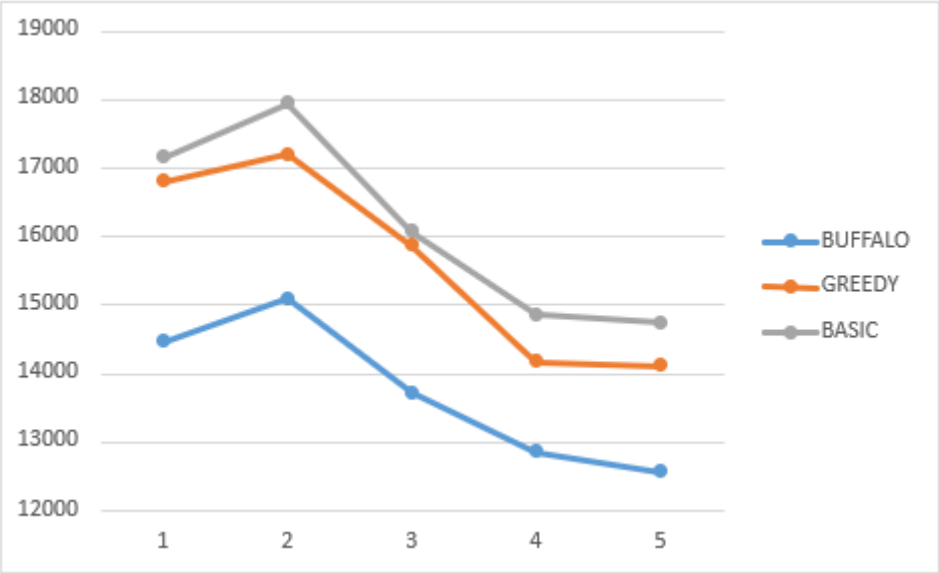


Slika 28. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 100_5_30_30

Izvor: izradio student

Kod kombinacija sa 100 dostavnih vozila, jednako kao i sa 50, povećale su se ukupne minute dostave sa par tisuća na desetke tisuća. Te su razlike vremena između algoritama još veće.
U ovoj kombinaciji ABO algoritam ima daleko najbolje rezultate. Sebičan algoritam ima dosta lošije rezultate, u 2. testnom skupu se potrefilo da je jednostavan algoritam imao bolje rezultate od sebičnog algoritma.

Dvadeset šesta kombinacija ima 100 kamiona koji idu na maksimalno 5 dostavnih mjesta. Zadnja lokacija koja može bit uključena u dostavljačku rutu je označena brojem 30 (11 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 9:00 sati. Rezultati algoritama za svih 5 instanci testnog skupa 100_5_30_60 prikazani su na slici 29.

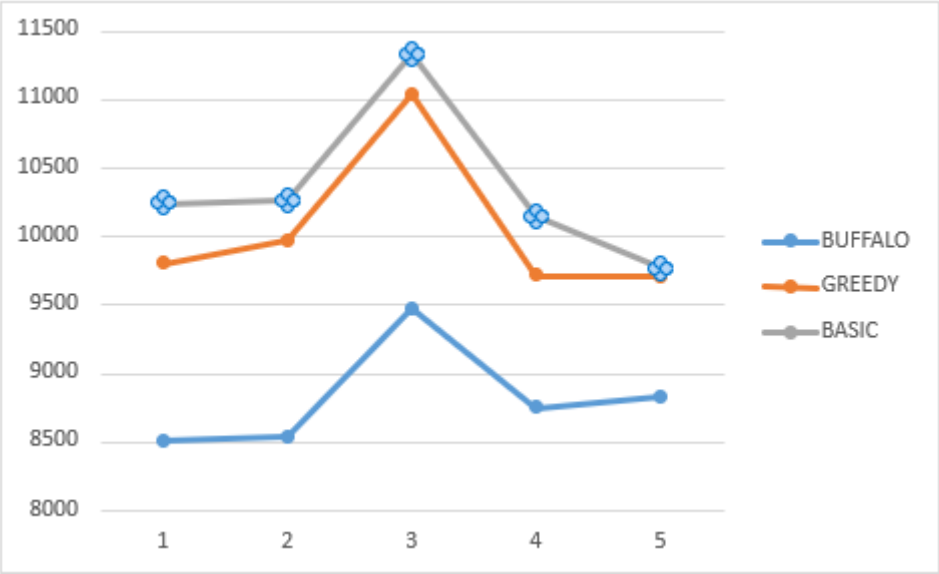


Slika 29. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 100_5_30_60

Izvor: izradio student

Jednako kao i kod prijašnje kombinacije, ABO algoritam daje najbolje rezultate. Sebičan algoritam ima nešto lošije rezultate, osim u bliskoj vremenskoj razlici u 3. testnom skupu. Jednostavan algoritam daje najgore rezultate.

