MATEMATIČKI FAKULTET
UNIVERZITET U BEOGRADU

Mladen Lazić

Realizacija veb aplikacije za problem rutiranja vozila

master rad

Beograd, 2019.

Mentor: **doc. dr Aleksandar Kartelj**
Matematički fakultet u Beogradu

Članovi komisije: **prof. dr Vladimir Filipović**
Matematički fakultet u Beogradu

prof. dr Saša Malkov
Matematički fakultet u Beogradu

Datum odbrane: _____

SADRŽAJ

1. UVODNI DEO

1.1 Uvod

Mnogi problemi dostave, kao što su dostava novina, hrane ili pošte mogu se definisati kao problem rutiranja vozila (eng. Vehicle Routing Problem - VRP). Problem rutiranja vozila je formalno uveden 1959. godine od strane Dantzig-a i Ramser-a. [1] Ovi autori su predložili jednostavno heurističko rešenje koje su ilustrovali na problemu malih dimenzija. Sledećih godina došlo je do pojave nekoliko heuristika zasnovanih na različitim principima uključujući štednju, geografsku blizinu itd. Heuristika štednja je poznata po svojoj brzini, jednostavnosti i tačnosti. Ovu heuristiku su predstavili Clarke i Wright. [2] Razvoj konkretnih algoritama za problem rutiranja vozila je počeo 1981. godine objavom dva rada. Prvi od ovih radova predložio je algoritam zasnovan na dinamičkom programiranju sa relaksacijom vremena i prostora dok je drugi predlagao dve matematičke formulacije. [3] [4] Nekoliko godina kasnije, Laporte, Desrochers i Nobert su prvi put predložili pristup sečiva za problem rutiranja vozila koji je baziran na rešenju linearne relaksacije celobrojnog modela. [5] Ovaj pristup je otvorio put ka novijim algoritmima. Od tada su predloženi različiti algoritmi bazirani na formulacijama matematičkog programiranja. Neke od najuspešnijih implementacija su od strane Fukasawa-a 2006. godine i Baldacci-ja 2008. godine. [6] [7] Detaljnije o problemu rutiranja vozila i načinu rešavanja se može naći u Laporte-ovom radu iz 2013. [8]

Implementacija modernih heuristika za problem rutiranja vozila je počela 1990-tih godina sa nastajanjem metaheuristika. Istraživanje ovog problema je stimulisalo porast i razumevanje metaheuristika koje su tada bile poznate. Rano istraživanje u ovoj oblasti je bilo podeljeno, sa vidljivom pristrašnošću prema tabu pretrazi predloženoj od strane Fred. W. Glover-a 1986. godine koja je formalizovana 1989. godine. [9] [10] Najbolje metaheuristike su one koje izvršavaju široku i duboku pretragu prostora rešenja i koje mogu da reše nekoliko varijanti problema. One generalno koriste više operatora, kao što je velika pretraga susedstva, predložena od strane Pisinger-a i Ropke-a 2007. godine [11] ili kombinuju genetičku pretragu sa lokalnom pretragom kao što je u

hibridnom genetičkom algoritmu koji je predstavljen od strane Vidal-a 2012. godine. [12]

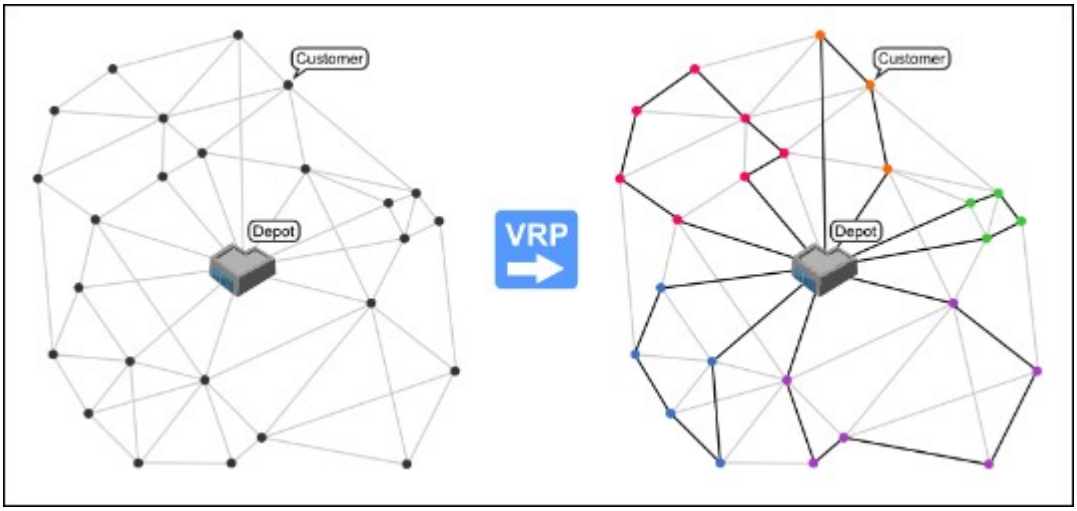
U ovom radu u poglavlju 1 će biti detaljno opisan problem rutiranja vozila. U poglavlju 2 će biti opisana dva algoritma za rešavanje problema: metoda zasnovana na totalnoj enumeraciji i heuristika simulirano kaljenje, kao i implementacija istih. Biće opisana funkcija cilja, kriterijumi zaustavljanja i kodiranje rešenja. U poglavlju 3 će biti testirani algoritmi na veštački generisanim test primerima kojih ima 6, a sadrže redom 5, 10, 12, 15, 25 i 50 lokacija za isporuku. Rezultati će biti prikazani i upoređeni dijagramski i tabelarno. U poglavlju 4 će biti opisane tehnologije koje se koriste kako na klijentskoj strani tako i na serverskoj i biće objašnjen način upotrebe grafičkog okruženja. U poglavlju 5 će biti predložena moguća unapređenja algoritama i grafičkog okruženja i biće navedeni doprinosi rada.

Cilj rada je razvoj veb aplikacije u kojoj će klijentska strana omogućavati da korisnik na jednostavan način, putem grafičkog interfejsa, unosi podatke o ulaznim parametrima: skladištu, lokacijama isporuke i broju vozila. Na serverskoj strani će biti implementirana logika aplikacije, koja će podrazumevati algoritme za rešavanje problema rutiranja kao i svu prateću serversku logiku za opsluživanje zahteva klijenta. Rute između dve lokacije će biti fiksirane i za njihovo dobijanje biće korišćen *Google Map API*.

Tehnologije koje će biti korištene na klijentskoj strani za implementaciju grafičkog korisničkog interfejsa su *HTML5, CSS3, JavaScript, AngularJS i Bootstrap*. Detaljan opis ovih tehnologija dat je u poglavlju 4.1. Na serverskoj strani za implementaciju algoritama, metode zasnovane na totalnoj enumeraciji i simulirano kaljenje će biti korišten *PHP*. Detaljnije o tome će biti opisano u poglavlju 4.2. Za komunikaciju između klijentske i serverske strane će biti korišten *JSON* format. Takođe će se koristi *Apache* veb server sa pokretanje aplikacije lokalno. Pregledači koji će biti korišteni su *Chrome* i *Mozila Firefox.* Za otklanjanje grešaka će se koristiti *WebInspector.* Za pisanje programskog kôda će biti korišten tekst editor *Sublime3.*

1.2 Opis problema rutiranja vozila

Data je flota vozila koja se nalazi u skladištu i zadat je skup lokacija koje vozila treba da posete radi dostavljanja određene količine robe na te lokacije. Svako vozilo ima definisan maksimalan kapacitet. Ne moraju biti iskorištena sva vozila iz skladišta. Lokacije isporuke smeju biti posećene samo jednom. Osnovni cilj je odrediti rute kojima će se vozila kretati tako da roba bude isporučena na svim lokacijama uz minimizaciju ukupnih troškova. Vozila se nakon isporuke robe vraćaju u polazno skladište. Trošak se može definisati na različite načine, npr. kao ukupno vreme potrebno da se sva vozila vrate nazad u skladište, ukupna potrošnja goriva, ukupno pređeno rastojanje, itd. U ovom slučaju radi se minimizacija po ukupnom rastojanju. Grafički prikaz problema i njegovog rešenja dat je na Slici 1.



Slika 1. VRP primer rešenja

Problem rutiranja vozila predstavlja uopštenje problema trgovačkog putnika kod koga jedan putnik kreće iz skladišta i potrebno je da obiđe sve lokacije samo jednom i da se vrati u skladište ali da pređeni put bude najmanji [13]. Određivanje optimalnog rešenja spada u grupu NP-teških problema tako da su dimenzije problema koje se mogu efikasno rešiti ograničene. [14] Za probleme većih dimenzija koristi se heuristički pristup.

2. PREDLOŽENA METODA SIMULIRANOG KALJENJA ZA REŠAVANJE PROBLEMA RUTIRANJA VOZILA

2.1 Opis metode simuliranog kaljenja

Simulirano kaljenje je (meta)heuristička metoda za rešavanje optimizacionih problema koja se uobičajeno koristi u problemima gde je prostor pretraživanja diskretan. Postoji veliki broj slabo struktuiranih problema koji se može rešiti u zadovoljavajućem vremenu zahvaljujući razvoju heurističkog programiranja. Veliki broj problema iz raznih oblasti kao što su upravljanje i planiranje se rešavaju isključivo ovakvim tehnikama. Heurističko programiranje nije uvek pouzdan izbor za rešavanje problema ali daje približno ili tačno rešenje i prevazilazi mnoge nedostatke strogih metoda. Kombinatorni problemi su obično slabije struktuirani, višekriterijumski ili nelinearni što ih čini sve više komplikovanijim za rešavanje. Međutim, razvijen je veliki broj heurističkih pristupa koji rešavaju i takvu vrstu problema. Neke od heurističkih metoda, pored simuliranog kaljenja, su:

- Genetski algoritmi;
- Tabu pretraga;
- Optimizacija kolonijom mrava;
- Metoda promenljivih okolina.

