Metode za povećanje popunjenosti smještajnih kapaciteta

Marinčić, Domagoj

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

METODE ZA POVEĆANJE POPUNJENOSTI SMJEŠTAJNIH KAPACITETA

Završni rad

Domagoj Marinčić

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1P: Obrazac za ocjenu završnog rada na sveučilišnom prijediplomskom studiju

Ocjena završnog rada na sveučilišnom prijediplomskom studiju

Ime i prezime pristupnika:	Domagoj Marinčić
Studij, smjer:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. pristupnika, god.	R4536, 27.07.2020.
JMBAG:	0165086455
Mentor:	prof. dr. sc. Josip Job
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Metode za povećanje popunjenosti smještajnih kapaciteta
Znanstvena grana završnog rada:	**Obradba informacija (zn. polje računarstvo)**
Zadatak završnog rada:	Pristupnik treba implementirati testno okruženje za ispitivanje metoda za povećanje popunjenosti smještajnih kapaciteta te demonstrirati promjenu popunjenosti na nekoliko primjera. Tema rezervirana za: Domagoj Marinčić

Datum prijedloga ocjene završnog rada od strane mentora:	18.09.2024.
Prijedlog ocjene završnog rada od strane mentora:	Izvrstan (5)
Datum potvrde ocjene završnog rada od strane Odbora:	25.09.2024.
Ocjena završnog rada nakon obrane:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije završnog rada čime je pristupnik završio sveučilišni prijediplomski studij:	28.09.2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O IZVORNOSTI RADA

Osijek, 28.09.2024.

Ime i prezime Pristupnika:	Domagoj Marinčić
Studij:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	R4536, 27.07.2020.
Turnitin podudaranje [%]:	6

Ovom izjavom izjavljujem da je rad pod nazivom**: Metode za povećanje popunjenosti smještajnih kapaciteta**

izrađen pod vodstvom mentora prof. dr. sc. Josip Job

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis pristupnika:

SADRŽAJ

1. UVOD

U današnjem dinamičnom i konkurentnom turističkom tržištu, popunjenost smještajnih kapaciteta smatra se jednim od ključnih pokazatelja uspješnosti poslovanja u hotelskoj industriji. Povećanje popunjenosti direktno utječe na prihode, zadovoljstvo gostiju i održivi razvoj poslovanja na duže staze. Zato je od velike važnosti razvijati i primjenjivati učinkovite metode za optimizaciju popunjenosti smještajnih kapaciteta.

Ovaj rad fokusira se na implementaciju testnog okruženja za ispitivanje različitih metoda koje se koriste za povećanje popunjenosti smještajnih kapaciteta. Cilj je pokazati kako različite strategije mogu utjecati na popunjenost kroz nekoliko konkretnih primjera. Tako se osigurava bolje shvaćanje efikasnosti svake individualne metode te se pružaju smjernice za njihovu primjenu u stvarnim poslovnim situacijama.

Rad je strukturiran tako da prvo daje teoretski pregled popunjenosti smještajnih kapaciteta i sličnih metoda i algoritama koji se već koriste u industriji. Nakon toga slijedi opis metodologije i tehnologija koje će se koristiti za postavljanje i implementaciju testnog okruženja. U glavnom dijelu rada predstavljeni su rezultati testiranja nekoliko metoda na odabranim primjerima te analiza dobivenih rezultata.

Naposljetku, u zaključku se sažimaju ključni nalazi rada, ističu prednosti i nedostaci testiranih metoda te daju preporuke za buduće istraživanje i primjenu ovih metoda u praksi. Kroz ovaj rad, nastoji se pridonijeti boljem razumijevanju i optimizaciji popunjenosti smještajnih kapaciteta, što je ključno za uspjeh u hotelskoj industriji.

1.1. Zadatak završnog rada

Zadatak ovog završnog rada je osmisliti i implementirati algoritme koji će poslužiti za najučinkovitiji raspored rezervacije gostiju uzimajući u obzir njihove želje, mogućnosti i dostupnost. Kroz razvoj testnog okruženja, ispitivat će se valjanost i efikasnost različitih algoritama koji će omogućiti upravo taj optimalan raspored. Provedeno testiranje će demonstrirati promjene u popunjenosti kapaciteta na odabranim primjerima, dok će analiza rezultata pružiti smjernice za praktičnu primjenu i daljnja istraživanja.

2. PREGLED PODRUČJA TEME RADA

U ovom poglavlju analizirat ćemo neke od postojećih algoritam i metoda koje se koriste za što učinkovitije popunjavanje smještajnih kapaciteta. Promatranje ovih rješenja dati će nam nove poglede koji se mogu primijeniti u svrhe unaprjeđenja ovog rada.

Problem maksimalne popunjenosti (engl. *Maximum Occupancy Problem*), jedan je od najpopularnijih pristupa u optimizaciji popunjenosti smještajnih kapaciteta te se uglavnom rješava korištenjem algoritama za optimizaciju i raspoređivanje. Navedeni algoritmi uzimaju u obzir razne ulazne parametre kao što su duljina trajanja boravka, željeni datumi boravka, fleksibilnost gostiju i kapacitet smještajnih jedinica.

U situacijama u kojima postoji više različitih načina za razmještavanje gostiju koriste se algoritmi raspoređivanja kao što su Backtracking metode, Branch and Bound pristup ili Greedy algoritam koji će pronaći najoptimalnije rješenje za pojedinu situaciju. Ovi algoritmi će pomoći da se smještajni kapaciteti što bolje popune tako što će unaprijediti raspored gostiju prema njihovim preferencijama i dostupnosti smještaja.

Heuristički algoritmi. Naziv heuristički algoritam koristi se za opisivanje algoritma koji se bazira na heuristici. Takvi su algoritmi obično iterativni i koriste se za rješavanje kompleksnih optimizacijskih problema gdje klasični pristupi možda nisu dovoljno učinkoviti. Neki od heurističkih algoritama su Genetski algoritmi (engl. *Genetic Algorithms*), Simulirano kaljenje (engl. *Simulated Annealing*) i Algoritmi mravlje kolonije (engl. *Ant Colony Optimization*). Ovi algoritmi temeljeni su na prirodnim procesima i evoluciji te pružaju visokokvalitetna rješenja za problem popunjenosti smještajnih kapaciteta.

Algoritmi za linearnu i cjelobrojnu optimizaciju su također često korišteni pri rješavanju problema ovog tipa. Uporabom linearnih programskih modela moguće je definirati problem popunjenosti kapaciteta kao skup linearnih jednadžbi i nejednadžbi i tako pronaći najbolje rješenje koje uvećava popunjenost uz minimalne troškove.

Primjer konkretne primjene ovih algoritama može se vidjeti u slučaju kada imamo više grupa gostiju s različitim preferencijama za datume boravka. Na primjer, jedna obitelj želi rezervirati prvi vikend u srpnju, dok drugoj obitelji odgovara i prvi i drugi vikend u srpnju. Algoritam bi trebao rasporediti ove obitelji tako da prva obitelj dobije prvi vikend, a druga obitelj drugi vikend, čime se maksimizira ukupna popunjenost smještajnih kapaciteta.

Pregledom sličnih rješenja, zadatak ovog rada je identificirati najefikasnije algoritme i metode i prilagoditi ih svojstvenim potrebama smještajnih kapaciteta. Cilj je osmisliti i implementirati

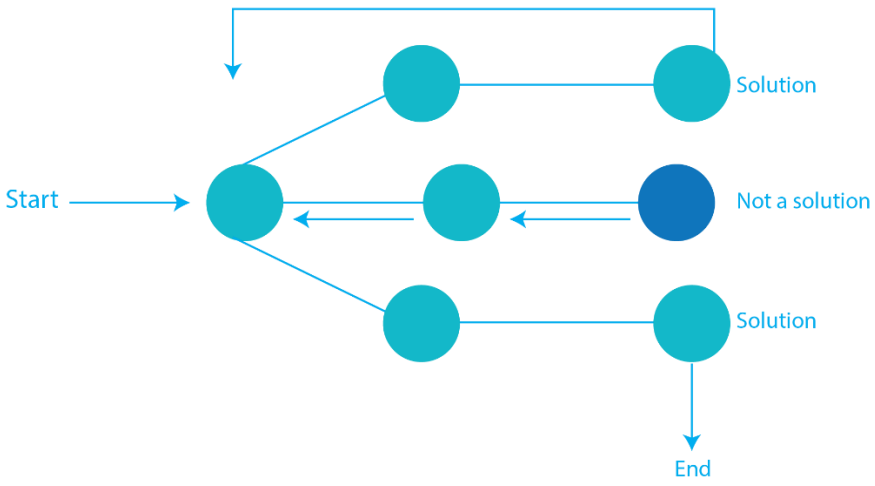
algoritme koji će omogućiti optimalno raspoređivanje gostiju, čime će se maksimizirati popunjenost i povećati učinkovitost upravljanja smještajnim kapacitetima.

2.1. Algoritmi za raspoređivanje

U procesu optimizacije raspoređivanja, posebice u domeni popunjenosti smještajnih kapaciteta, koristi se niz različitih algoritamskih pristupa. Ovisno o prirodi problema i dostupnim resursima, moguće je primijeniti metode koje pružaju optimalna ili aproksimativna rješenja. Izbor odgovarajućeg algoritma može značajno utjecati na efikasnost rješavanja problema i vrijeme potrebno za dobivanje rješenja, posebice u situacijama kada postoji veliki broj mogućih kombinacija. U nastavku su opisani neki od ključnih algoritama koji se koriste u svrhu raspoređivanja.

2.1.1. Backtracking algoritam

Backtracking je metoda koja se koristi za pronalaženje svih mogućih rješenja problema putem rekurzivnog pretraživanja. Termin „backtracking“ sugerira da je se potrebno vratiti i pokušati s drugim rješenjima ako trenutačno rješenje nije prikladno. Stoga se u ovom pristupu koristi rekurzija. Ovaj pristup koristi se za rješavanje problema koji imaju više rješenja [1]. U kontekstu popunjenosti smještajnih kapaciteta, backtracking algoritam može ispitivati sve moguće rasporede gostiju i odabrati onaj koji daje najveću popunjenost. Često je dobar za rješavanje manjih problema, no ukoliko se koristi za veće probleme može postati neizvediv zbog eksponencijalnog rasta mogućih rješenja.