Dvadeset sedma kombinacija ima 100 kamiona koji idu na maksimalno 5 dostavnih mjesta. Zadnja lokacija koja može bit uključena u dostavljačku rutu je označena brojem 40 (21 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 8:30 sati. Rezultati algoritama za svih 5 instanci testnog skupa 100_5_40_30 prikazani su na slici 30.

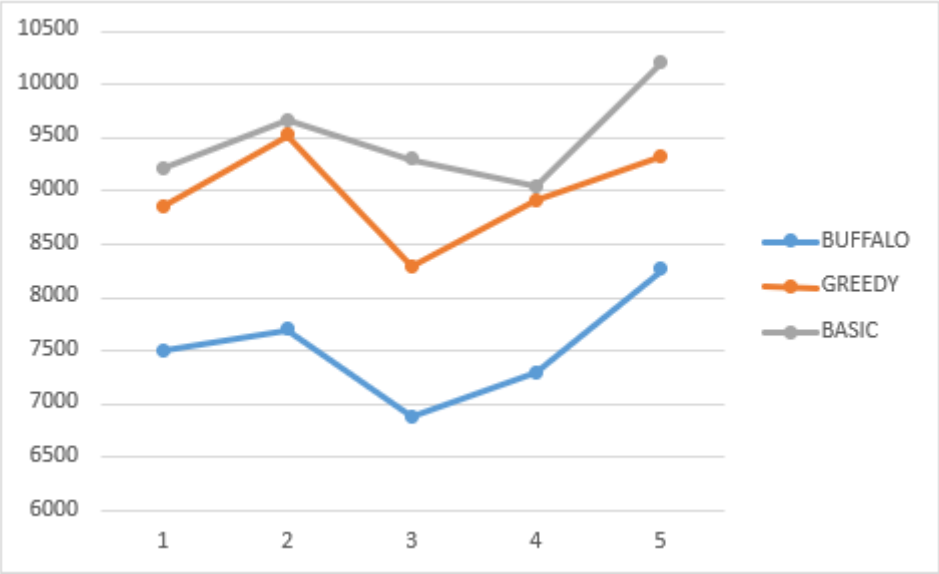


Slika 30. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 100_5_40_30

Izvor: izradio student

Slično kao i kod prijašnjih kombinacija, ABO algoritam daje najbolje rezultate, a u ovom slučaju sebičan algoritam i jednostavan algoritam daju dosta lošije rezultate, ali nije velika međusobna razlika.

Dvadeset osma kombinacija ima 100 kamiona koji idu na maksimalno 5 dostavnih mjesta. Zadnja lokacija koja može bit uključena u dostavljačku rutu je označena brojem 40 (21 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 9:00 sati. Rezultati algoritama za svih 5 instanci testnog skupa 100_5_40_60 prikazani su na slici 31.