Kaljenje je tehnika koja se inicijalno koristi u metalurgiji, grana nauke o materijalima i njihovim legurama. Proces metalurškog kaljenja se sastoji od više koraka. Prvo je potrebno zagrejati metal do veoma visoke temperature, zadržati metal na toj temperaturi a zatim početi sa procesom hlađenja sve dok se metal ne stvrdne. Toplota utiče na atome tako što ih oslobađa od njihovog početnog položaja i dopušta im da se slobodno kreću na višim energetskim nivoima. Sporo hlađenje daje atomima veću verovatnoću da pronađu razmeštaj u kojem će imati manju unutrašnju energiju nego pre procesa kaljenja. Manja unutrašnja energija atoma znači veću stabilnost i bolje karakteristike. Sa hlađenjem struktura metala postaje fiksna što prouzrokuje zadžavanje

novodobijenih svojstava materijala. Ako pustupak hlađenja nije dovoljno spor može dovesti do formiranja nepravilnog oblika kristalne rešetke, odnosno do lakog pucanja materijala. Scott Kirkpatrick je sa autorima Daniel Gelatt-om i Mario Vecchi-jem 1983. godine objavio rad u časopisu *Science* pod nazivom „Optimization by Simulated Annealing", u kome su prvi put izloženi osnovni principi ovog algoritma. [15]

U kontekstu implementacije metoda simuliranog kaljenja počinje se slučajnim izborom početnog rešenja i postavljanjem početne temparature na veoma visoku vrednost. Zatim se na slučajan način bira novo rešenje iz okoline postojećeg rešenja. Ako je novoizabrano rešenje bolje od trenutnog onda to rešenje postaje najbolje trenutno rešenje. Međutim, ako novoizabrano rešenje nije bolje od trenutnog rešenja, izračunava se verovatnoća mogućnosti da novoizabrano ipak postane trenutno najbolje rešenje. Verovatnoća prihvatanja lošijeg rešenja zavisi od parametra koji predstavlja temperaturu. Kako se algoritam izvršava, vrednost tog parametra opada. Opisani pristup obezbeđuje izlazak iz lokalnog optimuma. Na početku algoritma vrednost verovatnoće je velika pa će se u cilju prevazilaženja lokalnog optimuma ipak izabrati lošije rešenje. Tokom izvršavanja algoritma ta verovatnoća se smanjuje. Pred kraj izvršavanja algoritma verovatnoća prihvatanja lošijeg rešenja će biti jako mala pa se lošije rešenje verovatno neće ni prihvatiti jer se smatra da se trenutno rešenje nalazi blizu najboljeg rešenja ili da je optimum dostignut, odnosno da je nađeno najbolje rešenje. Konkretan opis algoritma zasnovanog na simuliranom kaljenju za rešavanje problema rutiranja vozila dat je u narednom poglavlju.

2.2 Opis metoda za rešavanje problema rutiranja vozila

Problem rutiranja vozila može biti rešen upotrebom raznih metoda. U ovom slučaju problem je rešen upotrebom sledećih:

- Metoda zasnovana na totalnoj enumeraciji;
- Simulirano kaljenje.

Kao što je već istaknuto, problem rutiranja vozila je NP-težak problem i kao takav ne može biti rešen metodom zasnovanoj na totalnoj enumeraciji u razumnom vremenu

ako na ulazu postoji veliki broj lokacija jer može doći do kombinatorne eksplozije. Simulirano kaljenje je metoda koja daje približno ili tačno rešenje u tom slučaju. Na Slici 2. dat je pseudokod metode simulirano kaljenje.

```
Algoritam SimuliranoKaljenje
Ulaz: locationsIndex, distanceBetweenLocations, distanceDepotFromLocations, goodsPerLocations,
vehiclesIndex, vehiclesCapacity
Izlaz: dvodimenzionalni niz – svaki podniz sadrži putanju za jedno vozilo

begin
   currentTemperature := 100.0
   coolingRate := 0.9999
   minimalTemperature := 0.0001
   candidate:= shuffle(locationIndex)
   while currentTemperature > minimalTemperature
      randomCandidate := generiši random kandidat
      distanceRandomCandidate := izračunaj kilometrazu za random kandidata
         if distanceRandomCandidate je validno
            if novo rešenje je bolje od trenutnog
               candidate := randomCandidate
               distanceCandidate := distanceRandomCandidate
            else
               E = distanceCandidate – distanceRandomCandidate
               T = currentTemperature
               random := generiši random broj [0, 1]
               if random <  (exp(E/T))
                  candidate := randomCandidate
                  distanceCandidate := distanceRandomCandidate
      currentTemperature = currentTemperature * coolingRate
   return candidate
end
```

Slika 2. Simulirano kaljenje

Ulazni parametri koji se koriste za izvršavanje algoritma simuliranog kaljenja i metode zasnovanoj na totalnoj enumeraciji koja je objašnjena u poglavlju 2.8 su:

1. *locationsIndex* – indeksi lokacija isporuke koji su definisani prilikom unosa lokacija. Predstavljeni su o obliku niza;

2. *distanceBetweenLocations* – udaljenosti izražene u kilometrima između svih lokacija isporuke. Podaci se dobijaju upotrebom *Google Map API-ja* koji je objašnjen u poglavlju 4 i predstavljeni su u obliku matrice;
3. *distanceDepotFromLocations* – udaljenosti između skladišta i lokacija ispruke izražene u kilometrima. Takođe su dobijeni upotrebom *Google Map API-ja* i predstavljeni su u obliku niza;
4. *goodsPerLocations* – podaci koji predstavljaju količinu robe koju je potrebno dostaviti na lokacije. Predstavljeni su u obliku niza;
5. vehicleIndex – indeksi vozila koji su definisani prilikom unosa. Predstavljeni su u obliku niza;
6. *vehicleCapacity* – podaci koji predstavljaju maksimalan kapacitet vozila. Predstavljeni su u obliku niza.

Na početku algoritma definišu se promenljive koje su ključne u izvršavanju algoritma:

1. *currentTemperature* = 100 – trenutna temperatura koja se u svakom koraku iteracije smanjuje za *coolingRate* faktor dok ne dostigne vrednost *minimalTemperature;*
2. *coolingRate* = 0.999999 – faktor hlađenja čija je vrednost fiksna tokom izvršavanja algoritma koji služi za hlađenje, odnosno promenu vrednosti *currentTemperature* koji teži *minimalTemperature;*
3. *minimalTemperature* = 0.0001 – minimalna temperatura. Ukoliko je dostignuta ili je manja od *currentTemperature* algoritam se zaustavlja;
4. *candidate* – kandidat rešenja. Na početku algoritma dobija vrednost promenljive *locationsIndex;*
5. *distanceCandidate* – čuva vrednost kandidata rešenja, odnosno potrebnu kilometražu da se obiđu sve lokacije po tom rešenju, dobijenu pozivom funkcije *getDistance*, koja će detaljno biti opisana u nastavku;
6. *randomCandidate* – potencijalni kandidat rešenja koji se dobija pozivom funkcije *generateRandomCandidate* nad kandidatom rešenja. Opis funkcije biće dat u nastavku;

7. *distanceRandomCandidate* - čuva vrednost potencijalnog kandidata rešenja, odnosno potrebnu kilometražu da se obiđu sve lokacije po tom rešenju dobijenu pozivom funkcije *getDistance*.

Na početku algoritma prvo je potrebno primeniti slučajnu permutaciju nad nizom *candidate* da bi se dobio slučajan raspored lokacija u nizu kao prvi kandidat rešenja. Nakon toga izračunava se kilometraža koju je potrebno preći tim kandidatom rešenja pozivom funckije *getDistance*. Navedena funkcija može vratiti izračunatu kilometražu ili *-1* ukoliko rešenje nije ostvarivo. Ako je povratna vrednost *-1* vrednost promenljive *distanceCandidate* se postavlja na nedostižnu vrednost čime se obezbeđuje da će prvo ostvarivo rešenje biti prihvaćeno kada se dobije u daljem izvršavanju algoritma. Zatim se prolazi kroz petlju sve dok je vrednost *currentTemperature* veća od *minimalTemperature*. U petlji se izračunava *randomCandidate* pozivom funkcije *generateRandomCandidate* u odnosu na trenutnog kandidata rešenja. Potom je potrebno izračunati *distanceRandomCandidate*, odnosno kilometražu koju je potrebno preći po tom rešenju. Ukoliko je vrednost *distanceRandomCandidate* manja od vrednosti *distanceCandidate* znači da je dobijeno rešenje bolje od trenutnog i dobija status najboljeg trenutnog rešenja. Ako ipak to nije slučaj potrebno je izračunati verovatnoću uzimanja novodobijenog rešenja. Da bi verovatnoća bila izračunata mora se izračunati razlika *E* između *distanceRandomCandidate* i *distanceCandidate* i generisati slučajan broj u opsegu [0, 1]. Ukoliko je slučajan broj manji od $e^{(E-currentTemperature)}$ novo rešenje ipak može biti prihvaćeno. Na kraju iteracije trenutna temperatura *currentTemperature* se umanjuje, odnosno množi sa faktorom hlađenja. Algoritam će se izvršavati dok vrednost *currentTemperature* ne postane manja ili jednaka sa *minimalTemperature* nakon čega je dobijeno približno ili tačno rešenje. Krajnje rešenje se čuva u promenljivoj *result* koja se vraća klijentskoj strani u određenom *JSON* formatu o kome će biti reči u poglavlju 2.3.

2.3 Kodiranje rešenja

Rešenje je predstavljeno u obliku dvodimenzionalong niza. Broj elemenata dvodimenzionalnog niza je jednak broju vozila. Elementi dvodimenzionalnog niza sadrže

podatke za svako vozilo pojedinačno. Ukoliko vozilo nije iskorišteno - niz za to vozilo je prazan što znači da neće biti korišteno u daljem izvršavanju aplikacije, odnosno u prikazivanju rute za to vozilo i popunjavanja izveštaja za korisnika. Rezultat koji se dobija od servera je u sledećem *JSON* formatu:

```
[
 [4, 0, 2, 12.029],
 [],
 [1, 3, 15.213]
]
```

U prikazanom rezultatu postoje tri vozila koja je bilo moguće koristiti u ovom izračunavanju. Svako vozilo ima svoj indeks koji je predstavljen pozicijom u dvodimenzionalnom nizu. Ako ceo rezultat označimo sa X, X[0] su podaci koje važe za vozilo sa indeksom 0, X[1] su podaci koji važe za vozilo sa indeksom 1 itd. do kraja dvodimenzionalnog niza. Kao što je istaknuto, podaci koji su vezani samo za jedno vozilo su takodje predstavljeni nizom. Neka je sa N označen broj elemenata tog niza. Prvih N-1 vrednosti predstavljaju indekse lokacija koje to vozilo treba da obiđe. U datom primeru, vozilo sa indeksom 0 treba da obiđe redom lokacije koje imaju indeks 4, 0, 2. Lokacije se moraju obilaziti u tom redosledu da bi ukupna kilometraža bila tačna. Na poslednjoj poziciji nalazi se podatak koliko će vozilo preći kilometara u tom obilasku. U tu sumu je uračunata putanja od lokacije skladišta do prve lokacije u nizu za to vozilo i od poslednje lokacije u nizu do lokacije skladišta. Po formulaciji problema vozilo mora da se vrati u lokaciju iz koje je krenulo, u ovom slučaju je to skladište. Analogno važi i za sva ostala vozila u nizu. Već je istaknuto da postoji mogućnost da vozilo neće biti korišteno kao u slučaju vozila na poziciji X[1]. Niz neiskorištenog vozila je prazan.