Slika 2.1. Skica Backtracking algoritma

2.1.2. Branch and Bound algoritam

Branch and Bound je metoda koja, dodavanjem graničnih uvjeta koji služe za eliminiranje potproblema koji ne mogu dovesti do boljih rješenja od trenutnog najboljeg rješenja, povećava učinkovitost backtracking-a. Ovaj pristup može značajno smanjiti broj ispitivanih rješenja, te je stoga pogodniji za uporabu kada imamo veće probleme optimizacije popunjenosti.

2.1.3 Greedy algoritam

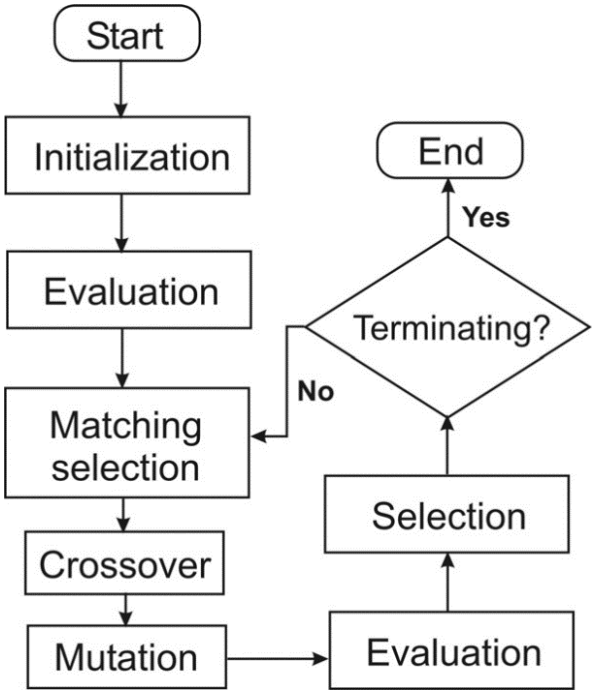
Greedy algoritmi rade na principu donošenja lokalno najpovoljnijih odluka u nadi da će to dovesti do globalno najdjelotvornijeg rješenja. Kada promatramo u okviru popunjenosti smještajnih kapaciteta, greedy algoritam bi mogao prioritetno rasporediti goste prema njihovim preferencijama datuma, odabirući uvijek onu opciju koja trenutno najviše povećava popunjenost. Iako greedy algoritmi ne garantiraju uvijek optimalno rješenje, u većini slučajeva daju dovoljno dobra rješenja u kratkom vremenu.

2.2. Heuristički algoritmi

Heuristički algoritmi su metode koje omogućuju pronalazak dobrih rješenja za složene probleme, kada je iscrpno pretraživanje svih mogućih rješenja neizvedivo. Umjesto traženja savršenog rješenja, ovi algoritmi koriste približne pristupe kako bi našli dovoljno dobre rezultate u kraćem vremenu. U nastavku su objašnjeni neki od ključnih heurističkih algoritama koji se koriste za optimizaciju popunjenosti smještajnih kapaciteta.

2.2.1. Genetski algoritam (engl. *Genetic Algorithm*)

Genetski algoritam je metoda za rješavanje kako ograničenih, tako i neograničenih optimizacijskih problema koja se temelji na prirodnoj selekciji, procesu koji pokreće biološku evoluciju [2]. Oni koriste populaciju mogućih rješenja koja evoluira kroz iteracije koristeći operatore poput selekcije, križanja (engl. *Crossover*) i mutacije. Sa strane gledišta popunjenosti smještajnih kapaciteta, genetički algoritam može generirati i evoluirati različite rasporede gostiju, tražeći optimalan raspored koji maksimizira popunjenost. Ovi algoritmi veoma su učinkoviti pri radu sa složenim problemima koji pri tome sadrže veliki broj varijabli.



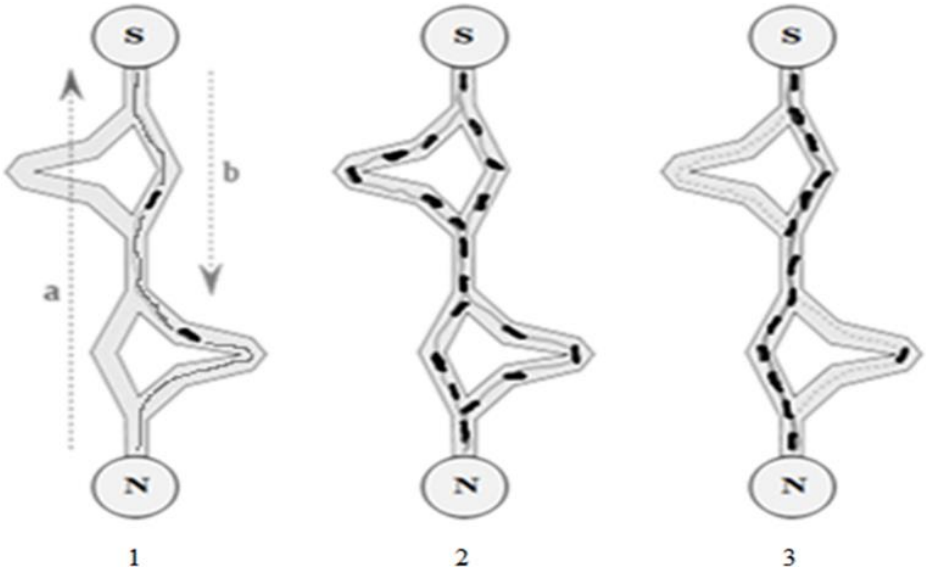
Slika 2.2. Generalni prikaz sheme genetskog algoritma

2.2.2. Simulirano kaljenje (engl. *Simulated Annealing*)

Simulirano kaljenje je heuristički algoritam inspiriran procesom oplemenjivanja metala koji se naziva kaljenje i koristi već 7000 godina. Proces se sastoji od 3 faze: zagrijavanje na visoku temperaturu, zadržavanje na toj temperaturi i zatim hlađenje. Ovaj pristup omogućava izlazak iz lokalnih optimuma u potrazi za globalnim optimumom. U optimizaciji popunjenosti, simulirano kaljenje može pomoći pronaći blizu optimalna rješenja čak i u složenim problemima.

2.2.3. Algoritam mravlje kolonije (engl. *Ant Colony Optimization*)

Algoritmi mravlje kolonije inspirirani su ponašanjem stvarnih mrava pri traženju hrane. Osnovna karakteristika kolektivnog ponašanja mrava je da svi članovi kolonije indirektno ili direktno razmjenjuju informacije o svom okruženju, tj. prisutan je fenomen kolektivne inteligencije. U kontekstu popunjenosti smještajnih kapaciteta, algoritam mravlje kolonije može učinkovito istraživati različite rasporede i iterativno poboljšavati rješenja kroz kolektivno ponašanje.



Slika 2.3. Skica algoritma mravlje kolonije

2.3. Algoritmi za linearnu i cjelobrojnu optimizaciju

Linearno i cjelobrojno programiranje koriste se za optimizaciju problema popunjenosti, pri čemu linearno radi s kontinuiranim, a cjelobrojno s diskretnim varijablama. Ove metode omogućuju učinkovito raspoređivanje gostiju i kapaciteta.

2.3.1. Linearno programiranje (engl. *Linear Programming*)

Linearno programiranje je matematička modelarska tehnika u kojoj se linearna funkcija maksimizira ili minimizira pod uvjetom različitih ograničenja [3]. Mnogi praktični problemi u operacijskim istraživanjima mogu se iskazati kao problemi linearnog programiranja. Kroz povijest su ideje iz područja linearnog programiranja pridonijele razvitku nekih od glavnih koncepata teorije optimizacije (dualnost, dekompozicija, važnost konveksnosti)

2.3.2. Cjelobrojno programiranje (engl. *Integer Programming*)

Cjelobrojno programiranje je varijanta linearnog programiranja gdje su sve ili neke varijable ograničene na cjelobrojne vrijednosti. Ovaj pristup je posebno koristan za probleme popunjenosti smještajnih kapaciteta gdje rasporedi i dodjele moraju biti cjelobrojne (npr. broj dana boravka, broj gostiju). Cjelobrojni modeli omogućuju preciznije modeliranje problema i pronalaženje optimalnih rješenja koja zadovoljavaju sve specifične zahtjeve.

3. ANALIZA ALGORITAMA

U ovom poglavlju dati ćemo uvid u ulazne podatke, što obuhvaćaju ulazni podatci, analizirat ćemo kreirane algoritme, objasniti njihov rad, funkcije koje se u njima koriste, postaviti neke pretpostavke o tome kakve rezultate možemo očekivati te na kraju prokomentirati same algoritme, njihove prednosti i nedostatke.

3.1. Pregled korištenih tehnologija

U ovom potpoglavlju opisane su tehnologije korištene prilikom izrade završnog rada. Korištenje odgovarajućih alata i tehnologija ključno je za osiguranje efikasnosti, skalabilnosti i točnosti razvijenih rješenja. Opisat ćemo programske jezike, razvojne okvire te sve potrebne resurse i alate koji su korišteni u procesu izrade projekta.

Za implementaciju algoritama i izradu testnog okruženja korišteni su moderni programski jezici poput Pythona, koji nudi širok spektar biblioteka i alata za znanstveno računanje, optimizaciju i analizu podataka. Također, opisat ćemo alate koji omogućuju prikaz i interpretaciju podataka na intuitivan i razumljiv način.

Razvoj testnog okruženja zahtijevao je korištenje odgovarajućih razvojnih okvira koji podržavaju brzu izradu prototipova i testiranje algoritama. Upotreba integriranih razvojnih okruženja (IDE) i sustava za kontrolu verzija bila je ključna za upravljanje razvojnim procesom i osiguranje kvalitete koda.

Uz navedene tehnologije, bit će obrađeni i alati koji su korišteni za testiranje učinkovitosti razvijenih algoritama u različitim scenarijima. Ovaj pregled pružit će detaljan uvid u tehnički aspekt projekta i omogućiti bolje razumijevanje korištenih tehnologija i njihovih prednosti.