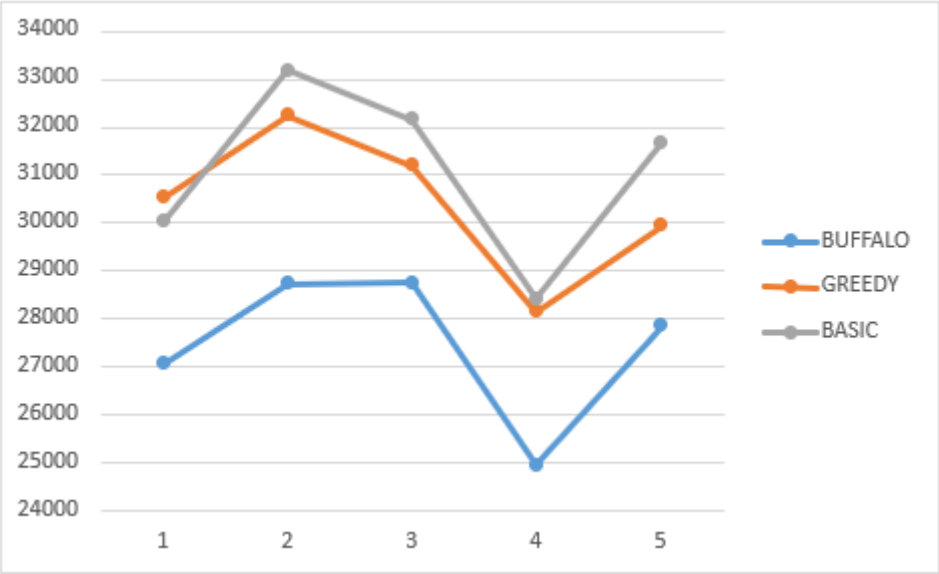


Slika 31. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 100_5_40_60

Izvor: izradio student

Isto kao i do sada, ABO algoritam daje najbolje rezultate, sebičan algoritam daje lošije rezultate u usporedbi sa ABO algoritmom, te u ovoj kombinaciji na 2. i 4. testnom skupu su jednostavnom algoritmu i sebičnom algoritmu rezultati dosta vremenski bliski.

Dvadeset deveta kombinacija ima 100 kamiona koji idu na maksimalno 10 dostavnih mjesta. Zadnja lokacija koja može bit uključena u dostavljačku rutu je označena brojem 30 (11 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 8:30 sati. Rezultati algoritama za svih 5 instanci testnog skupa 100_10_30_30 prikazani su na slici 32.

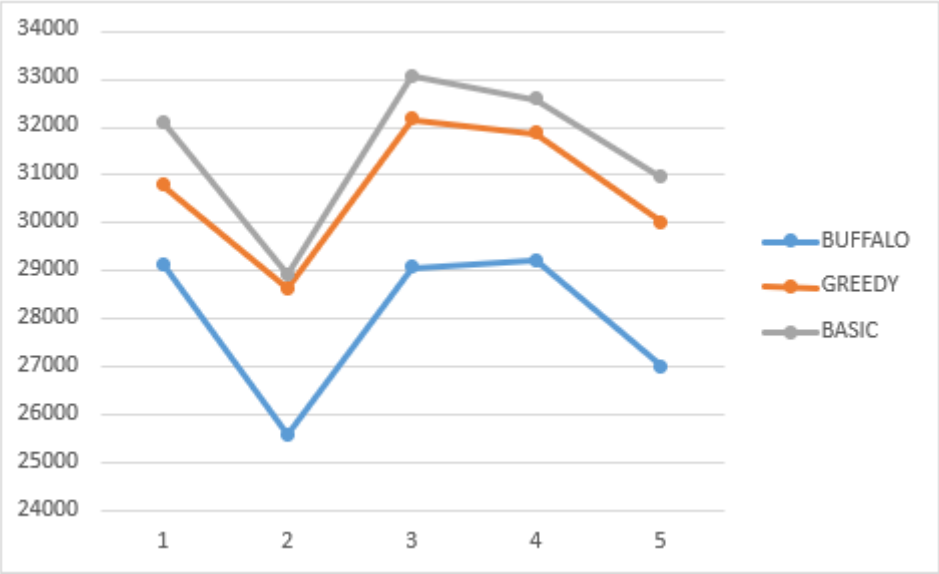


Slika 32. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 100_10_30_30

Izvor: izradio student

U ovoj kombinaciji ABO algoritam ima uvjerljivo najbolji rezultat. Sebičan algoritam ima lošiji rezultat, u 1. testnom skupu je gori od jednostavnog algoritma, te su dosta bliski rezultati u 4. testnom skupu. Jednostavan algoritam i dalje daje najgore rezultate.

Trideseta kombinacija ima 100 kamiona koji idu na maksimalno 10 dostavnih mjesta. Zadnja lokacija koja može bit uključena u dostavljačku rutu je označena brojem 30 (11 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 9:00 sati. Rezultati algoritama za svih 5 instanci testnog skupa 100_10_30_60 prikazani su na slici 33.