2.4 Konstrukcija rešenja na osnovu kôda rešenja

U ovom delu biće objašnjen način raspoređivanja robe po vozilima na osnovu dobijene permutacije lokacija. Funkcija koja je zadužena za ovo izračunavanje je *fillVehicle*. Navedena funkcija se koristi u simuliranom kaljenju i u metodi zasnovanoj na

totalnoj enumeraciji. Na svaku lokaciju može biti isporučena različita količina robe. Vozila mogu biti različitih kapaciteta. Ukupna kilometraža je usko povezana sa količinom robe koja se isporučuje i kapacitetom vozila.

Primer 1. *Neka postoje tri lokacije koje su jedna do druge, u oznakama A, B i C i jedna lokacija koja je na suprotnoj strani u oznaci D. Lokacije zahtevaju isporuku robe u narednim količinama redom 10, 50, 5, 10. Na raspolaganju su dva vozila X i Y kapaciteta 50 i 25. Neka je skladište podjednako udaljeno od lokacija koje su blizu jedna druge, A, B i C i lokacije D. Najekonomičnije rešenje bi bilo da jedno vozilo dostavi robu na lokacije A, B i C a drugo vozilo na lokaciju D. Međutim, pošto kapacitet vozila ne dopušta takvo rešenje, očigledno je da će doći do ukrštanja putanja pošto vozilo X mora posetiti lokaciju B, jer vozilo Y nema dovoljno kapaciteta da bi dostavilo robu i na tu lokaciju. Vozilo Y će posetiti lokacije A, C i D.*

Dakle, ukupna pređena kilometraža u velikom procentu zavisi od kapaciteta vozila i količine robe koja treba biti dostavljena. Implementacija funkcije *fillVehicle* je prikazana na Slici 3.

Raspored vozila u nizu koji se dobija kao argument funkcije nije bitan jer će funkcija biti pozvana za svaku permutaciju vozila i u slučaju metode zasnovane na totalnoj enumeraciji i u slučaju heuristike simulirano kaljenje. Prvo je potrebno izračunati ukupnu količinu robe koju treba isporučiti datom flotom vozila. Zatim se započinje iteriranje kroz petlju do trenutka dok se ne rasporedi sva roba po vozilima ili dok se ne iskoriste sva vozila koja su na raspolaganju. Glavni deo implementacije proverava da li količina robe trenutne lokacije koja se obrađuje može biti spakovana u trenutno vozilo. Ako je to moguće, roba se pakuje u to vozilo i od ukupnog ili preostalog kapaciteta vozila koje je na raspolaganju oduzima se količina koja je spakovana u njega. Nakon toga prelazi se na drugu lokaciju. Ako nije moguće spakovati robu u trenutno vozilo, prelazi se na sledeće vozilo. Ukoliko je spakovana roba za poslednju lokaciju, pakovanje je uspešno završeno. U slučaju da se došlo do poslednjeg vozila a roba nije uspešno spakovana, pakovanje nije moguće izvršiti za datu permutaciju lokacija te se prelazi na sledeću permutaciju. Ako robu nije moguće spakovati u raspoloživu flotu vozila ili je najveća količina robe koju je potrebno dostaviti na neku lokaciju veća od najvećeg

kapaciteta vozila, tada će biti vraćena vrednost *-1*. U ovom slučaju će biti prikazana odgovarajuća poruka korisniku.

```php
function fillVehicle($locations, $goodsPerLocations, $vehicleIndex, $vehiclesCapacity)
{
    $numberOfLocations = count($locations);
    $numberOfVehicles = count($vehiclesCapacity);
    if ($numberOfLocations <= 0 and $numberOfVehicles <= 0) {
        return -1;
    }

    $pathPerVehicle = array();
    for ($i = 0; $i < $numberOfVehicles; $i++) {
        array_push($pathPerVehicle, array());
    }

    $vehicleNumber = 0;
    $locationNumber = 0;
    $currentVehicleFreeCapacity = $vehiclesCapacity[$vehicleIndex[$vehicleNumber]];
    $sumOfGoodsPerLocations = array_sum($goodsPerLocations);
    while ($sumOfGoodsPerLocations) {
        if ($vehicleNumber == $numberOfVehicles) {
            return -1;
        }

        $goods = $goodsPerLocations[$locations[$locationNumber]];
        if ($goods <= $currentVehicleFreeCapacity) {
            $sumOfGoodsPerLocations-= $goodsPerLocations[$locations[$locationNumber]];
            $currentVehicleFreeCapacity-= $goodsPerLocations[$locations[$locationNumber]];
            array_push($pathPerVehicle[$vehicleIndex[$vehicleNumber]], $locations[$locationNumber]);
            $locationNumber+= 1;
        }
        else {
            $vehicleNumber+= 1;
            if ($vehicleNumber != $numberOfVehicles) {
                $currentVehicleFreeCapacity = $vehiclesCapacity[$vehicleIndex[$vehicleNumber]];
            }
        }
    }

    return $pathPerVehicle;
}
```

Slika 3. Konstrukcija rešenja

2.5 Funkcija cilja

Funkcija cilja ima najznačajniju ulogu u prihvatanju ili odbacivanju kandidata rešenja. U ovom slučaju funkcija cilja je minimizacija po kilometraži. Implementacija je prikazana

na Slici 4. Kao što je opisano, iz glavnog algoritma se poziva funkcija cilja da bi se odlučilo da li trenutnog kandidata rešenja prihvatiti ili odbaciti. Zatim se poziva funkcija nadležna za konstrukciju rešenja od čije povratne vrednosti zavisi nastavak izračunavanja u funkciji cilja. Ako je povratna vrednost *-1*, znači da rešenje za koje je potrebno izračunati funkciju cilja nije dopustivo odnosno da nije moguće spakovati robu u tom redosledu pa se i do glavnog algoritma takođe vraća vrednost *-1*. Ako je rešenje dopustivo, izračunava se vrednost funkcije cilja odnosno kilometraža koju je potrebno preći. Konkretno, iterira se kroz petlju po broju vozila i u svakoj iteraciji se izračunavaju sledeće vrednosti:

1. kilometraža između skladišta i prve lokacije u nizu za trenutno vozilo;
2. kilometraža od prve lokacije to poslednje lokacije redom za trenutno vozilo;
3. kilometraža od poslednje lokacije do skladišta;
4. ažurira se vrednost promenljive koja čuva ukupnu kilometražu za sva vozila.

Vrednosti koje prikazuju rastojanja u kilometrima između lokacija isporuke su predstavljene matricom. Rastojanja između skladišta i lokacija isporuke su predstavljena jednodimenzionalim nizom. O konstrukciji ovih podataka više u poglavlju 3. Nakon što su navedene vrednosti izračunate, glavnom algoritmu se vraća vrednost koja predstavlja kilometražu koju je potrebno preći tim kandidatom rešenja. Način procene da li će kandidat biti prihvaćen ili neće je objašnjeno u poglavlju 2.2.

```
function getResult($locations, $goodsPerLocations, $vehicleIndex, $vehiclesCapacity, $distanceBetweenLocations,
    $distanceDepotFromLocations)
{
    $k = fillVehicle($locations, $goodsPerLocations, $vehicleIndex, $vehiclesCapacity);
    for ($i = 0; $i < count($k); $i++) {
        $distance = 0;
        $cur = $k[$i];
        if (count($cur) != 0) {
            for ($j = 0; $j < count($cur) - 1; $j++) {
                $distance+= $distanceBetweenLocations[$cur[$j]][$cur[$j + 1]];
            }

            $distance+= $distanceDepotFromLocations[$cur[0]];
            $distance+= $distanceDepotFromLocations[$cur[count($cur) - 1]];
            array_push($k[$i], $distance);
        }
    }

    return $k;
}
```

Slika 4. Funkcija cilja

2.6 Prelaženje u naredno rešenje

Simulirano kaljenje je heuristika koja zahteva mehanizam prelaska iz trenutnog u naredno rešenje. Naredno rešenje je poželjno uzimati tako da ima sličnosti sa prethodnim jer je u tom slučaju veća verovatnoća da će se dobiti bolje rešenje od trenutnog nego da se izabere sasvim nasumično novo rešenje. Potencijalno rešenje je predstavljeno nizom indeksa lokacija. Ideja koja stoji iza algoritma prelaženje u naredno rešenje je da se izabere slučajan broj, koji predstavlja neku poziciju u nizu i da se zameni sa svojim susedom. Neka je N duzina niza i neka je K slučajan broj. Ukoliko postoji samo jedan član niza, jasno je da ne postoji više mogućnosti te prelaženje u naredno rešenje nema uticaja u ovom slučaju. Ako postoje dva člana niza, za vrednost slučajnog broja K se vraća nula te će biti izvršena zamena mesta ta dva elementa. Inače, ako postoji više od dva člana niza biće izabran slučajan broj K u opsegu [0, N-2] i biti izvršena zamena elemenata na pozicijama K, K+1. Nakon što je novo potencijalno rešenje generisano, pristupa se delu koji izračunava validnost rešenja, ukupnu kilometražu i poređenje sa trenutnim rešenjem.

2.7 Kriterijum zaustavaljanja

Kriterijum zaustavljanja može uticati na kvalitet krajnjeg rešenja u algoritmu simuliranog kaljenja. Postoji nekoliko vrsta kriterijuma među kojima su sledeći:

1. Tolerancija funkcije – algoritam se izvršava sve dok prosečna promena vrednosti objektivne funkcije u iteracijama ne bude manja od vrednosti funkcije tolaranicije [16];

2. Maksimalan broj iteracija – algoritam se zaustavlja kada broj iteracija dostigne maksimalan broj iteracija. Može se eksplicitno zadati maksimalna vrednost iteracija [16];

3. Dostignuta minimalna temperatura – algoritam se završava kada trenutna temperatura bude manja ili jednaka minimalnoj temperaturi. Postupak hlađenja i broj iteracija zavisi od faktora hlađenja i minimalne temperature;

4. Rešenje se ponavlja N puta – ukoliko se rešenje ponovilo N puta u poslednjih N iteracija smatra se da se došlo da dobrog rešenja i algoritam se zaustavlja;
5. Rešenje je pronađeno – u nekim slučajevima ne znači da svaka iteracija predstavlja jedno rešenje. Ukoliko se prvi put dobije rešenje algoritam se zaustavlja.

U slučaju problema rutiranja vozila ideja je da se pronađe najbolje moguće rešenje a ne samo rešenje jer postoji veliki broj rešenja zbog velike povezanosti lokacija. Zbog toga kriterijum zaustavljanja koji se ovde koristi je „Dostignuta minimalna temperatura" koji sa jedne strane ograničava broj iteracija koji će biti izvršen a sa druge strane garantuje da će biti nađeno bolje rešenje nego da je algoritam prekinut nekim drugim kriterijumom zaustavljanja kao što je „Rešenje je pronađeno".