3.1.1. Visual Studio Code

Visual Studio Code je lagan, ali snažan uređivač izvornog koda koji radi na računalu i dostupan je za Windows, macOS i Linux. Dolazi s ugrađenom podrškom za JavaScript, TypeScript i Node.js te ima bogat ekosustav proširenja za druge jezike i okruženja (kao što su C++, C#, Java, Python, PHP, Go, .NET) [4]. VS Code je jedan od najčešće korištenih alata za razvoj softvera u povijesti. Ova popularnost uglavnom je rezultat velikog ekosustava proširenja koje je pridonijela zajednica [5]. Visual Studio Code poznat je po brzoj izvedbi i mogućnosti rada na velikim projektima bez mnogo opterećenja na procesor ili memoriju. Također, pruža napredne značajke uređivanja koda kao što su preuređenje koda, automatsko dovršavanje koda, pretraživanje i zamjena teksta u kodu i brojne druge. Pri izradi ovog završnog rada Visual

Studio Code koristio je kao razvojno okruženje za izradu i prikaz rezultata rada analiziranih algoritama, te provjeru i testiranje učinkovitosti algoritama za raspoređivanje.

3.1.2 Programski jezik Python

Python je programski jezik visokog nivoa opće namjene. Python je dinamički tipiziran i koristi automatsko skupljanje smeća (garbage collection). Podržava različite programske paradigme, uključujući strukturirano (posebno proceduralno), objektno orijentirano i funkcionalno programiranje [6]. Kreirao ga je Guido van Rossum, a prvi put je objavljen 20. veljače 1991. Ime programskog jezika Python dolazi od stare BBC-jeve televizijske humoristične serije pod nazivom *Monty Python's Flying Circus* [7]. Danas je jedan od najpopularnijih programskih jezika zahvaljujući svojoj jednostavnosti i širokoj primjeni. Dolazi s velikom standardnom bibliotekom, a zbog ogromne popularnosti razvijen je i veliki broj vanjskih biblioteka kao što su Pandas i NumPy koje su također korištene za izradu algoritama opisanih u ovom radu.

3.1.3 Biblioteke

Biblioteke koje su korištene u procesu kreiranja algoritama su Pandas, NumPy, Random i DateTime.

Pandas:

Pandas je brz, snažan, fleksibilan i jednostavan alat otvorenog koda za analizu i manipulaciju podacima, izgrađen na temelju programskog jezika Python [8]. U sklopu ovog rada biblioteka pandas je korištena za organizaciju i manipulaciju podataka kroz DataFrame.

NumPy:

NumPy je temeljni paket za znanstveno računarstvo u Pythonu. To je Python biblioteka koja pruža višedimenzionalni objekt nizova, razne izvedene objekte (kao što su maskirani nizovi i matrice), te niz rutina za brze operacije nad nizovima, uključujući matematičke, logičke operacije, manipulaciju oblikom, sortiranje, odabir i još mnogo toga [9].

Random:

Ova biblioteka omogućava rad i generiranje pseudo-slučajnih brojeva te odabir slučajnih elemenata iz sekvenci. Također podržava razne metode za rad sa slučajnim vrijednostima poput

generiranja cijelih brojeva, decimalnih brojeva ili odabira elemenata iz liste. Neke od glavnih funkcija random biblioteke su: random.random(), random.randint(a, b), random.choice(seq), random.shuffle(seq) i brojne druge.

DateTime:

Modul DateTime u Pythonu nudi klase za manipulaciju datumima i vremenima. Iako je aritmetika s datumima i vremenima podržana, fokus implementacije je na učinkovitoj ekstrakciji atributa za oblikovanje izlaza i manipulaciju podacima.

3.2. Formulacija ulaznih podataka

Ulazni podatci koje predajemo algoritmima na obradu i korištenje organizirani su u obliku CSV datoteka. Za potrebe ovog rada definirana su 3 skupa podataka. Ti skupovi sadržavaju osnovne informacije koje će algoritmima poslužiti kako bi kasnije uspješno mogli raspoređivati rezervacije. Te informacije uključuju redni broj gosta (npr. „Guest_6“), željeni početni datum rezervacije za svakog gosta, krajnji datum rezervacije, ukupna duljina boravka te jedna boolean vrijednost koja predstavlja fleksibilnost gosta, odnosno je li gost fleksibilan za dodjeljivanje nekog drugog datuma ukoliko je njegov željeni termin boravka zauzet. Kao što ćemo kasnije vidjeti neki algoritmi uopće neće uzimati u obzir fleksibilnost gosta u svojim izračunima dok drugi hoće. Za potrebe ovog rada definirana su 3 skupa podataka. Prvi skup se sastoji od 10 primjera s unaprijed definiranim podatcima, drugi skup od 30 primjera i treći skup od 100 primjera. Svrha je utvrditi kako algoritmi reagiraju na različitu veličinu skupova. Prilikom rada s prvim skupom podataka rezervacije se raspoređuju unutar jednog mjeseca. Kod rada s drugim skupom raspoređuju se unutar dva mjeseca, kod trećeg skupa unutar tri mjeseca. Trajanje boravka je određeno odabirom broja između 1 i 7, dakle vrijednost 7 je maksimalnu duljina boravka za jednog gosta. Na kraju, fleksibilnost je ulazni podatak koji poprima vrijednosti true ili false. Svi navedeni podatci integriranu su u CSV datotekama koje omogućavaju jednostavnu i učinkovitu organizaciju, te su osnovni podatkovni izvor za primjenu razvijenih algoritama. Ulazni podatci se učitavaju iz datoteka te se onda prosljeđuju algoritmima na dalju obradu. Na temelju ovih podataka algoritmi za raspored rezervacija analiziraju različite pristupe za optimizaciju popunjenosti smještajnih kapaciteta.

3.3. Ispis rezultata

Nakon što su podaci učitani, predaju se algoritmima na korištenje. Po završetku korištenja, algoritmi vraćaju obrađene podatke koji se spremaju u varijablu te ispisuju na ekran kako bi se vidjelo na koji je način algoritam analizirao podatke, tj. u našem slučaju kako su raspoređene rezervacije. Ukoliko je rezervacija odobrena, to jest ako je algoritam uspješno pronašao datum za rezervaciju, onda ćemo za gosta ispisati poruku o uspješnoj rezervaciji koja sadržava redni broj gosta te početni i krajnji datum boravka. Tako imamo pregledan uvid kako su raspoređene rezervacije i koliko su algoritmi bili učinkoviti. U slučaju kada za gosta nije uspješno pronađen niti jedan odgovarajući termin boravka ispisuje se poruka o neuspješnoj rezervaciji. Ovo pomaže u razumijevanju gdje algoritam nije bio efikasan odnosno gdje nije mogao zadovoljiti zahtjev. S pomoću tih informacija može se dalje raditi na poboljšanju algoritma.

```
1   Guest,Start day,End day,Stay duration,Flexibility
2   Guest_1,2024-07-05,2024-07-10,5,True
3   Guest_2,2024-07-15,2024-07-18,3,False
4   Guest_3,2024-07-20,2024-07-22,2,True
5   Guest_4,2024-07-01,2024-07-06,5,False
6   Guest_5,2024-07-07,2024-07-08,1,True
7   Guest_6,2024-07-12,2024-07-18,6,False
8   Guest_7,2024-07-22,2024-07-26,4,True
9   Guest_8,2024-07-03,2024-07-06,3,False
10  Guest_9,2024-07-12,2024-07-14,2,True
11  Guest_10,2024-07-17,2024-07-21,4,True
12
```

Slika 3.1. Primjer ulaznih podataka

3.4. Zajedničke funkcije

Izrada algoritama za efikasno raspoređivanje rezervacija podrazumijeva i korištene prikladnih funkcija. Jedna od takvih funkcija koja se pojavljuje u gotovo svim kreiranim algoritmima jest funkcija all_days_free(). Funkcija kao ulazne parametre uzima početni dan boravka, trajanje boravka te listu u kojoj se spremaju zauzeti dani. Cilj funkcije je provjera jesu li željeni dani slobodni ili zauzeti za rezervaciju.

```
def all_days_free(start_day, duration, occupied_days):
    for day in range(int(duration)):
        current_day = start_day + timedelta(days=day)
        if current_day in occupied_days:
            return False
    return True
```

Programski kod 3.1. Funkcija za pronalazak slobodnih dana

3.5. Algoritmi za raspoređivanje na temelju redoslijeda dospijeća zahtjeva

Prva skupina algoritama koja se predstavlja jest skupina algoritama koji raspoređuju pristigle rezervacije prema redoslijedu zaprimanja. U nastavku su analizirani algoritmi koji imaju jednak uvjet prema kojemu razvrstavaju rezervacije, a razlikuju se po tome uzima li se u obzir fleksibilnost gosta za dodjeljivanje drugog datuma ukoliko prvotni nije raspoloživ.

Prvi algoritam koji ćemo analizirati nazvan je 'first_come_first_served_algorithm'. Implementira jednostavnu logiku dodjeljivanja rezervacija smještajnog kapaciteta. Osnovni cilj ovog algoritma jest raspored rezervacija po redoslijedu dospijeća. To znači da algoritam raspoređuje rezervacije onako kako one pristižu, ne uzimajući u obzir nikakve dodatne varijable kao što je na primjer duljina trajanja boravka. Pretpostavka jest da ovaj algoritam nije izuzetno efikasan upravo zato što u svojim izračunima ne uzima dovoljno varijabli u obzir.

```
def first_come_first_served_algorithm(df, start_date, end_date):
    schedule = {}
    occupied_days = set()

    for index, guest in df.iterrows():
        start_day = guest['Start day']
        duration = guest['Stay duration']

        all_days_free = True
        guest_days = []
        for i in range(int(duration)):
            current_day = start_day + timedelta(days=i)
            guest_days.append(current_day)

            if current_day > end_date or current_day in occupied_days:
                all_days_free = False
                break

        if all_days_free:
            for day in guest_days:
                occupied_days.add(day)
            schedule[guest['Guest']] = start_day
        else:
            schedule[guest['Guest']] = None

    return schedule
```

Programski kod 3.2. 'first_come_first_served_algorithm'

Ključni dijelovi algoritma:

1. Ulazni parametri:
    - 'df': Skup koji sadrži podatke o gostima, te duljini trajanju boravka i početnom danu rezervacije
    - 'start_date': Prvi dan od kojeg se može rezervirati smještaj
    - 'end_date': Zadnji dan do kojeg se može rezervirati smještaj
2. Inicijalizacija:
    - 'schedule': Praćenje dodijeljenih rezervacija, gdje je ključ ime gosta, a vrijednost ili početni dan boravka ili „None“ ako rezervacija nije moguća.
    - 'occupied_days': Set koji sadržava dane koji su već zauzeti, služi za praćenje dostupnosti.
3. Iteracija kroz podatkovni skup:

- Algoritam prolazi kroz svaki element skupa koristeći 'df.iterrows()' te dohvaća početni dan i trajanje boravka za svakog gosta.