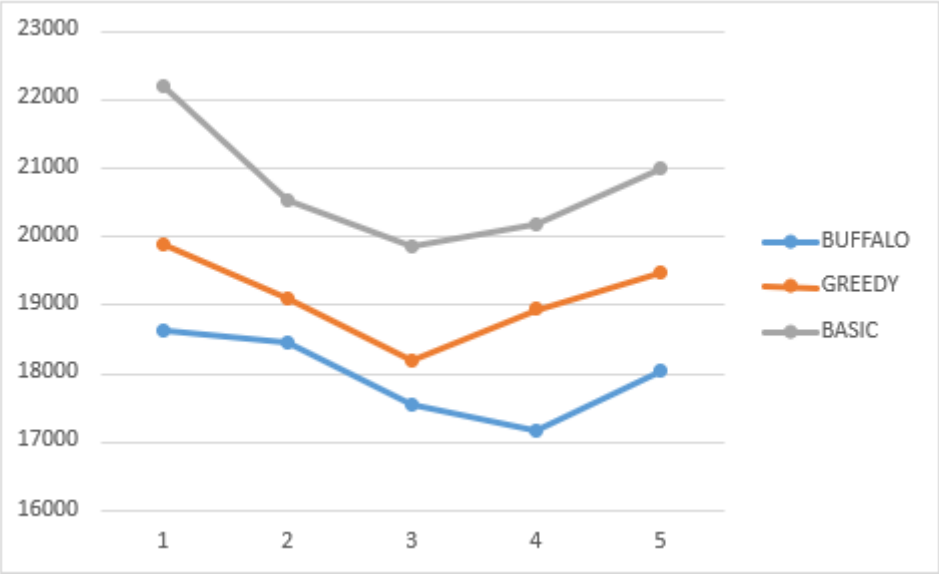


Slika 33. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 100_10_30_60

Izvor: izradio student

Isto kao i prije. U ovoj kombinaciji ABO algoritam daje najbolje rezultate. Sebičan algoritam daje lošije rezultate a jednostavan algoritam najgore, s tim što su u 2. testom skupu rezultati za sebičnog algoritma i jednostavnog algoritma jako bliski.

Trideset prva kombinacija ima 100 kamiona koji idu na maksimalno 10 dostavnih mjesta. Zadnja lokacija koja može bit uključena u dostavljačku rutu je označena brojem 40 (21 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 8:30 sati. Rezultati algoritama za svih 5 instanci testnog skupa 100_10_40_30 prikazani su na slici 34.

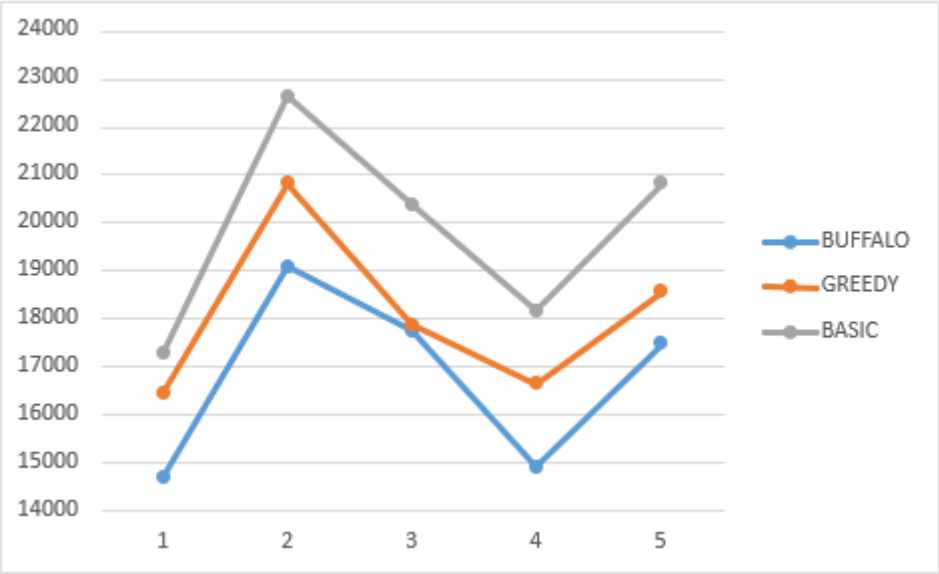


Slika 34. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 100_10_40_30

Izvor: izradio student

Jednako kao i prije. U ovoj kombinaciji ABO algoritam daje najbolje rezultate. Sebičan algoritam daje lošije rezultate a jednostavan algoritam najgore rezultate.

Trideset druga i zadnja kombinacija ima 100 kamiona koji idu na maksimalno 10 dostavnih mjesta. Zadnja lokacija koja može bit uključena u dostavljačku rutu je označena brojem 40 (21 lokacija ukupno), te svi kamioni imaju polazno vrijeme između 8:00 i 9:00 sati. Rezultati algoritama za svih 5 instanci testnog skupa 100_10_40_60 prikazani su na slici 35.