2.8 Metoda zasnovana na totalnoj enumeraciji

Metoda zasnovana na totalnoj enumeraciji je jedan od načina za rešavanje problema rutiranja vozila. Ovaj pristup je poznatiji kao naivni pristup (eng. brute force) čija je ideja da proveri sve mogućnosti. Postoje i prednosti i mane s obzirom da navedeni problem pripada grupi problema NP-težak. Vreme izvršavanja algoritma zavisi od broja lokacija na ulazu. Za manji broj lokacija algoritam će se izavršiti u razumnom vremenu ali veći broj lokacija dovodi do kombinatorne eksplozije. Prednosti ovog algoritma u odnosu na heuristiku simulirano kaljenje je u tome što uvek daje tačno, odnosno najbolje rešenje. Međutim, mana je što za veći broj lokacija neće biti izvršen u razumnom vremenu te se savetuje korišćenje heuristike simulirano kaljenje. Pseudokod je dat na Slici 5. Na početku algoritma potrebno je definisati promenljivu *bestDistance* na nedostižnu vrednost i *goodPerVehicles* na negativan broj koju ukazuje da još uvek ne postoji rešenje. Nakon toga, iterira se po svim permutacijama indeksa lokacija u okviru čega se iterira po svim permutacijama indeksa vozila. Neophodno je iterirati po permutacijama vozila jer tačnost rezultata zavisi od načina pakovanja u vozila.

```
Algoritam UporednaMetodaZasnovanaNaTotalnojEnumeraciji
Ulaz: locations, distanceBetweenLocations, distanceDepotFromLocations, goodsPerLocations,
vehiclesIndex, vehiclesIndexCapacity
Izlaz: dvodimenzionalni niz – svaki podniz sadrži putanju za jedno vozilo

begin
   bestDistance := 99999999
   goodsPerVehicles := - 1
   for svaku permutaciju l po locations do
      if l != 0
         locations := sledeća_permutacija(locations)
      for svaku permutaciju k po vehiclesIndex do
         if k != 0
            vehiclesIndex := sledeća_permutacija(vehiclesIndex)
         pathPerVehicle := ispuni vozila robom za trenutnu permutaciju lokacija i vozila
         currentDistanceForAllVehicle := 0

         if vozila uspešno popunjena
            for svako vozilo i u nizu pathPerVehicle do
               if vozilo je napunjeno robom
                  currentDistance := izračunaj putanju za to vozilo
                  currentDistanceForAllVehicle += currentDistance

            if bestDistance > currentDistanceForAllVehicle
               bestDistance := currentDistanceForAllVehicle
               goodsPerVehiles := pathPerVehicle

   return goodsPerVehiles
end
```

Slika 5. Metoda zasnovana na totalnoj enumeraciji

Primer 2. *Neka su date lokacije A, B, C koje su jedna do druge i koje zahtevaju sledeće količine robe redom 50, 20, 10 i neka su data vozila X i Y sledećih kapaciteta redom 50 i 80. Ukoliko je trenutna permutacija vozila A, B, C i vozila X, Y rešenje koje bi se dobilo bi bilo:*

1. *Vozilo X dostavlja robu na lokaciju A;*
2. *Vozilo Y dostavlja robu na lokacije B i C.*

Ukupna kilometraza bi bila zbir kilometraza za vozilo X i Y. Međutim, ukoliko se ispita i druga permutacija vozila, odnostno Y, X rešenje bi bilo:

1. *Vozilo X nije iskorišteno;*
2. *Vozilo Y dostavlja robu na lokacije A, B i C.*

Ukupna kilometraža bi bila kilometraža vozila Y. Iz navedenog se može zaključiti da je drugi slučaj optimalniji.

Ukoliko je pakovanje u vozila po trenutnim permutacijama moguće za svako vozilo izračunava se kilometraža koje ono treba da pređe. Dobijena vrednost se dodaje na ukupnu kilometražu koja se računa za sva vozila. Nakon toga se proverava da li je trenutni kandidat bolji od već postojećeg. Ako je bolji, prihvata se kao trenutno najbolje rešenje, ako nije prelazi se na sledeću permutaciju. Na kraju algoritma dobiće se najbolje moguće rešenje koje podrazumeva najkraći put da se isporuči roba i vrati u skladište.

3. EKSPERIMENTALNI REZULTATI

3.1 Test primeri

Za testiranje aplikacije koriste se primeri koji su generisani sa lokacijama na području grada Beograda. Razlog tome je da bi se jasnije videla tačnost izračunavanja i prikaz putanja. U aplikaciji su na raspolaganju 6 grupa test primera i to sa 5, 10, 12, 15, 25 i 50 lokacija na koje je potrebno dostaviti robu. Takođe svaki test primer ima određeni broj vozila različitih kapaciteta. Svaki naredni test primer uključuje lokacije i vozila iz prethodnih test primera. Test primeri su sledeći:

Test primer 1.

	Naziv lokacije	**Kapacitet robe**
1.	Stjepana Ljubiše, Beograd, Serbia	10
2.	Prilaz, Beograd, Serbia	10
3.	Birčaninova, Beograd, Serbia	10
4.	Gandijeva, Beograd, Serbia	5
5.	Mije Kovačevića, Beograd	20

	Naziv vozila	**Kapacitet vozila**
1.	BMW	20
2.	Fiat	50

Test primer 2.

	Naziv lokacije	**Kapacitet robe**
6.	Vojvode Stepe, Beograd, Serbia	10
7.	Đorđa Stanojevića, Beograd, Serbia	5
8.	Nehruova, Beograd, Serbia	10

9.	Bulevar kralja Aleksandra, Beograd, Serbia	20
10.	Bulevar Mihajla Pupina, Beograd, Serbia	10

	Naziv vozila	**Kapacitet vozila**
3.	Mercedes	50

Test primer 3.

	Naziv lokacije	**Kapacitet robe**
11.	Resavska, Beograd, Serbia	5
12.	Beogradska, Beograd, Serbia	5

Test primer 4.

	Naziv lokacije	**Kapacitet robe**
13.	Kneza Danila 23, Beograd, Serbia	10
14.	Dalmatinska, Beograd, Serbia	20
15.	Španskih boraca, Beograd, Serbia	20

	Naziv vozila	**Kapacitet vozila**
4.	Ford	80

Test primer 5.

	Naziv lokacije	**Kapacitet robe**
16.	Kneza Miloša, Beograd, Serbia	30
17.	Kralja Milana, Beograd, Serbia	5
18.	Nemanjina, Beograd, Serbia	25
19.	Cvijićeva, Beograd, Serbia	10
20.	Mozerova, Beograd, Serbia	20
21.	Dimitrija Tucovića, Beograd, Serbia	10

22.	Mije Kovačevića, Beograd, Serbia	5
23.	Starine Novaka, Beograd, Serbia	5
24.	Mirijevo, Belgrade, Serbia	10
25.	Karaburma, Belgrade, Serbia	10

	Naziv vozila	**Kapacitet vozila**
5.	Opel	150

Test primer 6.

	Naziv lokacije	**Kapacitet robe**
26.	Francuska, Beograd, Serbia	10
27.	Kapetan-Mišina, Beograd, Serbia	5
28.	Visokog Stevana, Beograd, Serbia	10
29.	Kneginje Zorke, Beograd, Serbia	10
30.	Kneginje Ljubice, Beograd, Serbia	10
31.	Svetogorska, Beograd, Serbia	10
32.	Svetozara Markovića, Beograd, Serbia	10
33.	Zeleni Venac, Belgrade, Serbia	5
34.	Savska, Beograd, Serbia	5
35.	Narodnih heroja, Beograd, Serbia	10
36.	Tošin bunar, Beograd, Serbia	5
37.	Ugrinovačka, Beograd, Serbia	5
38.	Dobanovački put, Beograd, Serbia	10
39.	Vojni put 2, Beograd, Serbia	10
40.	Svetog Nikole, Beograd, Serbia	10
41.	Braće Ribnikar, Beograd, Serbia	5
42.	Pop Stojanova, Beograd, Serbia	5
43.	Ustanička, Beograd, Serbia	10
44.	Bulevar Oslobođenja, Beograd, Serbia	10
45.	Bulevar Nikole Tesle, Beograd, Serbia	10

46.	Bulevar Zorana Đinđića, Beograd, Serbia	10
47.	Dragoslava Srejovića, Beograd, Serbia	5
48.	Severni bulevar, Beograd, Serbia	10
49.	Crveni Krst, Belgrade, Serbia	10
50.	Bulevar despota Stefana, Beograd, Serbia	10

	Naziv vozila	**Kapacitet vozila**
6.	Subaru	250

Na navedenim test primerima biće prikazani i upoređeni rezultati dobijeni simuliranim kaljenjem i metodom zasnovanoj na totalnoj enumeraciji u poglavlju 3.3.

3.2 Matrica rastojanja

Matrica rastojanja sadrži rastojanja između svih lokacija isporuke. Rastojanja su izražena u kilometrima. Dimenzija matrice je NxN gde je N broj lokacija isporuke. Matrica se dobija od *Distance Matrix API-ja* koji je opisan u poglavlju 4.3. Na prvi pogled izgleda da je matrica simetrična, ali nije jer u određenim putanjama postoje jednosmerne ulice. Indeksi redova, odnosno kolona, odgovaraju indeksima lokacija isporuke. Trivijalno je jasno da su po dijagonali nula vrednosti jer je to polje koje je presek iste lokacije i u koloni i u vrsti. Matrica rastojanja je neophodan ulazni parametar u oba algoritma. Primer matrice je dat na sledećoj tabeli koja je izračunata na osnovu *Test Primer 1.*

0	11.421	2.648	11.404	1.014
12.173	0	9.264	4.836	12.022
2.533	8.753	0	8.736	3.304
10.302	5.014	7.393	0	10.151
0.743	11.705	3.034	11.688	0

Tabela 1. Matrica rastojanja

3.3 Informacije o izvršnom okruženju

Aplikacija je zasnovana na klijent-server arhitekturi. Klijentska strana omogućava interakciju sa korisnikom, kao što je unos, promena i brisanje podataka i prikazavanje rezultata. Serverska strana služi za opsluživanje zahteva klijenta i na serverskoj strani je implementirana logika aplikacije. S obzirom da je reč o veb aplikaciji, ista je postavljena (eng. host) na udaljeni server koja može biti pronađena na sledećem linku:

https://group-parachute.000webhostapp.com

Za korišćenje aplikacije sa udaljenog servera neophodan je samo pregledač (eng. browser) kao što je Chrome, Firefox, Opera, Safari, itd.

Aplikacija takođe može biti izvršena na lokalnom serveru koja u tom slučaju može biti preuzeta sa sledećeg linka:

https://github.com/mladenlazic/Master

Za korišćenje aplikacije na lokalnom serveru neophodno je podesiti lokalni server kao što je Wamp, EasyPHP, Xampp, itd. S obzirom da aplikacija koristi „Google Map API" neophodna je internet konekcija u oba slučaja.