4. Provjera zauzetosti:
    - Za svaki dan u trajanju gostovog boravka, algoritam provjerava dostupnost smještajnog kapaciteta. Ako on nije dostupan na bilo koji dan algoritam prekida provjeru i označava da su dani zauzeti.
5. Dodjela rezervacije:
    - Ako su svi dani slobodni, algoritam dodjeljuje rezervaciju danom gostu a svi dani koje je gost odabrao za boravak dodaju se u varijablu 'occupied_days', kako bi ti dani bili označeni u budućim provjerama zauzetosti.
    - Ako bilo koji dan unutar željenog perioda boravka nije dostupan, algoritam neće dodijeliti rezervaciju gostu, a u raspored se zapisuje „None“ za tog gosta
6. Povratak rezultata:
    - Konačno, funkcija vraća 'schedule', odnosno raspored u kojemu je sadržan popis gostiju i njihovi rezervirani dani.

```
Guest_1 je rezerviran od 07-05 do 07-10
Guest_2 je rezerviran od 07-15 do 07-18
Guest_3 je rezerviran od 07-20 do 07-22
Guest_4 nije dobio rezervaciju
Guest_5 nije dobio rezervaciju
Guest_6 nije dobio rezervaciju
Guest_7 je rezerviran od 07-22 do 07-26
Guest_8 nije dobio rezervaciju
Guest_9 je rezerviran od 07-12 do 07-14
Guest_10 nije dobio rezervaciju
```

Slika 3.2. Primjer ispisa rezultata algoritma

Na slici 4.4 vidimo ispis rezultata za ulazne podatke iz prvog podatkovnog skupa (Slika 4.1.). Jasno se vidi da gosti koji su na vrhu popisa, odnosno čije zahtjeve algoritam prvo provjerava, imaju prioritet pri dodjeli rezervacija u odnosu na goste pri dnu popisa. Tako uočavamo u navedenom primjeru na slici da je 'Guest_1' dobio prednost pri dodjeljivanju rezervacije u odnosu na 'Guest_4'. Rezervacije im se poklapaju u određenim danima ali je algoritam prednost dao prvom zbog toga što je se njegov zahtjev obradio ranije. Tu se uočava potencijalni problem ovog algoritma i prostor gdje bi se moglo raditi na poboljšanju.

Prednosti ovog algoritma su jednostavnost i jasnoća. Jednostavan način dodijele rezervacija po redoslijedu prijava, s jasnim pravilima za odbijanje kada su željeni datumi nedostupni. Nedostatak je taj što se smještajni kapacitet ne popunjava optimalno. Velika je mogućnost da gosti s dužim periodom boravka budu odbijeni iako bi kapaciteti bili popunjeniji.

Ovaj algoritam moguće je učiniti značajno efikasnijim uzimanje u razmatranje fleksibilnost gosta. Do sada je algoritam gledao sam vrijeme dospijeća zahtjeva, no ukoliko bi dodali opciju prema kojoj se gleda i fleksibilnost gostiju to bi uvelike pridonijelo povećanju ukupne popunjenosti jer bi omogućilo dodjelu rezervacija nekim gostima kojima je ona prvotno bila odbijena. Naravno, i ovaj algoritam možda neće udovoljiti zahtjevima svih gostiju, međutim pretpostavka je da će se ukupna popunjenost smještajnog kapaciteta povećati.

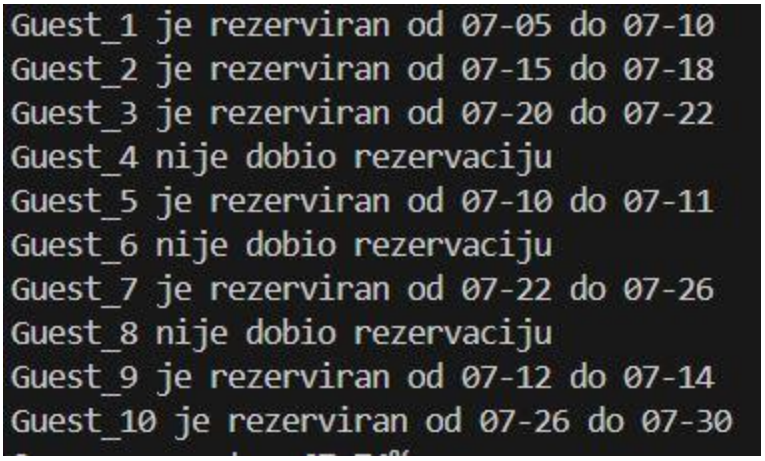
```
def flexible_reservation_algorithm(df, start_date, end_date):
    schedule = {}
    occupied_days = set()

    for index, guest in df.iterrows():
        start_day = guest['Start day']
        duration = guest['Stay duration']
        flexible = guest['Flexibility']

        if check_free_days(start_day, duration, occupied_days, start_date,
end_date):
            for day in range(int(duration)):
                current_day = start_day + timedelta(days=day)
                occupied_days.add(current_day.date())
            schedule[guest['Guest']] = start_day
        elif flexible:
            available_start_date = find_alternate_start_date(start_day,
duration, occupied_days, start_date, end_date)
            if available_start_date:
                for day in range(int(duration)):
                    occupied_days.add(available_start_date +
timedelta(days=day))
                schedule[guest['Guest']] = available_start_date
            else:
                schedule[guest['Guest']] = None
        else:
            schedule[guest['Guest']] = None

    return schedule
```

Programski kod 3.3. Prikaz flexible_reservation_algorithm-a

Na Programskom kodu 3.3. je prikazana poboljšana verzija prethodnog algoritma. Dodatak je provjera fleksibilnosti za goste. Ako je gost fleksibilan, a njegov željeni datum rezervacije nije dostupan, izvodi se funkcija 'find_alternate_start_date()', koja traži moguće datume s istom duljinom boravka koje bi mogla dodijeliti gostu. Ukoliko uspješno pronađe datume vraća vrijednost True i gostu se dodjeljuje rezervacija, u suprotnom gost neće dobiti rezervaciju.

```
def find_alternate_start_date(original_start_day, duration, occupied_days,
start_date, end_date):
    for day in range((end_date - start_date).days + 1):
        candidate_start_day = start_date + timedelta(days=day)
        if candidate_start_day >= original_start_day and
check_free_days(candidate_start_day, duration, occupied_days, start_date,
end_date):
            return candidate_start_day
    return None
```

Programski kod 3.4. Funkcija za provjeru slobodnih dana



Slika 3.3. Primjer ispisa rezultata algoritma

Na slici 4.7. je prikazan jedan primjer rezultata izvođenja algoritma za ulazne podatke iz prvog podatkovnog skupa. Primjećuje se da 'Guest_10' nije dobio željenu rezervaciju, no s obzirom na to da je fleksibilan dodijeljeni su mu drugi datumi, što ne bi bio slučaj u prethodnom algoritmu. Na taj način povećala se ukupna popunjenost smještajnog kapaciteta i prema tome algoritam se može okarakterizirati efikasnijim u odnosu na prethodni.

3.6. Algoritmi za raspoređivanje na temelju duljine trajanja boravka

U ovom poglavlju predstavit ćemo i analizirati drugačiju skupinu algoritama, algoritme koji raspoređuju pristigle rezervacije gledajući duljinu trajanja boravka kao prioritetni podatak. Prvi algoritam iz te skupine koji se razmatra nazvan je greedy_reservation_alhorithm jer se temelji na „pohlepnom“ pristupu. Za razliku od prve skupine algoritama koji sve svoje rasporede temelje na redoslijedu zaprimanja rezervacija, je 'greedy_reservation_algorithm' kao glavni uvjet prilikom raspoređivanja uzima duljinu trajanja boravka. Prema tome gosti koji svoj smještaj rezerviraju na dulji vremenski period imat će prednost u odnosu na one goste koji rezerviraju na kraće vrijeme. Ovaj algoritam ne uzima u obzir fleksibilnost gosta pri dodjeli rezervacija kao ni prvi algoritam iz prve skupine, što smanjuje učinkovitost. Pretpostavka je da je ovaj algoritam efikasniji od prvog algoritma jer prednost daje gostima s dužim periodom boravka što omogućuje da se najdulje rezervacije prve osiguraju, a potom se gostima s kraćim rezervacijama dodjeljuju preostali dani.

```
def greedy_reservation_algorithm(df, start_date, end_date):
    schedule = {}
    occupied_days = set()

    df_sorted = df.sort_values(by='Stay duration', ascending=False)

    for index, guest in df_sorted.iterrows():
        start_day = guest['Start day']
        duration = guest['Stay duration']

        if start_day < start_date or start_day > end_date:
            schedule[guest['Guest']] = None
            continue

        all_days_free = True
        for day in range(int(duration)):
            current_day = start_day + timedelta(days=day)
            if current_day in occupied_days or current_day > end_date:
                all_days_free = False
                break

        if all_days_free:
            for day in range(int(duration)):
                occupied_days.add(start_day + timedelta(days=day))
            schedule[guest['Guest']] = start_day
        else:
            schedule[guest['Guest']] = None

    return schedule
```

Programski kod 3.5. Prikaz algoritma

Ulazni podatci za algoritam dolaze u obliku tablice koja sadrži sljedeće kolone:

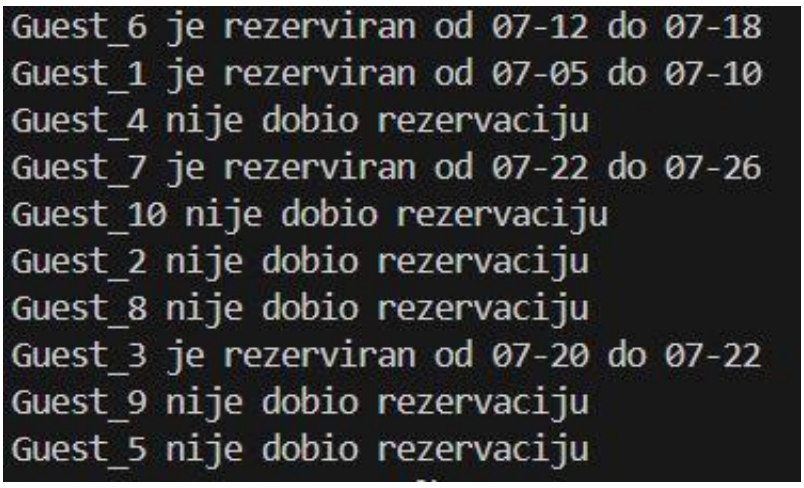
- Gost: identifikacija gosta
- Početni dan: datum početnog dana rezervacije
- Trajanje boravka: duljina trajanja rezervacije

Algoritam radi sa skupom već zauzetih dana('occupied_days'), koji prati dane koje su prethodni gosti popunili i koji provjerava dostupne dane.