Slika 35. Ukupan broj minuta koja su dostavna vozila provela u gradskom središtu za svih 5 instanci testnog skupa 100_10_40_60

Izvor: izradio student

Na zadnjoj kombinaciji, ista je priča kao i do sada. ABO algoritam daje najbolje rezultate, s tim što mu je u 3. testnom skupu sebičan algoritam bio dosta blizu. Ali kao i do sada, sebičan algoritam daje lošije rezultate, te jednostavan algoritam daje najgore rezultate.

5.2. ANALIZA UČINKOVITOSTI ALGORITAMA

Nakon provedbe 160 testnih skupova na 3 različita algoritma. Jednostavan algoritam je davao najgore rezultate, što je i očekivano s obzirom na jednostavnost izvedbe algoritma. Sebičan algoritam je bio nešto bolji, ali je kvaliteta njegovog rezultata primarno ovisila o sadržaju testnog skupa, te bi davao najbolji rezultat jedino kad bi mu se potrefilo da je nasumično generiran testni skup omogućio da dobije najbolji rezultat. To se dešavalo češće kod jednostavnijih kombinacija sa 10 i 20 kamiona, ali kod 50 i 100 bi to bila rijetkost.

ABO algoritam je imao najbolja rješenja, pogotovo kod kompleksnijih kombinacija sa 50 i 100 kamiona kada su minute u tisućama. U slučajevima kad je ostvario drugo najbolje vrijeme (iza pohlepnog algoritma), uvijek je imao oko 1% devijacije od najboljeg rješenja. Dok je kod pohlepnog algoritma razlika bila mnogo veća.

ABO algoritam je davao najbolje rješenje u 87.5% slučajeva, što znači da je daleko najučinkovitiji od 3 algoritma. Još važnije je to što je davao najbolje rješenje ta kombinacije od

50 i 100 dostavnih kamiona, iz razloga što su to kombinacije koje su bliže stvarnom stanju broja dostavnih vozila u gradskom prometu.

5.2.1. Ograničenja i moguća poboljšanja

Unatoč odličnim rezultatima kombinacija za 50 i 100 dostavnih vozila, dogodilo se da u 12.5% slučajeva, specifično kod kasnijih kombinacija kod 10 i 20 dostavnih vozila, nije mogao pronaći jednako ili bolje vrijeme od sebičnog algoritma. Po može ukazivati na problem u kojem se algoritam preoptimizira, tj. nije sposoban istražiti daljnje bolje rješenje unutar 10000 iteracija. Budući da se testiralo sa nasumičnim testnim skupovima koji u sebi imaju raznolike kombinacije vremena polaska, redoslijeda i ukupnog broja lokacija te nasumičnom prirodom dodjeljivanja brojeva od 1-100 svim lokacijama da se odredi redoslijed polaska, moguće je da jednostavno nije poklopila dobra kombinacija nasumičnim putem. Naime, to je lako riješivši problem na način da se namještaju parametri za ukupni broj generacija ili ponavljanja dodjeljivanja nasumičnih brojeva u slučaju čestog ponavljanja istog trenutnog najboljeg rješenja.

Drugo ograničenje je hardverske prirode. Potrebna je poprilična dobra procesorska moć kako bi se programski dio algoritma izvodio u brzom ili normalnom vremenu. Tijekom procesa dobivanja rezultata testnih skupova, za 10000 generacija, bilo je potrebno između 60 i 90 minuta za jedan testni skup neke od kombinacija za 50 i 100 dostavnih vozila, a 45 i 60 minuta za kombinacije od 10 i 20 dostavnih vozila. Za rad u stvarnim uvjetima, gdje ima mnogo više varijabli, a samim time i generacija izvođenja radi što boljeg ukupnog rješenja može rezultirati još većim vremenom izvođenja programskog koda, tako da je potrebno uložiti u dobar stroj za skraćivanje vremena izvođenja programskog koda.

Jedno od mogućih poboljšanja trenutnog modela bi bilo u hibridiziranju ABO algoritma sa sebičnim algoritmom ili nekim drugim metaheurističkim algoritmom koji bi mogao dodatno povećati točnost rezultata ili pružiti alternativu kao brzo i dovoljno efikasno rješenje u slučaju nepredviđenih okolnosti prometu, te samim time i radu ABO algoritma.