Aplikacija koristi nekoliko tehnologija. Neke od njih su neophodne dok su neke korištene jer omogućavaju lakšu implementaciju, čitljiviji kôd i lakše razumevanje kôda. Na klijentskoj strani su korišćeni HTML5 za kreiranje elementa na stranici, CSS3 za dizajniranje stranice, JavaScript za razvijanje funkcionalnosti, AngularJS za komunikaciju između HTML-a i JavaScript-a, Bootstrap isključivo da bi se napravila

aplikacija koja je prilagodljiva većem broju rezolucija (eng. responsive) pa čak i mobilnim telefonima, jQuery za dinamičko kreiranje elemenata. Na serverskoj strani je korišćen PHP7 u kome su implementirani i algoritmi. Da bi aplikacija funkcionisala neophodni su interpreteri za navedene jezike. Za komunikaciju između klijenta i servera je korišćen *JSON* format.

Karakteristike računara su veoma bitne za izvršavanje aplikacije. Na lošijoj konfiguraciji računara izračunavanje za veći broj lokacija će znatno duže trajati nego na boljoj konfiguraciji. Najbitnija komponenta računara u ovom slučaju je centralni procesor (eng. CPU - Central Processor Unit) na kome se vrši izračunavanje. Ostale komponente koje su neophodne ali ne utiču znatno na izračunavanje i brzinu rada algoritma su RAM memorija (eng. Random Access Memory), HDD (eng. Hard Disk Drive), napojna jedinica (eng. PSU – Power Supply Unit), grafička kartica, i matična ploča. Da su karakteristike računara bitne ukazuje i sledeći primer.

Primer 3. *U primeru će biti prikazana razlika u vremenu izvršavanja algoritma na dva različita računara. Od karakteristika biće naveden CPU kao najvažnija komponenta u ovom slučaju, RAM i OS (eng. Operating System). Neka su računari označeni sa R1 i R2.*

R1: CPU i3 M370 2.4GHz, RAM 6GB, OS Windows 10 x64 Education
R2: CPU i7 7500U 3.5GHz, RAM 16GB, OS Ubuntu x64 16.04

U Tabeli 2. prikazano je vreme koje je potrebno da se navedeni test primeri izvrše na računarima R1 i R2 upotrebom algoritma simulirano kaljenje.

	R1	*R2*
Test primer 1	*41,55min*	*0,98min*
Test primer 2	*171,25min*	*4,77min*
Test primer 3	*179,41min*	*4,82min*
Test primer 4	*186,56min*	*4,98min*
Test primer 5	*323,56min*	*9,31min*
Test primer 6	*335,08min*	*9,76min*

Tabela 2. Vreme izvršavanja BF i SA

Vreme izvršavanja se kontroliše coolingRate faktorom i to tako da ne bude duže od 10 minuta na računaru R2. Rezultati testiranja su dati u sledećem poglavlju. Iz Tabela 2. se primećuje da konfiguracija računara u velikoj meri utiče na vreme izvršavanja algoritma.

Važno je napomenuti da je ovo NP-težak problem što značajno utiče na vreme izvršavanja algoritma i da to ne može biti prevaziđeno boljom konfiguracijom računara.

3.4 Tabelarno i dijagramski predstavljeni eksperimentalni rezultati

U ovom poglavlju biće upoređeni rezultati između algoritma simulirano kaljenje i metode zasnovanoj na totalnoj enumeraciji. Algoritmi su testirani na test primerima koji su opisani u poglavlju 3.1. na računaru R2. Vreme izvršavanja je kontrolisano u oba algoritama. U slučaju metode zasnovanoj na totalnoj enumeraciji, ukoliko vreme izvršavanja prelazi 3600 sekundi, algoritam će biti pekinut i biće ispisano do tada pronađeno rešenje. Kao što je navedeno u *Primeru 3,* tokom testiranja algoritma simulirano kaljenje vreme izvršavanja je kontrolisano *coolingRate* faktorom u odnosu na instancu koja se testira i to tako da ne bude duže od 10 minuta. Maksimalna vrednost za ovaj faktor je 0,999999 koja se smanjuje ako vreme izvršavanja prelazi zadatu granicu. Kao što se može zaključiti, vreme izvršavanja i ovog algoritma se povećava u zavisnosti od broja lokacija i broja vozila. U Tabeli 3. prikazane su vrednosti *coolingRate* faktora za svaku instancu.

	coolingRate
Test primer 1	0,999999
Test primer 2	0,999999
Test primer 3	0,999999
Test primer 4	0,999995
Test primer 5	0,99998
Test primer 6	0,9998

Tabela 3. Vrednosti coolingRate faktora po instancama

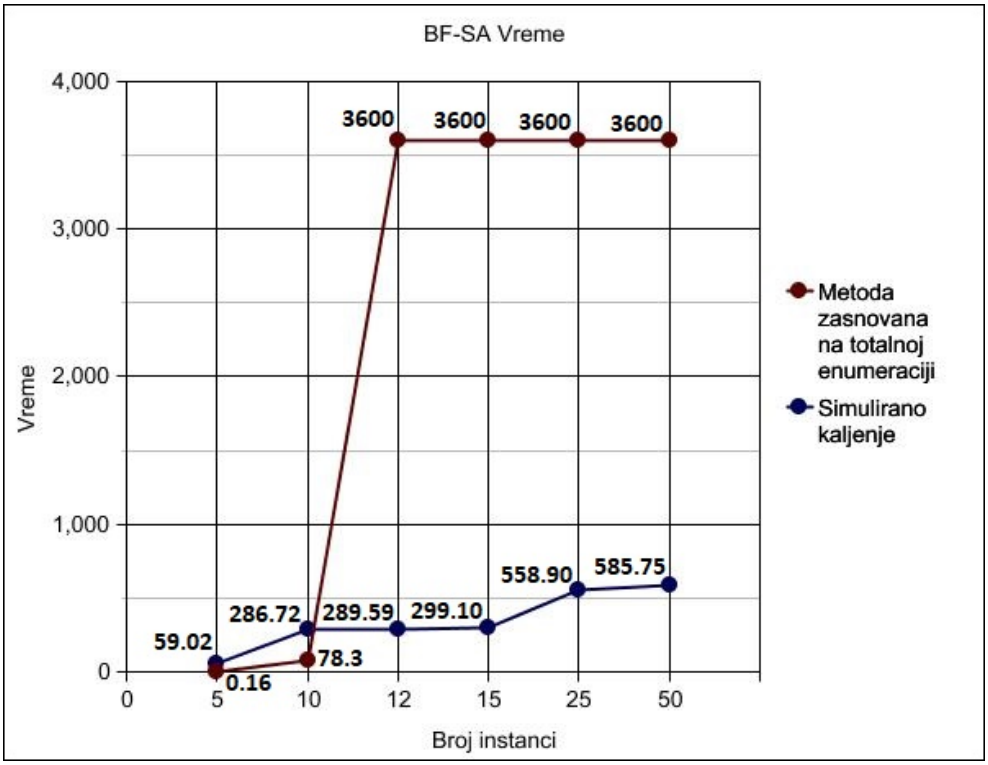
Metoda zasnovana na totalnoj enumeraciji se izvršava u razumnom vremenu za „Test primer 1" i „Test primer 2". Za ostale test primere algoritam prevazilazi razumno vreme izvršavanja pa je njegovo izvršavanje prekinuto na 3600 sekindi i upisano do tada pronađeno rešenje. Međutim, algoritam simulirano kaljenje daje tačno ili približno rešenje u odnosu na zadato vreme izvršavanja. Da bi poređenje bilo validno, algoritam simulirano kaljenje je pokrenut deset puta i čuva se minimalno kilometarsko rešenje, prosečno kilometarsko rešenje i prosečno trajanje izvršavanja algoritma u deset pokušaja. Rezultati su prikazani tabelarno i dijagramski.

	Instanca	Dimenzija	BF rešenje	BF vreme	SA rešenje (min)	SA prosečno rešenje	SA prosečno vreme
1.	Test primer 1	5	27.4km	0.16s	27.4km	27.43km	59,02s
2.	Test primer 2	10	52.8km	78.3s	52.8km	53.81km	286.72s
3.	Test primer 3	12	57.0km	3600s	57.2km	59.43km	289.59s
4.	Test primer 4	15	87.7km	3600s	53.45km	58.47km	299.10s
5.	Test primer 5	25	146.5km	3600s	80.82km	88.10km	558.90s
6.	Test primer 6	50	236.5km	3600s	164.0km	176.27km	585.75s

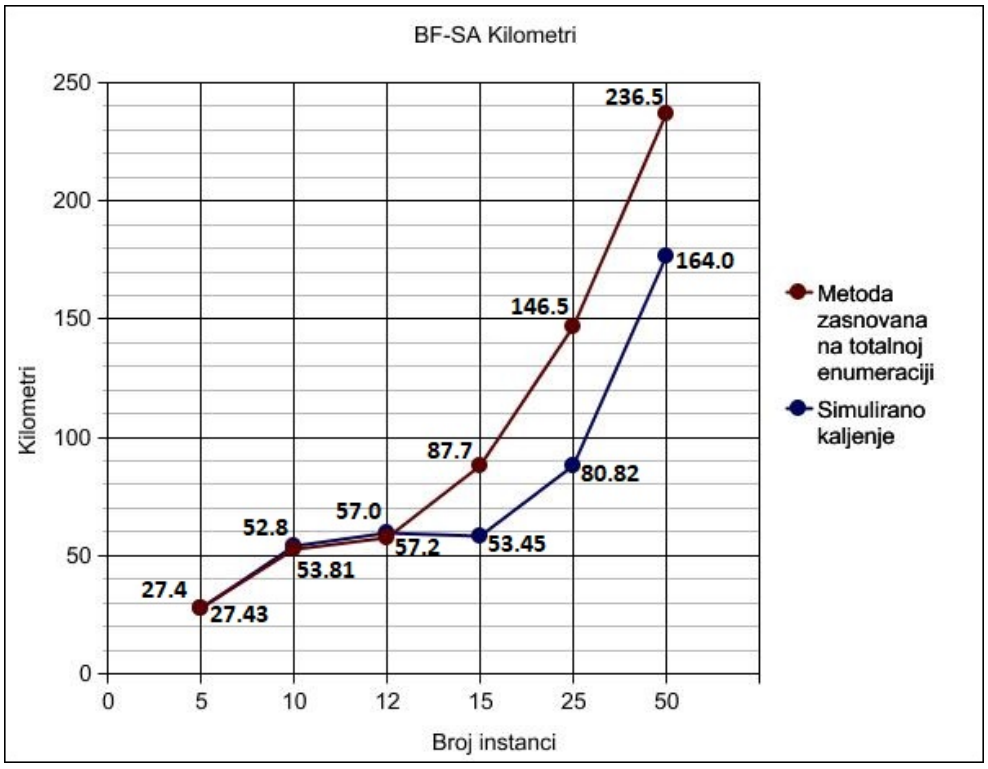
Tabela 4. BF i SA rezultati

Na test primeru „Test primer 1" se vidi iz Tabele 4, Dijagrama 1 i Dijagrama 2 da implementacija metode zasnovanoj na totalnoj enumeraciji daje bolje rezultate i u pogledu vremena izvršavanja i ukupne kilometraže ako se posmatra prosečna kilometraža u slučaju algoritma simulirano kaljenje jer u deset pokretanja simulirano kaljenje je takođe dalo tačno rešenje. Razlika u vremenu izvršavanja je 58,86s dok je razlika u kilometraži 0.03km što je zanemarivo na dobijene vrednosti. Ovo ponašanje je očekivano jer je simulirano kaljenje heuristički pristup. Već na „Test Primer 2" razlika u vremenu izvršavanja je nesto veća i iznosi 208,42s dok je razlika u kilometraži 1,01km ako se posmatra prosečno rešenje algoritma simulirano kaljenje jer je i u ovom slučaju u deset pokretanja algoritam dao tačno rešenje. Dakle, ukoliko je potrebno izračunati putanju za manji broj lokacija na ulazu, preporučuje se upotreba algoritma zasnovanog

na totalnoj enumeraciji odnosno za više od 10 ulaznih lokacija poželjno je koristiti algoritam simulirano kaljenje.