U sklopu algoritma koristi se funkcija 'all_days_free()' koja je zajednička funkcija za većinu algoritama i koja je opisana ranije u tekstu. Glavna funkcija 'greedy_reservation_algorithm-a' raspoređuje rezervacije oslanjajući na takozvani pohlepni pristup, pri čemu preferira gosta s duljim vremenom trajanja rezervacije kako bi maksimalno popunila smještaj. Njeni koraci su:

- Sortiranje pristiglih zahtjeva: sve rezervacije se sortiraju prema duljini trajanja boravka od duljih rezervacija prema kraćim.
- Provjera slobodnih dana: nakon sortiranja, prolazi se kroz svaki pristigli zahtjev te se provjerava jesu li dani unutar željenog perioda slobodni.
- Dodjela rezervacije:
    - ukoliko je drugi korak uspješno proveden i svi željeni dani su dostupni gostu se dodjeljuje rezervacija,
    - ako nema slobodnih dana, gost neće dobiti rezervaciju (dodjeljuje se „None“)

Po završetku svih ovih koraka algoritam vraća raspored u kojemu su sadržane sve odobrene i odbijene rezervacije.



Slika 3.4. Primjer ispisa rezultata algoritma

Na slici vidimo ispis rezultata dobivenih nakon pokretanja opisanog algoritma za podatke iz prvog podatkovnog skupa. Primjećuje se da prvu rezervaciju dobiva 'Guest_6', a razlog za to je taj što njegova rezervacija ima najdulje trajanje. Prema tome algoritam prvu rezervaciju dodjeljuje njemu unatoč činjenici da 'Guest_6' nije prvi čiji je zahtjev algoritam obradio. Ako se pogleda dalje uočavamo da algoritam ostaje dosljedan te nastavlja davati prednost onim gostima koji imaju dulje trajanje rezervacije. Ukoliko dva ili više gostiju imaju jednaku duljinu boravka, onda se tek redoslijed zaprimanja zahtjeva dovodi u razmatranje.

Neke od ključnih prednosti ovog algoritma su jednostavnost i efikasnost. Algoritam je temeljen na pohlepnom pristupu zbog čega omogućava bolje i efikasnije popunjavanje u odnosu na prethodni algoritam. Algoritam daje prednost gostima s duljim vremenom boravka kako bi maksimizirao popunjenost. Relativno je jednostavan jer za svakog gosta odlučuje o rezervaciji bez potrebe za vraćanjem na prethodne odluke. Također, može vrlo brzo donijeti odluke o raspodjeli rezervacija, no možda neće uvijek dati najbolje rješenje za popunjenost svih dana.

Što se tiče ograničenja ovog algoritam, nedostatak je taj što se donose lokalno najbolje odluke, bez da se procjenjuje globalni optimum. Ovo može dovesti do neiskorištenih dana u nekim scenarijima. Drugi nedostatak je taj što se fleksibilnost ne uzima u obzir i time se gubi još jedan način maksimizacije popunjenosti.

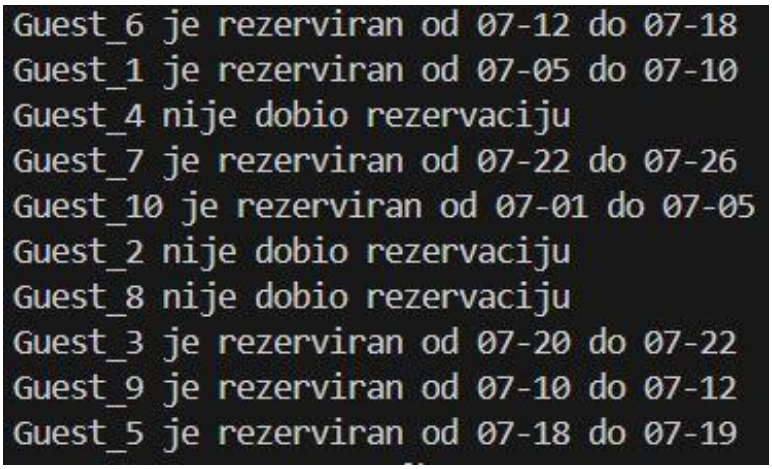
Sada ćemo ispitati efikasniju verziju prethodnog algoritma koji je nazvan 'greedy_flexible_reservation_algorithm', algoritam se temelji na istim principima kao i prethodni algoritam uz jedan dodatak, a to je da pri dodjeljivanju rezervacija u obzir uzima fleksibilnost gostiju. Dakle, ukoliko traženi vremenski raspon rezervacije nije dostupan ispituje se fleksibilnost gosta. Ako gost nije fleksibilan odnosno ne odgovara mu nijedan drugi datum osim traženog, neće dobiti rezervaciju. Međutim ukoliko je gost fleksibilan onda algoritam provjerava preostale slobodne dane u skupu zauzetih dana i pokušava pronaći alternativne datume, uz istu duljinu boravka, koje bi mogao dodijeliti gostu.

```
def greedy_flexible_reservation_algorithm(df, start_date, end_date):
    schedule = {}
    occupied_days = set()
    df_sorted = df.sort_values(by='Stay duration', ascending=False)

    for index, guest in df_sorted.iterrows():
        start_day = guest['Start day']
        duration = guest['Stay duration']
        flexible = guest['Flexibility']

        if start_day < start_date or start_day > end_date:
            schedule[guest['Guest']] = None
            continue
        if all_days_free(start_day, duration, occupied_days):
            for day in range(int(duration)):
                occupied_days.add(start_day + timedelta(days=day))
            schedule[guest['Guest']] = start_day
        elif flexible:
            available_start_date = None
            for day in range((end_date - start_date).days + 1):
                available_start_day = start_date + timedelta(days=day)
                if available_start_day + timedelta(days=int(duration) - 1) >
end_date:
                    break
                if all_days_free(available_start_day, duration,
occupied_days):
                    available_start_date = available_start_day
                    for d in range(int(duration)):
                        occupied_days.add(available_start_day +
timedelta(days=d))
                    schedule[guest['Guest']] = available_start_date
                    break

            if available_start_date is None:
                schedule[guest['Guest']] = None
        else:
            schedule[guest['Guest']] = None

    return schedule
```

Programski kod 3.6. Prikaz algoritma

Dodatak ovom algoritmu u odnosu na prethodni jest postupak provjere preostalih slobodnih dana koji se mogu dodijeliti gostima ukoliko su oni označeni kao fleksibilni unutar datoteka koje predstavljaju skup ulaznih vrijednosti. Funkcija 'all_days_free()' traži slobodan datum

unutar zadanog mjeseca koji može zadovoljiti željenu duljinu boravka. Ako pronađe odgovarajući period, funkcija vraća početni datum tog perioda, u suprotnom vraća se „None“.



Slika 3.5. Primjer ispisa rezultata algoritma

Slika 3.5. predstavlja rezultate raspoređivanja rezervacija od strane 'greedy_flexible_reservation_algorithm-a'. Kao i u prethodnom primjeru, vidimo da se prednost daje gostima koji imaju najveće vrijednosti za trajanje boravka, no nakon toga se provjerava fleksibilnost gosta. Stoga se za osobu 'Guest_5' ipak dodjeljuje rezervaciju, iako ne u željenom periodu.

Samim time što ovaj algoritam uzima u obzir fleksibilnost gostiju, to ga čini značajno efikasnijim u odnosu ne prethodnu verziju algoritma. Algoritam omogućava bolje iskorištavanje dostupnih resursa i može uvelike poboljšati popunjenost smještaja što ga čini boljim izborom za složenije probleme raspodjele rezervacija.

Posljednji predstavljeni algoritam omogućava dodjelu rezervacija i onim gostima čiji su željeni datumi bili zauzeti. Ali algoritam potom traži bilo koje dostupne datume, u mjesecu pazeći pri tome na trajanje boravka. To često rezultira uspješnom pronalasku slobodnih datuma, no postoji mogućnost da je vremenska razlika između početnog željenog datuma i konačnog, dodijeljenog datuma jako velika. Tu u praksi čini ovaj algoritam jako efikasnim, no u stvarnosti gosti možda ipak neće biti fleksibilni na tako velike razlike u datumima. U sljedećim stranicama analizirat će se algoritam koji bi mogao ponuditi rješenje na takav problem. Algoritam je nazvan 'greedy_offset_reservation_algorithm'. Kao i ostali algoritmi iz ove skupine i ovaj algoritam se zasniva na vremenu trajanja boravka kao glavnom uvjetu pri dodjeli rezervacija. Ono što ga razlikuje od prethodnog algoritma jest način na koji se manifestira fleksibilnost gostiju. Umjesto da se za goste koji su naznačeni kao fleksibilni traži bilo koji

dostupni datum u mjesecu koji zadovoljava traženu duljinu boravka, algoritam provjerava postoji li mogućnost dodijele rezervacije ukoliko bi ona započinjala 1 ili 2 dana ranije od željenog početnog datuma ili 1 ili 2 dana kasnije od istog datuma. Ovakav pristup donosi rješavanje problema raspoređivanja rezervacija koji bi bio korisniji u stvarnosti.

```
def greedy_offset_reservation_algorithm(df, start_date, end_date):
    schedule = {}
    occupied_days = set()
    shifted_reservations = 0

    df_sorted = df.sort_values(by='Stay duration', ascending=False)

    for index, guest in df_sorted.iterrows():
        start_day = guest['Start day']
        duration = guest['Stay duration']
        flexible = guest['Flexibility']

        if all_days_free(start_day, duration, occupied_days):
            new_start_day = start_day
        elif flexible:
            new_start_day = find_flexible_days(start_day, duration,
occupied_days, start_date, end_date)
            if new_start_day != start_day and new_start_day is not None:
                shifted_reservations += 1
        else:
            new_start_day = None

        if new_start_day:
            for day in range(int(duration)):
                occupied_days.add(new_start_day + timedelta(days=day))
            schedule[guest['Guest']] = new_start_day
        else:
            schedule[guest['Guest']] = None

    return schedule, shifted_reservations
```