5.2.2. Utjecaj na prometnu situaciju u gradu Rijeci

Implementacija optimizacijskih algoritama za upravljanje dostavnim vozilima u gradu Rijeci može značajno utjecati na smanjenje prometnih gužvi i poboljšanje protočnosti prometa. Korištenje ABO algoritma, koji se pokazao najefikasnijim, posebno kod većih skupova vozila, može dovesti do optimizacije ruta dostavnih vozila, smanjujući ukupno vrijeme putovanja i broj vozila na cestama u isto vrijeme.

S obzirom na složenost gradskog prometa, posebno u uvjetima visokog opterećenja, optimizacija kretanja dostavnih vozila mogla bi smanjiti prometne zastoje te pridonijeti smanjenju emisije štetnih plinova. Također, brže i učinkovitije dostave pozitivno bi utjecale na gospodarsku aktivnost u gradu.

Kombinacija ABO algoritma s dodatnim metodama optimizacije može pružiti fleksibilnost u stvarnim prometnim uvjetima, omogućujući brže prilagodbe promjenama u prometnim obrascima. Time bi se omogućila bolje upravljanje prometom, čime bi se smanjila vjerojatnost zastoja i povećala ukupna učinkovitost gradskog prometnog sustava.

6. ZAKLJUČAK

Ovaj rad se bavi optimizacijom dostavnih ruta u gradskim područjima, s posebnim fokusom na grad Rijeku. U radu su istražena tri različita algoritma za rješavanje problema optimizacije – jednostavan algoritam, sebičan algoritam i algoritam afričkog bizona. Sva tri algoritma su testirani uz pomoć generatora testnih skupova koji je generirao pet testnih skupova s nasumičnim vrijednostima za svaku kombinaciju ključnih parametra. Rezultati testiranja na različitim skupovima podataka pokazali su da ABO algoritam daje najefikasnija rješenja, posebno u kompleksnijim scenarijima s većim brojem dostavnih vozila. Jednostavan algoritam nije bio dovoljno učinkovit za složenije situacije, dok je sebičan algoritam pokazao određenu razinu uspješnosti, ali je njegova učinkovitost ovisila o specifičnostima testnih skupova. ABO algoritam se istaknuo kao najpouzdaniji, s najboljim rezultatima u 87.5% slučajeva, što ga čini najpogodnijim za primjenu u stvarnim uvjetima gradskog prometa. Unatoč postignutim rezultatima, identificirana su određena ograničenja, uključujući potrebu za visokim računalnim resursima i problem preoptimizacije u nekim slučajevima. Moguća poboljšanja uključuju hibridizaciju ABO algoritma s drugim metodama kako bi se dodatno poboljšala točnost i učinkovitost. Primjena ovih algoritama u stvarnim uvjetima, kao što su gradski prometni sustavi, mogla bi značajno smanjiti prometne gužve, optimizirati vrijeme dostave i smanjiti emisiju štetnih plinova, što bi imalo pozitivan utjecaj na kvalitetu života u gradu Rijeci. Uvođenje takvih tehnoloških rješenja predstavlja korak prema pametnijim i održivijim gradovima budućnosti.

LITERATURA

[1] B. L. Golden, A. A. Assad, L. Levy and F. G. Gheysens, “The feet size and mix vehicle routing problem”, Computers & Operations Research, Vol. 11, 1984, pp. 49–66.

[2] Toth, P., & Vigo, D. (2002). *The vehicle routing problem*. Society for Industrial and Applied Mathematics.

[3] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1), 269-271.

[4] Javaid, Muhammad Adeel, Understanding Dijkstra's Algorithm (April 10, 2013).

[5] Goldberg, Andrew V., and Tomasz Radzik. *A heuristic improvement of the Bellman-Ford algorithm*. Stanford, CA, USA:: Stanford University, Department of Computer Science, 1993.

[6] Baker, E., & Ayechew, M. A. (2003). A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, 30(5), 787-800.

[7] Odili, Julius. (2015). AFRICAN BUFFALO OPTIMIZATION ALGORITHM: A NEW METAHEURISTIC.

[8] M Gulić, M Žuškin, V Kvaternik - An Overview and Comparison of Selected State-of-the-Art Algorithms Inspired by Nature, TEM Journal, 2023

[9] Macready, W.G.: No Free Lunch Theorems for Optimization. IEEE Transactions on Evolutionary Computation 1(1), 67-82

[10] Koza, J. R. (1994). *Genetic programming: on the programming of computers by means of natural selection* (Vol. 1). MIT press.

[11] Internetski izvor za preuzimanje Apache NetBeans: https://netbeans.apache.org/front/main/download/

POPIS SLIKA