Dijagram 1. BF-SA-Vreme



Dijagram 2. BF-SA-Kilometri

4. IMPLEMENTACIJA GRAFIČKOG OKRUŽENJA ZA REŠAVANJE PROBLEMA RUTIRANJA VOZILA

4.1 Prikaz i način upotrebe grafičkog okruženja kroz slike

Za implementaciju grafičkog interfejsa koristi se nekoliko tehnologija kao što je *HTML5, CSS3, JavaScript, AngularJS* i *Bootstrap*.

HTML (eng. HyperText Markup Language) je standardizovani jezik koji služi za kreiranje veb stranica. Počeci nastanka *HTML* jezika datiraju još u ranim osamdesetim. Fizičar Tim Berners Lee radom u CERN-u predložio je i dizajnirao ENQUIRE, sistem koji su istraživači u CERN-u koristili za razmenu dokumenata. Tim Berners je kasnije 1989 godine došao na ideju korišćenja definisanog hipertekst sistema koji će se koristiti na internetu. Prva forma *HTML*-a se pojavila davne 1991 godine. Sadržala je 18 elemenata i bila je dosta ograničena što se tiče sadržaja. Važno je pomenuti da je prethodnik *HTML-a* bio *SGML* (Standard Generalized Markup Language) i da je *HTML* tada bio vrsta uprošćenog *SGML* jezika. Razvojem interneta, razvijali su se veb pretraživači kao i *HTML* koji je vremenom pored označavanja tekstualnog sadržaja, omogućavao označavanje multimedijalnog sadržaja i uključivanje skriptnih kodova.

CSS (eng. Cascading Style Sheets) je jednostavan jezik koji se koristi da se njime opisuju veb stranice i korisnički interfejsi napisani u *HTML*-u i *XHTML*-u, ali se takođe mogu opisati bilo koje vrste *XML* dokumenata uključujući i sam *XML*, *SVG* ili *XUL*. *CSS*, pored *HTML* predstavlja deo stranice koji daje vizuelnu strukturu i opis veb sajtova, veb aplikacija kao i aplikacija za mobilne telefone. Kotišćenjem *CSS*-a može se kontrolisati boja teksta, stil fontova, razmaci između elemenata, koja se slika ili boja pozadine koristi, izgled okvira(eng. layout), pozicioniranje elemenata na stranici za različite uređaje itd. Neke pogodnosti koje pruža *CSS*:

1. *CSS* čuva vreme – može biti napisan samo jednom i koristiti se u različitim *HTML* stranicama. Može se definisati izgled za svaki *HTML* element i kasnije koristiti u mnogim veb stranicama.
2. Lako održavanje – pravljenjem globalne izmene svi elementi na veb stranici će biti izmenjeni automatski.
3. Podržava veliki broj uređaja – *CSS* omogućava podešavanje sadržaja tako da bude prilagodljiv velikom broju uređaja (eng. responsive).

JavaScript pomaže klijentskoj strani u interakciji sa veb stranicama čije su funkcionalnosti prethodno definisane u *JavaScript* jeziku. Brendan Eich je razvio *JavaScript* programski jezik dok je radio za *Netscape Communication* korporaciju. *JavaScript* je najpopularniji skriptni jezik na Internetu kojeg podržavaju svi poznatiji pregledači (*Internet Explorer, Mozilla Firefox, Netscape, Opera, Safari, Chrome*). Neke primene *JavaScripta* su:

1. *JavaScript* može da dinamički unese kôd u *HTML* stranu.
2. *JavaScript* reaguje na događaje - *JavaScript* može da se podesi tako da se izvrši kad se nešto desi, npr. kad se strana učita ili kad korisnik klikne na *HTML* element.
3. *JavaScript* može da pročita ili ispiše *HTML* elemente - *JavaScript* može da pročita i da promeni sadržaj *HTML* elementa.
4. *JavaScript* se koristi za proveru ispravnosti unetih podataka - *JavaScript* može da se koristi za proveru ispravnosti podataka unetih u formu, da proveri ispravnost podataka pre nego što se pošalju serveru.
5. *JavaScript* se koristi za kreiranje kukija (eng. cookie) - *JavaScript* može da se koristi za čuvanje i vraćanje informacija o računaru posetioca, itd.

AngularJS je frejmvork (eng. framework) otvorenog kôda koji održava *Google* i zajednica pojedinačnih programera i korporacija za rešavanje mnogih izazova prilikom

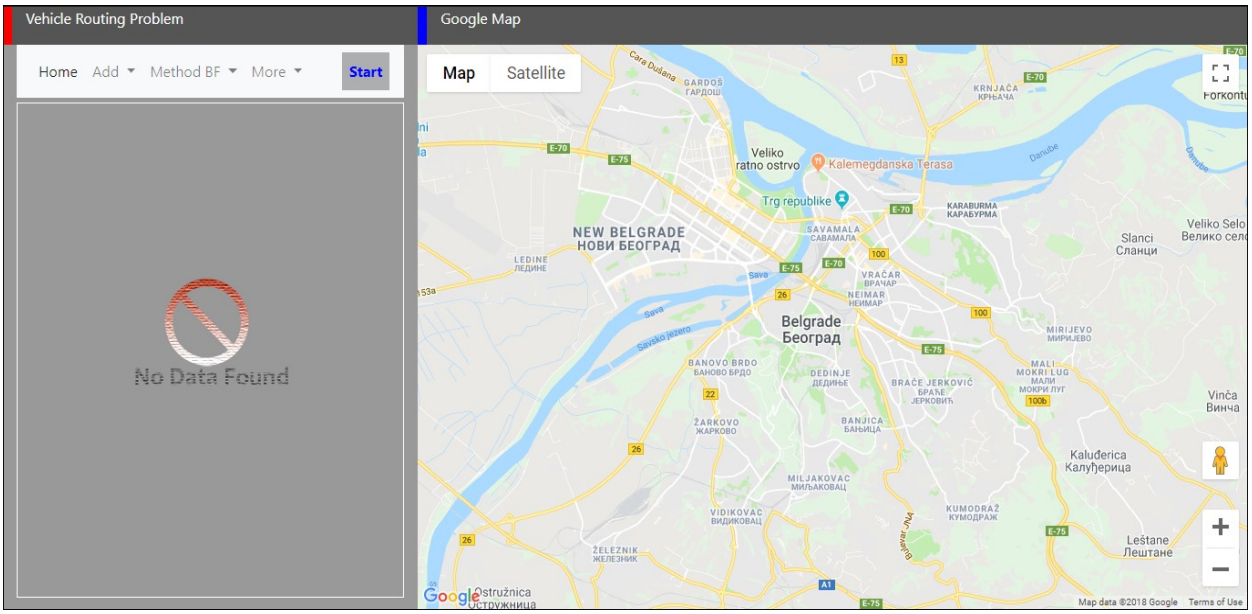
kreiranja jednostraničnih aplikacija. Ova biblioteka radi tako što prvo pročita *HTML* strane, koja ima ugrađene dodatne nestandardne tag atribute. Angular te atribute interpretira kao direktive da bi vezao ulazne ili izlazne delove stranice za model koji je predstavljen standardnim *JavaScript* promenjivim. Vrednosti tih *JavaScript* promenjivih se mogu ručno podesiti u kôdu, ili mogu biti preuzete od statičnih i dinamičnih *JSON* izvora [17]. *AngularJS* je dizajniran sa ciljevima da:

1. Razdvoji *DOM* (eng. Document Object Model) manipulacije od aplikacione logike.
2. Razdvoji klijentsku stranu aplikacije od serverske strane aplikacije. Ovo omogućava rad na razvoju obe strane paralelno, i omogućava ponovnu upotrebu obe strane.
3. Obezbedi strukturu za redosled razvoja aplikacije – od projektovanja korisničkog interfejsa, preko pisanja poslovne logike do testiranja.

Bootstrap predstavlja besplatni veb frejmvork (eng. framework) otvorenog kôda, za kreiranje veb sajtova i veb aplikacija. Baziran je na HTML i CSS šablonima za tipografiju, kreiranju formulara, dugmadi, navigacionim i ostalim komponentama interfejsa, kao i opcionim JavaSkript dodacima. Cilj *Bootstrap* frejmvorka je olakšavanje programiranja za veb. *Bootstrap* je frejmvork za veb aplikacije, tj. softverski frejmvork koji je dizajniran tako da podrži razvoj dinamičkih veb sajtova i veb aplikacija. Od maja 2015. ovaj projekat je najviše ocenjivan projekat na *GitHub*-u sa preko 107.000 ocena i 48.000 forkova (engl. fork). Postoji nekoliko verzija ovog frejmvorka od kojih svaka donosi novine. *Bootstrap 3* je podržan u najnovijim verzijama pregledača *Chrome, Firefox, Internet Explorer, Opera* i *Safari*. Dodatno je podržan i u *Internet Explorer*-u 8 i najnovijem *Firefox* izdanju sa dodatnom podrškom (engl. Firefox Extended Support Release - ESR). Od verzije 2.0 *Bootstrap* podržava prilagodljiv veb dizajn (engl. resposnive web design). Ovo znači da se izgled veb strane dinamički prilagođava, uzimajući u obzir tip uređaja koji se koristi (desktop računar, tablet, mobilni telefon). Počevši od verzije 3.0, *Bootstrap* je usvojio princip dizajna "mobilni uređaji pre svega", naglašavajući prilagodljiv veb dizajn kao podrazumevanu opciju [18]. Sav kôd vezan za

Bootstrap dostupan je i besplatan na *GitHub*-u. Programeri se podstiču da učestvuju u ovom projektu i da sami doprinesu njegovom razvoju [19].