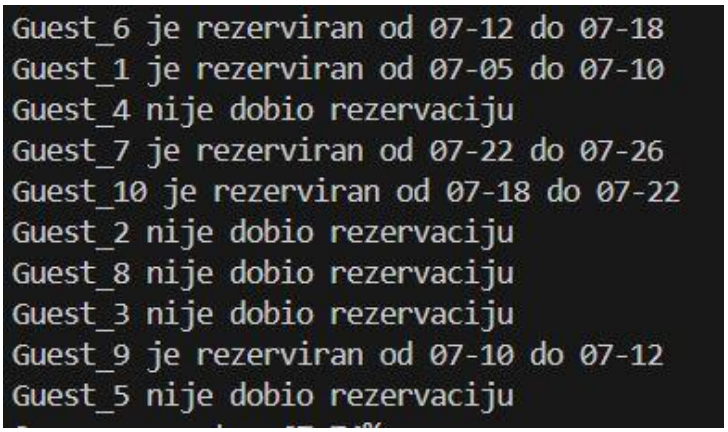
Programski kod 3.7. Prikaz algoritma

Ukoliko je gost fleksibilan izvodi se funkcija 'find_flexible_days()'.

```python
def find_flexible_days(start_day, duration, occupied_days, max_attempts=2):

    for offset in range(-max_attempts, max_attempts + 1):
        new_start_day = start_day + timedelta(days=offset)
        if all_days_free(new_start_day, duration, occupied_days):
            return new_start_day
    return None
```

Programski kod 3.8. Funkcija za pronalazak slobodnih dana unutar raspona

Funkcija provjerava dostupnost unutar dva dana prije i dva dana poslije željenog početnog dana.



Slika 3.6. Primjer ispisa rezultata algoritma

Slika 3.6. prikazuje primjer rezultata algoritma. Primjećuje se veliki broj odbijenih rezervacija. To ukazuje na manju efikasnost ovog načina raspodjele rezervacija, no ipak cilj ovog algoritma je prikazati jedan pristup rješavanju problema koji bi možda bio korisniji u stvarnosti u odnosu na prethodne algoritme.

4. TESTIRANJE ALGORITAMA

U ovom poglavlju predstavljaju se rezultati testova sprovedenih nad algoritmima koji su opisani u ovom završnom radu. Ovi testovi uključuju provjere kao što su postotak zauzetosti smještaja, prosječan broj odbijenih rezervacija i vrijeme izvođenja algoritama.

4.1. Funkcije za ispitivanje efikasnosti

Za potrebe izračuna efikasnosti i usporedbe algoritama izrađene su funkcije prema kojima ćemo vrednovati algoritme.

```
def calculate_occupancy_rate(zauzeti_dani, total_days=61):
    return len(zauzeti_dani) / total_days

def get_occupied_days(reservation_schedule, df):
    occupied_days = set()

    for guest, start_date in reservation_schedule.items():
        if start_date is not None:
            # Pronađi trajanje boravka iz DataFrame-a
            duration = df.loc[df['Guest'] == guest, 'Stay duration'].values[0]
            # Dodaj sve dane od početka rezervacije do kraja trajanja
            for day in range(int(duration)):
                occupied_days.add(start_date + timedelta(days=day))

    return occupied_days
```

Programski kod 4.1. Funkcija za provjeru postotne popunjenosti

Prva takva funkcija je 'calculate_occupancy_rate()', prikazana na slici. Ova funkcija radi na jednostavnom principu, a to jest, uzima sve dane koje je algoritam uspio dodijeliti gostima i dijeli ga s ukupnim brojem dana u mjesecu.

```
def number_of_rejected_reservations(reservation_schedule):
    rejected_reservations = sum(1 for guest, date in
reservation_schedule.items() if date is None)
    return rejected_reservations
```

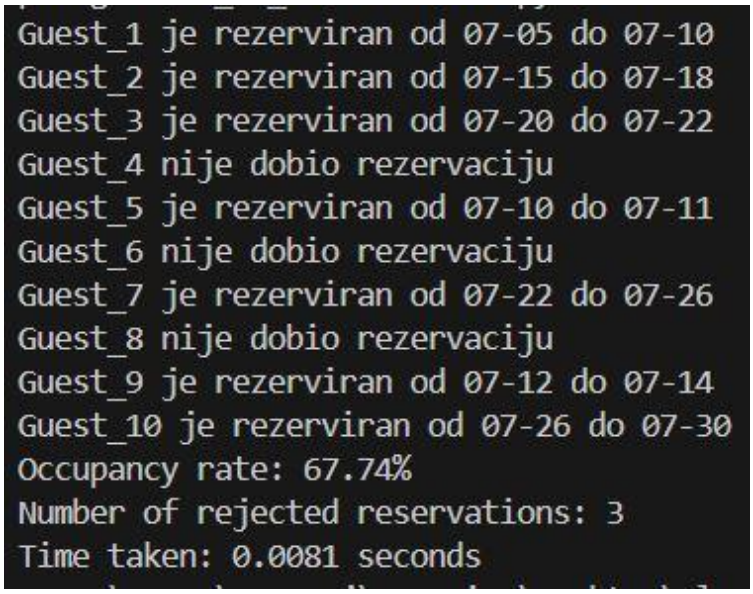
Programski kod 4.1. Funkcija za provjeru broja odbijenih rezervacija

Sljedeća je funkcija 'number_of_rejected_reservation()' koja vraća broj odbijenih rezervacija, to jest, broj gostiju kojima algoritam nije uspio dodijeliti termin rezervacije.

```python
def measure_time(func, *args, **kwargs):
    start_time = time.time()
    result = func(*args, **kwargs)
    end_time = time.time()
    elapsed_time = end_time - start_time
    return result, elapsed_time
```

Programski kod 4.3. Funkcija za mjerenje trajanja algoritma

I posljednja funkcija koja se u ovom radu koristi za evaluaciju učinkovitosti je funkcija 'measure_time()' koja jednostavno mjeri vrijeme potrebno za izvođenje algoritma.



Slika 4.1. Primjer ispisa s testnim rezultatima

Na slici 4.4. se vidi primjer ispisa za sve tri funkcije. Ovo je bio jedan primjer evaluacije 'flexible_reservation_algorithm-a' koji ima postotak popunjenosti od 67.74 %, 3 odbijene rezervacije te je vrijeme potrebno za izvođenje algoritma 0.0081 sekundi.

4.2. Evaluacija

U ovom dijelu predstavit će se konkretni rezultati dobiveni izvođenjem testova. Kako bi što bolje ispitali efikasnost algoritama definirana su 3 skupa podataka veličine 10, 30 i 100 podatkovnih uzoraka. Cilj nam je međusobna usporedba algoritama te analiza rada algoritama pri različitim veličinama skupova ulaznih podataka. Za svaki algoritam mjerimo njegovo

trajanje, broj odbijenih rezervacija i postotna popunjenost smještajnog kapacitete na svakom od 3 podatkovna skupa.

'first_come_first_served_algorithm':

- Podatkovni skup s 10 uzoraka:
    - Vrijeme izvođenja: 0,001 sekundi
    - Broj odbijenih rezervacija: 5
    - Postotna popunjenost: 51,61%
- Podatkovni skup s 30 uzoraka:
    - Vrijeme izvođenja: 0,002 sekundi
    - Broj odbijenih rezervacija: 21
    - Postotna popunjenost: 40,98%
- Podatkovni skup sa 100 uzoraka:
    - Vrijeme izvođenja: 0,007 sekundi
    - Broj odbijenih rezervacija: 74
    - Postotna popunjenost: 77,17%

'flexible_reservation_algorithm':

- Podatkovni skup s 10 uzoraka:
    - Vrijeme izvođenja: 0,0014 sekundi
    - Broj odbijenih rezervacija: 3
    - Postotna popunjenost: 67,74%
- Podatkovni skup s 30 uzoraka:
    - Vrijeme izvođenja: 0,0029 sekundi
    - Broj odbijenih rezervacija: 10
    - Postotna popunjenost: 73,77%
- Podatkovni skup sa 100 uzoraka:
    - Vrijeme izvođenja: 0,012 sekundi
    - Broj odbijenih rezervacija: 42
    - Postotna popunjenost: 96,74%

'greedy_reservation_algorithm':

- Podatkovni skup s 10 uzoraka:
    - Vrijeme izvođenja: 0,001 sekundi

    - Broj odbijenih rezervacija: 6
    - Postotna popunjenost: 54,84%
- Podatkovni skup s 30 uzoraka:
    - Vrijeme izvođenja: 0,003 sekundi
    - Broj odbijenih rezervacija: 22
    - Postotna popunjenost: 67,21%
- Podatkovni skup sa 100 uzoraka:
    - Vrijeme izvođenja: 0,008 sekundi
    - Broj odbijenih rezervacija: 81
    - Postotna popunjenost: 88,04%

'greedy_flexible_reservation_algorithm':

- Podatkovni skup s 10 uzoraka:
    - Vrijeme izvođenja: 0,002 sekundi
    - Broj odbijenih rezervacija: 3
    - Postotna popunjenost: 77,41%
- Podatkovni skup s 30 uzoraka:
    - Vrijeme izvođenja: 0,005 sekundi
    - Broj odbijenih rezervacija: 19
    - Postotna popunjenost: 100,00%
- Podatkovni skup sa 100 uzoraka:
    - Vrijeme izvođenja: 0,0179 sekundi
    - Broj odbijenih rezervacija: 80
    - Postotna popunjenost: 100,00%

Za posljednji algoritam, dodana su još dvije metrike za procjenu efikasnosti, a to su broj pomaknutih rezervacija i postotak pomaknutih rezervacija. One dakle predstavljaju broj, odnosno postotak gostiju kojima je algoritam uspio dodijeliti rezervaciju na temelju toga što im je pomaknuto prvobitnu rezervaciju za 1 ili 2 dana, prije ili poslije. To je omogućeno tako da je u algoritam dodana varijabla koja će pratiti broj gostiju kojima je promijenjen datum. Algoritam vraća tu varijablu koju ispisujemo, a također ju prosljeđujemo i u funkciju measure_shifted_reservations() s pomoću koje onda računamo i postotak pomaknutih rezervacija.

```
def measure_shifted_reservations(shifted_reservations, total_guests):

    return (shifted_reservations / total_guests) * 100
```

Programski kod 4.4. Funkcija za računanje postotka pomaknutih rezervacija

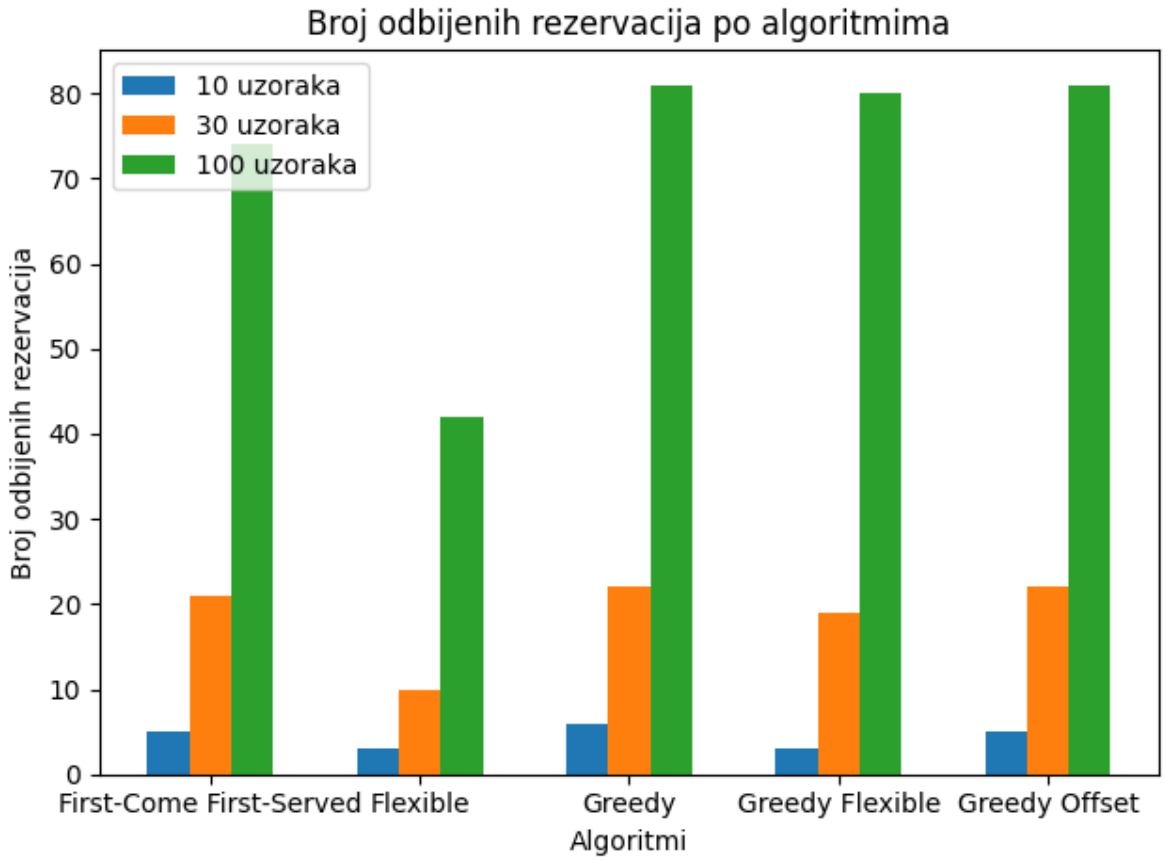
'greedy_offset_reservation_algorithm':

- Podatkovni skup s 10 uzoraka:
  - Vrijeme izvođenja: 0,001 sekundi
  - Broj odbijenih rezervacija: 5
  - Postotna popunjenost: 67,74%
  - Broj pomaknutih rezervacija: 2
  - Postotak pomaknutih rezervacija: 20.00%
- Podatkovni skup s 30 uzoraka:
  - Vrijeme izvođenja: 0,004 sekundi
  - Broj odbijenih rezervacija: 22
  - Postotna popunjenost: 68,85%
  - Broj pomaknutih rezervacija: 1
  - Postotak pomaknutih rezervacija: 3.33%
- Podatkovni skup sa 100 uzoraka:
  - Vrijeme izvođenja: 0,011 sekundi
  - Broj odbijenih rezervacija: 81
  - Postotna popunjenost: 89,13%
  - Broj pomaknutih rezervacija: 4
  - Postotak pomaknutih rezervacija: 4.00%

Prikazani rezultati daju neke uvide u sam rad algoritama. Rezultati pokazuju da su algoritmi koji pri svojim izračunima uzimaju u razmatranje fleksibilnost gosta znatno učinkovitiji. Najbolju postotnu popunjenost ostvario je 'greedy_flexible_reservation_algorithm' koji je uz to imao mali broj odbijenih rezervacija, dok je npr. 'first_come_first_served_algorithm' imao nižu popunjenost. 'greedy_offset_reservation_algorithm' također postiže visoku popunjenost međutim vrijeme izvođenja ovog algoritma predstavlja nedostatak kada je brzina ključna. Fleksibilni algoritmi su pokazali bolju ravnotežu između brzine izvođenja i učinkovitosti, što ih čini pogodnijima za optimizaciju smještaja.

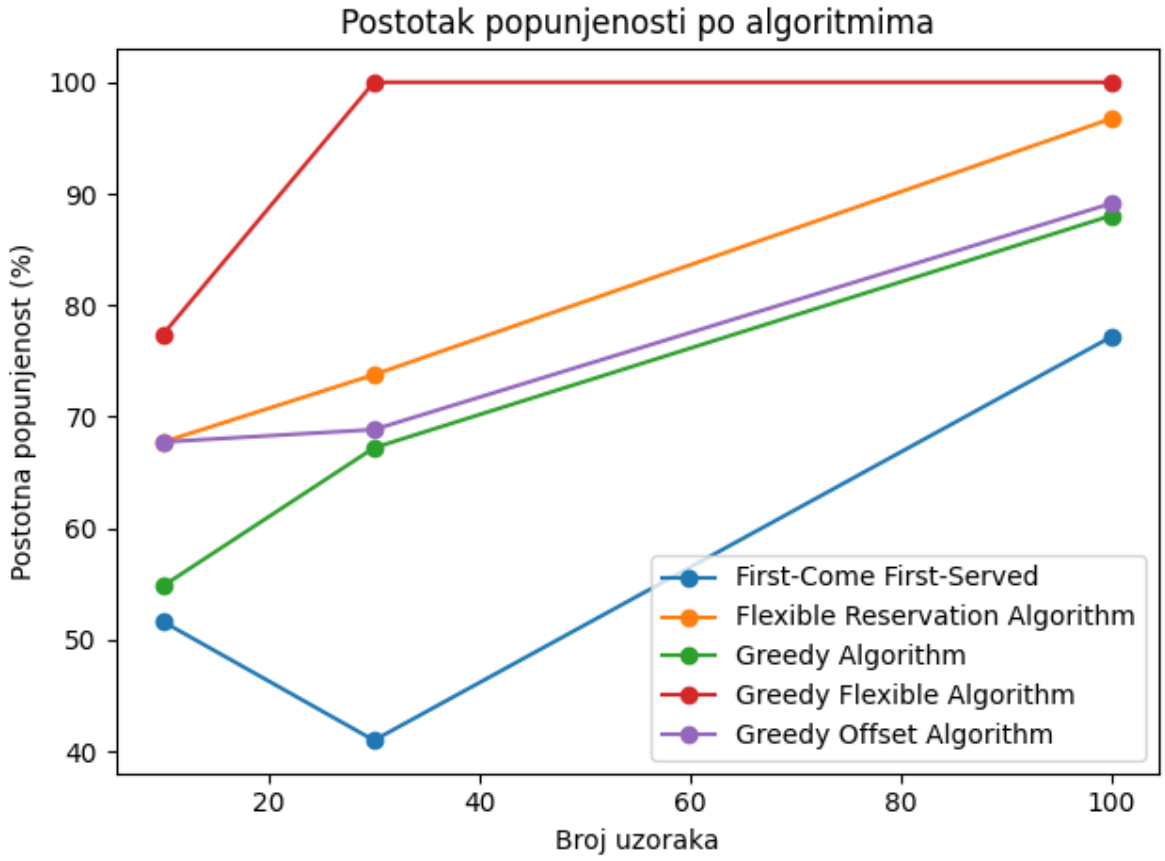
4.3. Vizualizacija podataka

U ovom poglavlju grafički ćemo prikazati odnose između algoritama na temelju podataka za različite mjere efikasnosti koje smo računali u prethodnom poglavlju.



Slika 4.2. Prikaz broja odbijenih rezervacija za svaki algoritam u svakom podatkovnom skupu

Slika 4.2. prikazuje ukupan broj odbijenih rezervacija za svaki algoritam na 10, 30 i 100 uzoraka. Uočava se dosljedan učinak algoritama na svakom podatkovno skupu, to jest algoritam koji ima najmanje odbijenih rezervacija na skupu od 10 primjera ima najmanje odbijenih rezervacija i na skupu od 100 primjera. Također, može se primijetiti da algoritmi koji se temelje ne pohlepnom pristupu imaju veoma slične vrijednosti, pogotovo na skupu od 100 uzoraka.



Slika 4.3. Prikaz promjene postotne popunjenosti ovisno o broju uzoraka

Slika 4.3. prikazuje kako se mijenja postotna popunjenost smještajnih kapaciteta za svaki algoritam s porastom broja uzoraka. Primjećujemo kako se na 2 od 5 algoritama događa pad postotne popunjenosti kada se broj uzoraka poveća s 10 na 30. Primjećujemo gotovo linearan rast kada je u pitanju 'flexible_reservation_algorithm', dok se najefikasnijim ispostavlja 'greedy_flexible_algorithm' koji za zadane skupove podataka od 30 i 100 primjera ostvaruje postotnu popunjenost od 100 %.