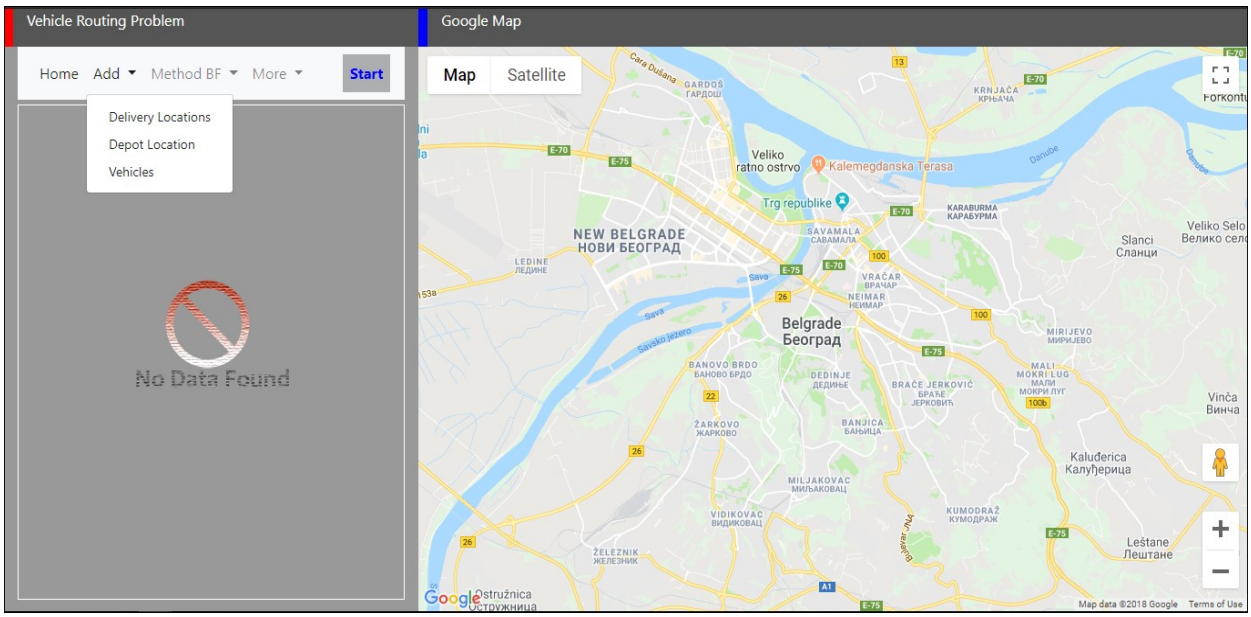
Početna strana aplikacije izgleda kao na slici 6. Na levoj strani nalazi se meni koji služi za upravljanje aplikacijom, dok je na desnoj strani integrisana mapa na kojoj se prikazuje rezultat i lokacije prilikom unošenja.



Slika 6. Početna strana aplikacije

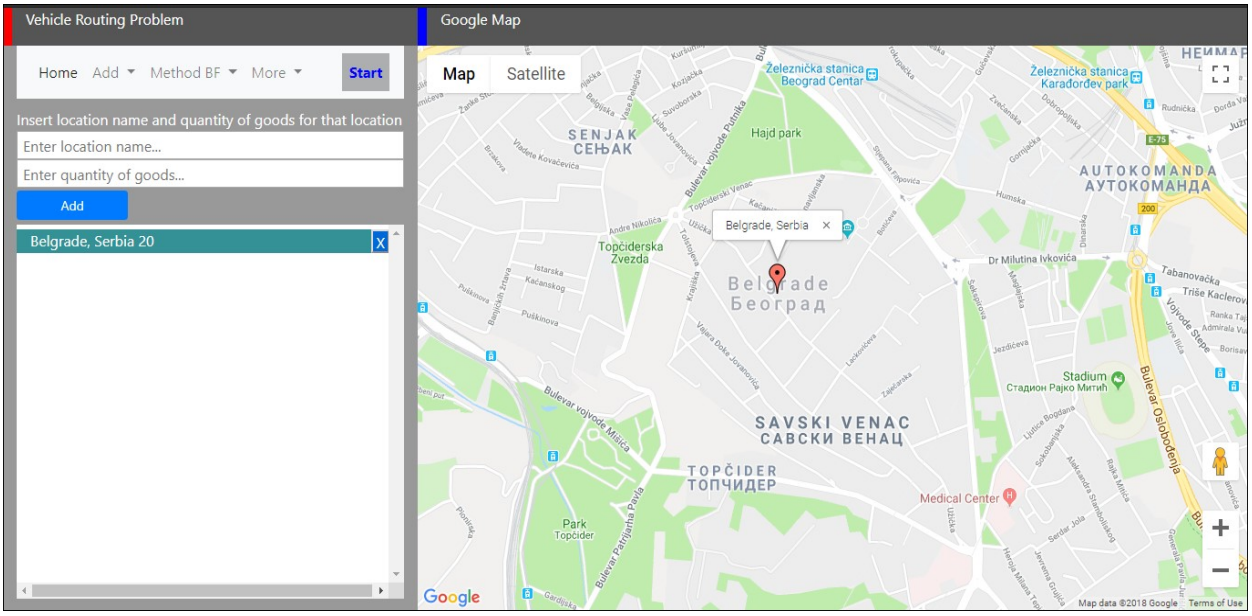
Dugme *Home* služi za osvežavanje, ponovno učitavanje aplikacije. Klikom na dugme *Add* pojavljuje se padajući meni na kome se može izabrati vrsta podatka koji se želi uneti kao na Slici 7.

Klikom na polje *Delivery Location* prikazuje se forma za unos podataka koji su relevantni za lokacije isporuke kao na Slici 8. To podrazumeva naziv lokacije i količinu robe koju je potrebno dostaviti na tu lokaciju. Prilikom unosa imena lokacije pojavljuje se auto dopuna od strane „Google Map API" koja sadrži deo ili celi naziv unete lokacije i koji pomaže u lakšem pronalaženju željene lokacije. Kada se klikne na plavo *Add* dugme, pre čuvanja podataka vrši se njihova provera. Ime lokacije mora da odgovara ponuđenom imenu od strane „Google Map API" što dalje omogućava dobijanje neophodnih informacija kao što su koordinate. Količina robe mora biti pozitivan broj i veći od nule.



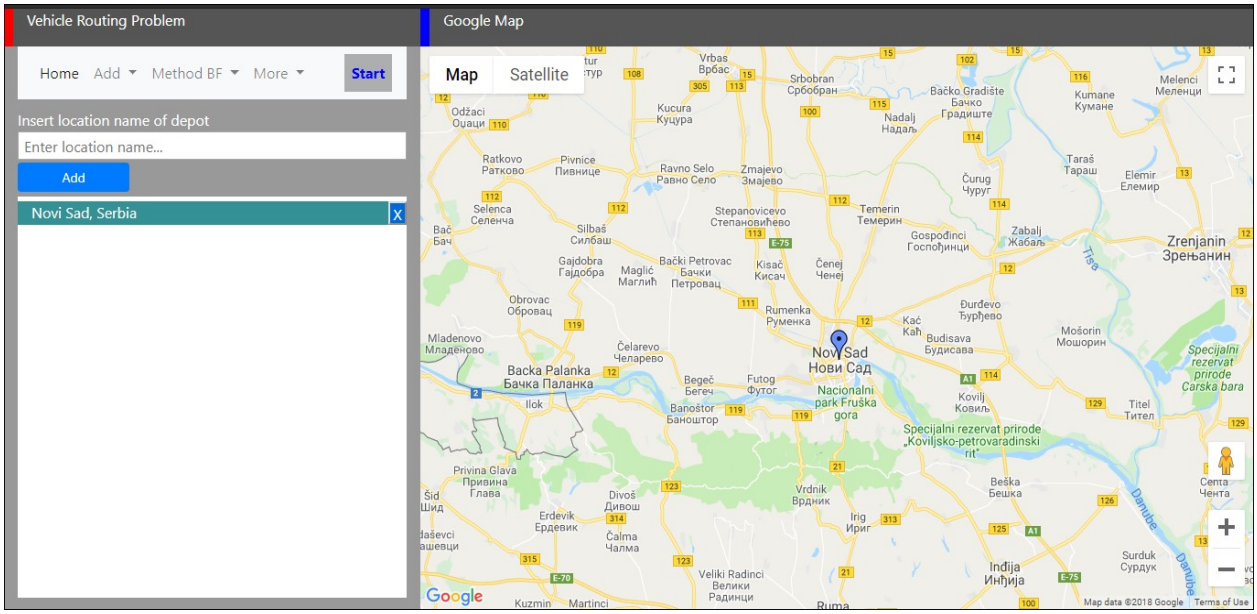
Slika 7. Padajući meni za izbor podataka za unos

U slučaju bilo kakve greške priliko unosa prikazuje se poruka sa odgovarajućim sadržajem nakon čega će polja biti obrisana i omogućen ponovni unos. Ukoliko su podaci ispravno uneti, na listi se pojavljuje ta lokacija koji sadrži ime lokacije i količinu robe koju je potrebno dostaviti na tu lokaciju. Takođe na mapi se pojavljuje marker crvene boje koji pokazuje tačno na unetu lokaciju isporuke. Klikom na marker prikazuje se ime lokacije. Na listi unetih lokacija, za svaku lokaciju postoji posebno dugme *X* koje služi za brisanje te lokacije sa liste i mape.



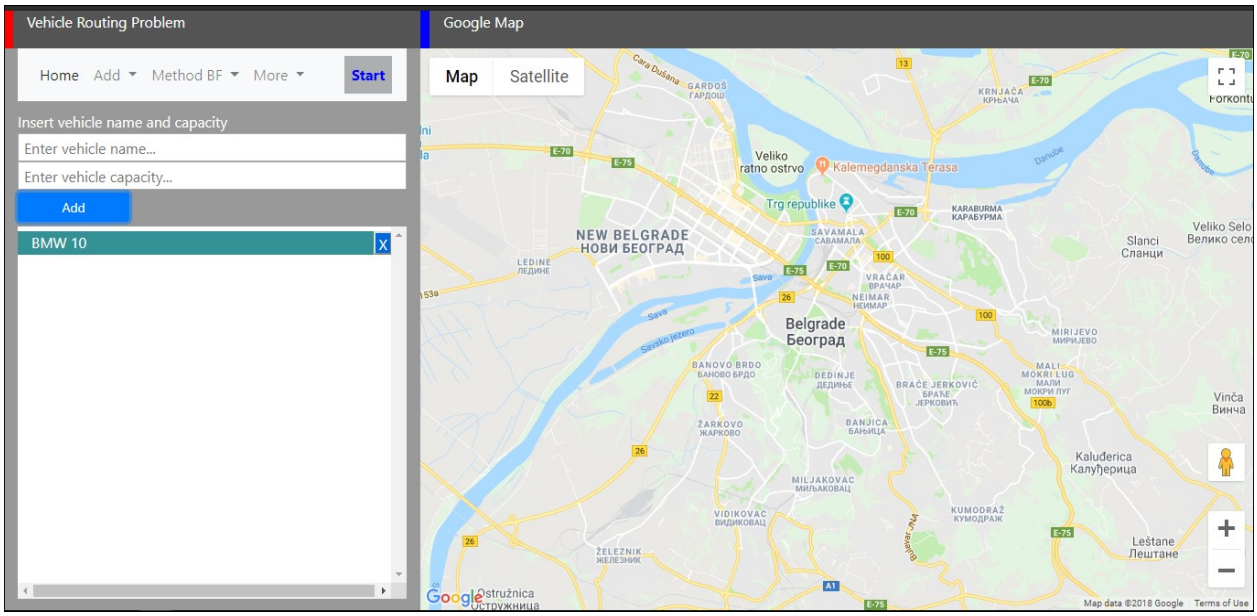
Slika 8. Unos lokacija isporuke

Klikom na polje *Depot Location* otvara se forma u kojoj je moguće uneti lokaciju skladišta kao na Slici 9. Pravila za unos i brisanje su ista kao u slučaju lokacija isporuke. Jedina razlika je što se u ovom slučaju pojavljuje plavi marker na mapi koji označava da je u pitanju skladište.