4.4. Praćenje napretka

U ovom potpoglavlju predstavit će se postupno poboljšanje algoritama uzimajući algoritam 'first_come_first_served_algorithm' kao polazišnu točku. Na temelju tog algoritma će se zatim nadograđivati značajke koje omogućavaju da što više ljudi dobije rezervaciju. Promjene ćemo testirati na podatkovnom skupu od 30 uzoraka, a mjerit će se samo postotna popunjenost. U jednom od prethodnih poglavlja izračunata je postotna popunjenost algoritma 'first_come_first_serverd_algorithm' i ona na skupu od 30 primjera iznosi 40,98 %. Kako bi se

ona povećala u algoritam je potrebno dodati određena proširenja. Jedno od njih je i proširenje pretrage. To znači da će, ukoliko željeni datum nije dostupan, a gost je fleksibilan, algoritam provjeravati nekoliko dana prije ili poslije (npr +/- 2 dana od željenog datuma).

```
if not all_days_free and flexible:
            for offset in range(-search_range, search_range + 1):
                alternate_start_day = start_day + timedelta(days=offset)
                guest_days = [alternate_start_day + timedelta(days=i) for i in
range(int(duration))]

                if all(day <= end_date and day not in occupied_days for day in
guest_days):
                    all_days_free = True
                    break
```

Programski kod 4.5. Prikaz proširenja pretrage

U Programskom kodu 4.5. prikazana je opcija koja se dodaje u 'first_come_first_served_algorithm' u svrhu proširenja pretrage nekoliko dana od željenog datuma. Nakon provedenih testova na skupu od istih 30 uzoraka na kojima su provođeni testovi za sve algoritme, rezultat za postotnu popunjenost iznosi 60,66 %, što predstavlja značajno poboljšanje.

Kako bismo dodatno povećali popunjenost u algoritam je moguće dodati opciju koja će za sve goste koji nisu dobili rezervacije u prethodnim verzijama algoritma, provjeravati opciju fleksibilnost te ukoliko je ona 'True', pokušat će se pronaći bilo koji datum unutar raspona s istom duljinom trajanja boravka i dodijeliti gostu.

```
def find_any_available_dates(start_date, duration, occupied_days, end_date):
    current_day = start_date
    while current_day <= end_date:
        if is_date_available(current_day, duration, occupied_days, end_date):
            return [current_day + timedelta(days=i) for i in
range(int(duration))]
        current_day += timedelta(days=1)
    return None
```

Programski kod 4.6. Funkcija za pretragu preostalih slobodnih dana

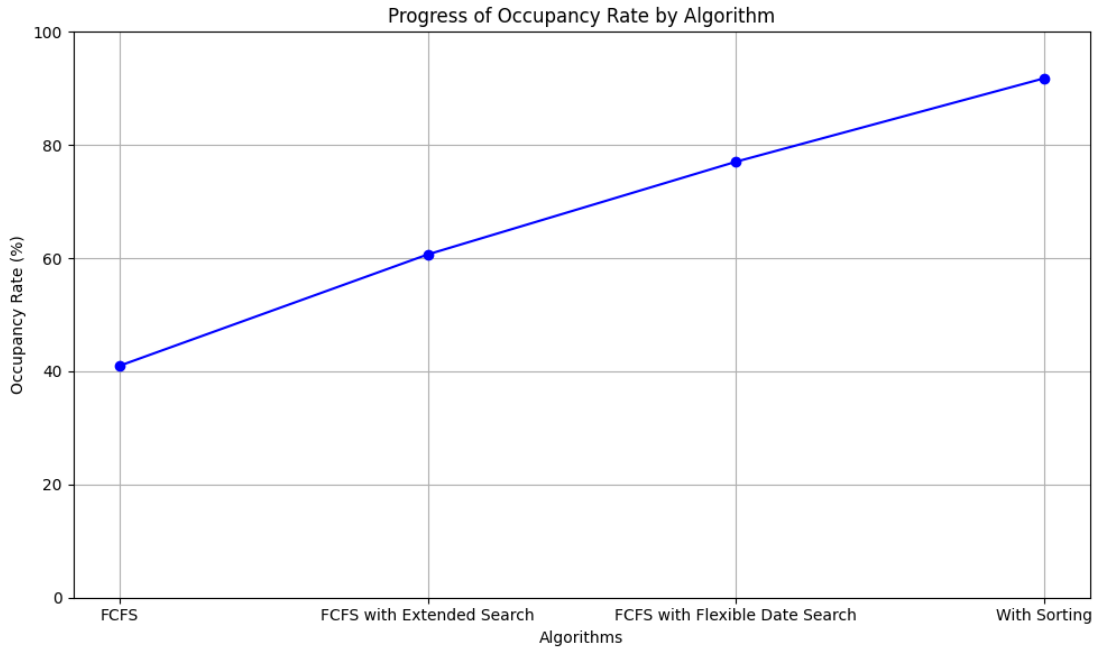
U algoritam je dodana funkcija 'find_any_available_dates()' koja je prikazana u Programskom kodu 4.6. Nakon provedenih testova rezultat za postotnu popunjenost iznosi 77,05 %, što je opet veliki napredak u odnosu na prethodnu verziju.

Za dodatno povećanje popunjenosti moguće je sortirati rezervacije od onih duljih prema kraćim, a nakon toga raspoređivati preostale rezervacije prema kriterijima fleksibilnosti.

```
    df = df.sort_values(by='Stay duration', ascending=False)
```

Programski kod 4.7. Sortiranje prema duljini boravka

U algoritam je dodano sortiranje rezervacija, a ostatak je ostao isti. Dakle promjena je samo u tome što se prvo provjeravaju dulje rezervacije, nakon toga vrši se prvo provjera slobodnih dana u rasponu +/- 2 dana i na kraju pronalazak bilo kojih slobodnih datuma s istom duljinom boravka za fleksibilne goste. Nakon odrađenih testova rezultat za postotnu popunjenost iznosi 91.80 %.



Slika 4.4. Prikaz poboljšanja s dodavanjem dodatnih opcija

Na grafu je prikazan postupni napredak počevši od algoritma 'first_come_first_served', dodavanje pojedinih značajki uočavamo kako se i postotna popunjenost povećava.

5. ZAKLJUČAK

Glavni zadatak ovog završnog rada bio je predstaviti i testirati metode za povećanje ukupne popunjenosti smještajnog kapaciteta. U tu svrhu analizirano je nekoliko algoritama za rješavanje tog problema. Algoritmi su predstavljeni u dvije glavne skupine: algoritme temeljene na redoslijedu dospijeća zahtjeva i algoritme za rezervaciju prema duljini trajanja boravka. U prvoj skupini analizirana su dva algoritma koja se razlikuju prema tome uzimaju li u obzir fleksibilnost gosta ili ne. U drugoj skupini predstavljena su tri algoritma od kojih se prva dva također razlikuju prema kriteriju fleksibilnosti, dok je svrha trećeg algoritma bila predstaviti rješenje problema koje bi možda lakše našlo primjenu u stvarnom svijetu. Nakon što su svi algoritmi opisani uslijedilo je njihovo testiranje prema metrikama koje su također predstavljene i opisane u ovom radu. Provedena su testiranja nad algoritmima i dobiveni su podatci o učinkovitosti algoritama. Na temelju tih podataka može se zaključiti da su algoritmi koji su u svojim izračunima i rasporedima rezervacija uzimali u razmatranje kriterij fleksibilnosti gostiju, dali bolje rezultate od ostalih algoritama, te prema tome njih možemo označiti kao efikasnije.

LITERATURA

[1] Programiz, »Backtracking Algorithm«, [Mrežno]. Available at: https://www.programiz.com/dsa/backtracking-algorithm.

[2] MathWorks, »What is the Genetic Algorithm?«, [Mrežno]. Available at: https://www.mathworks.com/help/gads/what-is-the-genetic-algorithm.html

[3] Britannica, »Linear programming mathematics«, [Mrežno]. Available at: https://www.britannica.com/science/linear-programming-mathematics

[4] Visual Studio Code, [Mrežno]. Available at: https://code.visualstudio.com/docs

[5] Visual Studio Code, »extensions are all you need«, [Mrežno]. Available at: https://code.visualstudio.com/blogs/2024/06/24/extensions-are-all-you-need

[6] Justapedia, »Python programming language«, [Mrežno]. Available at: https://justapedia.org/wiki/Python_(programming_language).

[7] Python Institute, »about python«, [Mrežno]. Available at: https://pythoninstitute.org/about-python

[8] Pandas, »pandas«, [Mrežno]. Available at: https://pandas.pydata.org/

[9] NumPy, »stable«, [Mrežno]. Available at: https://numpy.org/doc/stable/

SAŽETAK

Ovaj završni rad daje nam uvid u algoritme i metode čija je svrha povećanje ukupne popunjenosti smještajnih kapaciteta. U uvodnom dijelu rada detaljnije su opisani postojeći algoritmi s istom svrhom. Potom su predstavljene tehnologije korištene pri izradi algoritama, razvojno okruženje, programski jezik te biblioteke koje su korištene. U glavnom dijelu opisan je razvoj algoritama. Prvo su opisani potrebni ulazni podaci te je objašnjen način prikaza podataka, a nakon toga su analizirani svi algoritmi uz uvid u kod i primjer ispisa za unaprijed definirani skup ulaznih podataka. Date su određene pretpostavke o efikasnosti algoritama te prednosti i potencijalni nedostatci u radu algoritama. U završnom dijelu testirani su konkretni algoritmi i praćen je napredak koristeći određene funkcije prema kojima su vrednovani. Na temelju dobivenih podataka doneseni su konačni utisci i mišljenja o učinkovitosti algoritama u zaključku.

Ključne riječi: algoritmi, fleksibilnost, povećanje popunjenosti, testiranje

ABSTRACT

Title: Methods for Increasing Accommodation Capacity Utilization

This thesis provides an insight into algorithms and methods aimed at increasing the overall occupancy of accommodation capacities. The introduction section describes existing algorithms with the same purpose in more detail. Subsequently, the technologies used in the development of the algorithms, the development environment, programming language, and libraries utilized are presented. The main section details the development of the algorithms. Initially, the necessary input data are described, and the method of data presentation is explained. Following this, all algorithms are analyzed with an overview of the code and an example output for a predefined set of input data. Assumptions about the efficiency of the algorithms, as well as their advantages and potential drawbacks, are discussed. In the final section, specific algorithms are tested, and progress is tracked using certain functions by which they are evaluated. Based on the obtained data, final impressions and opinions on the effectiveness of the algorithms are presented in the conclusion.

Keywords: algorithms, flexibility, occupancy increase, testing

ŽIVOTOPIS

Domagoj Marinčić, rođen je 28. lipnja 2001. u Žepču. Osnovnu školu pohađao je u Osnovnoj školi fra Grga Martić u Brankovićima. 2016. godine upisuje Opću gimnaziju u Katoličkom školskom centru „Don Bosco“ u Žepču. Nakon srednje škole upisuje preddiplomski sveučilišni studij Računarstva, smjer Programsko inženjerstvo, na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku pri Sveučilištu Josipa Jurja Strossmayera u Osijeku, gdje i trenutno studira za vrijeme pisanja ovog rada.