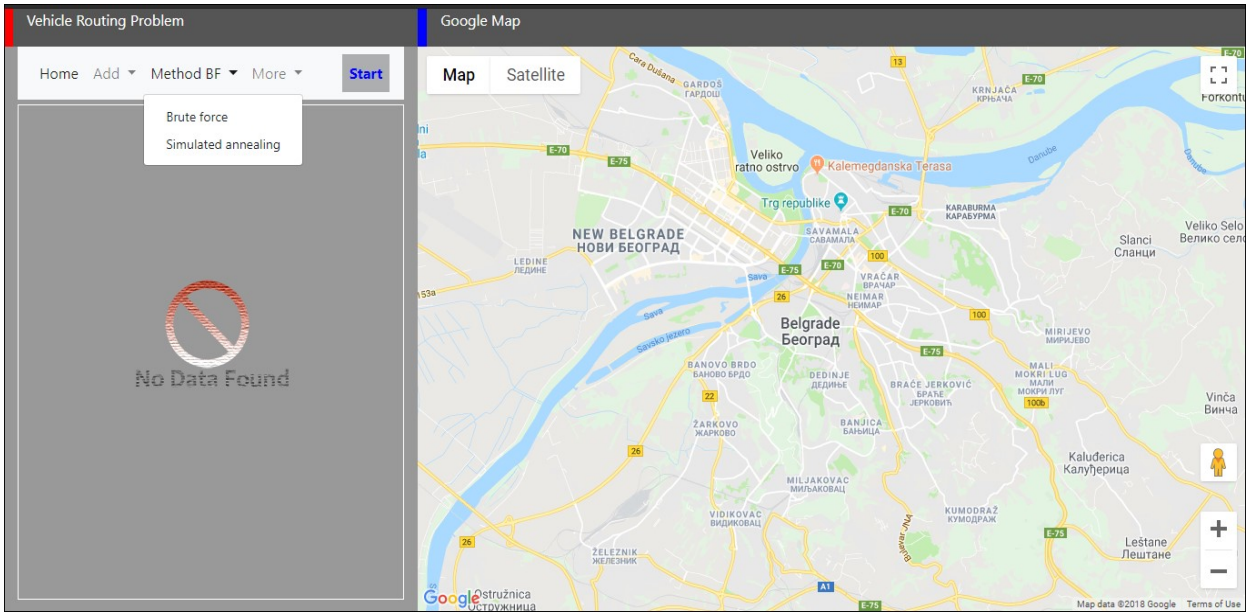
Slika 9. Unos skladišta

Klikom na polje *Vehicles* otvara se forma za unos vozila kao na Slici 10. Potrebno je uneti ime vozila i maksimalni kapacitet vozila. Ime vozila je proizvoljno. Kapacitet vozila mora biti pozitivan broj veći od nule. U slučaju greške prilikom unosa prikazuje se odgovarajuća poruka.



Slika 10. Unos vozila

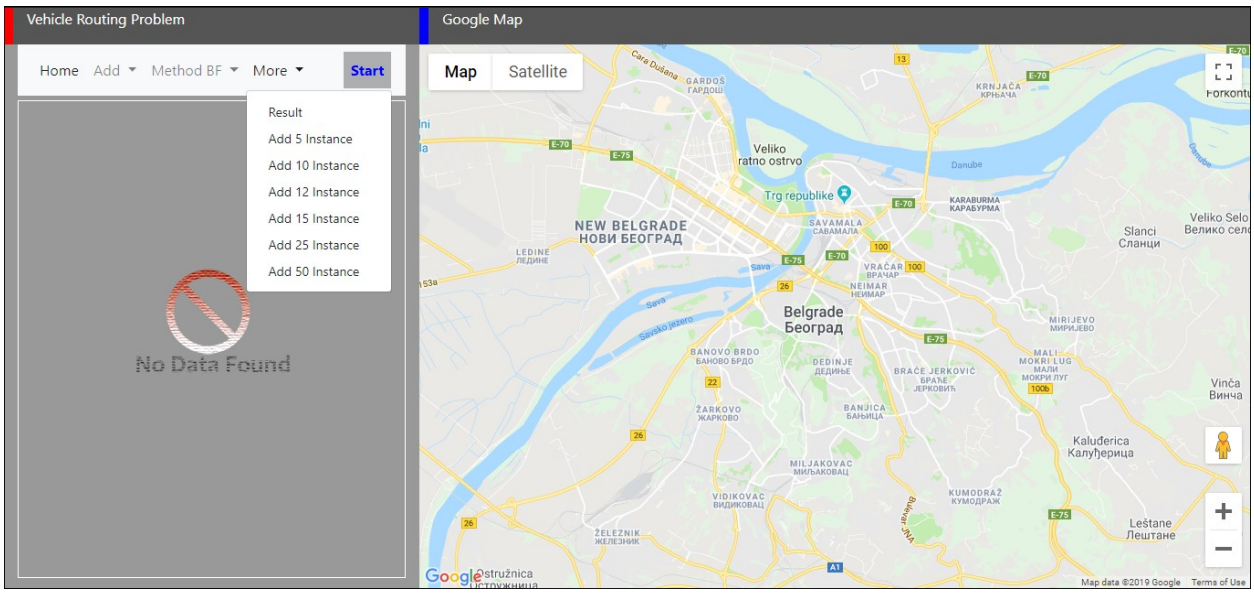
Sledeće polje u meniju je *Method*. Klikom na to polje otvara se padajući meni u kome možete izabrati algoritam koji će biti korišten u izračunavanju kao na Slici 11. U nastavku polja *Method* ispisana su početna slova naziva odabranog algoritma *BF*[1] ili *SA*[2]. Podrazumevana vrednost je *BF.*



Slika 11. Izbor algoritma

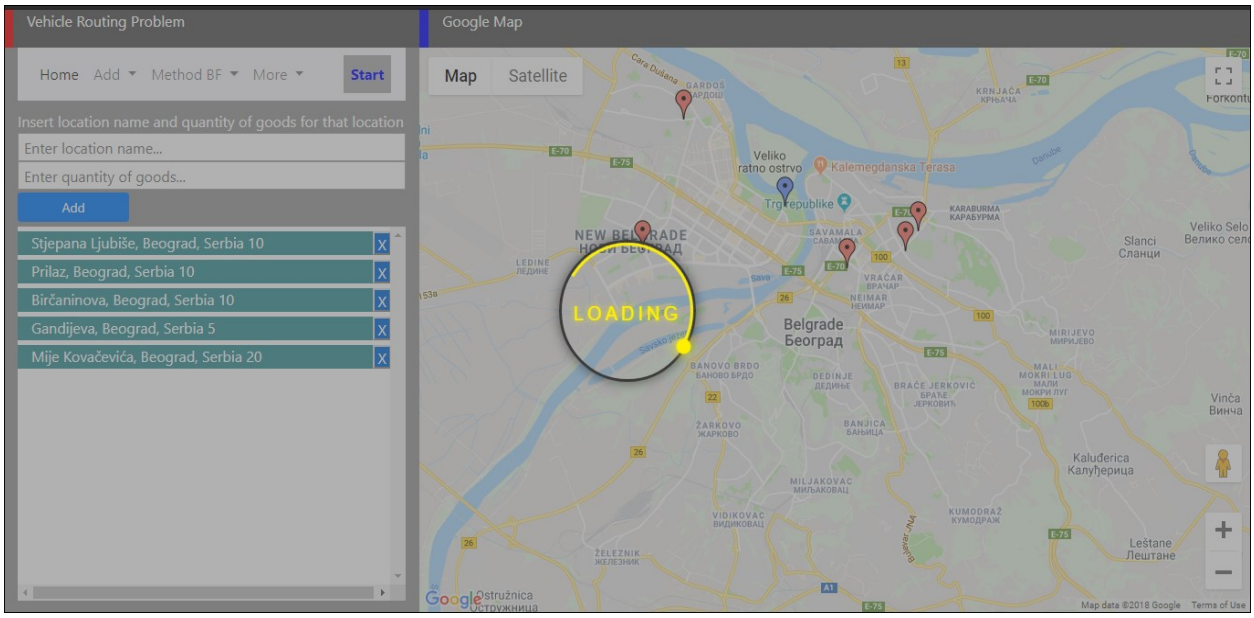
Zadnje polje u meniju je *More.* Klikom na to dugme se takođe otvara padajući meni u kome se može izabrati polje *Result, Add instance 5, Add instance 10, Add instance 12, Add instance 15, Add instance 25 ili Add instance 50* kao na Slici 12. Ukoliko se klikne da polje *Result* a rezultat ne postoji, odnosno nije bilo pokrenuto izvršavanje, prikazaće se poruka „*Result can not be found. Please insert data and press start".* Klikom na neko polje za dodavanje instance, biće dodata veštački uneta instanca za testiranje koje su opisane u poglavlju 3.1.

[1] BF – Brute Force
[2] SA – Simulated Annealing



Slika 12. Opcije klikom na *More*

Glavno dugme je dugme *Start.* Pre pritiska na to dugme moraju svi podaci biti unešeni, u suprotnom prikazaće se odgovarajuća poruka u odnosu na to koji podaci nisu uneti. Takođe mora biti ispoštovano da je ukupni kapacitet robe manji ili jednak od ukupnog kapaciteta vozila u suprotom se dobija poruka sa obaveštenjem da je potrebno korigovati unešene podatke. Klikom na dugme *Start* pojavljuje se *Loader*[3] sve dok se izvršavanje ne završi, kao na Slici 13.



Slika 13. Prikaz *Loader-a*

[3] Loader – animacija koja označava da je neko izvršavanje u toku i da je potrebno sačekati

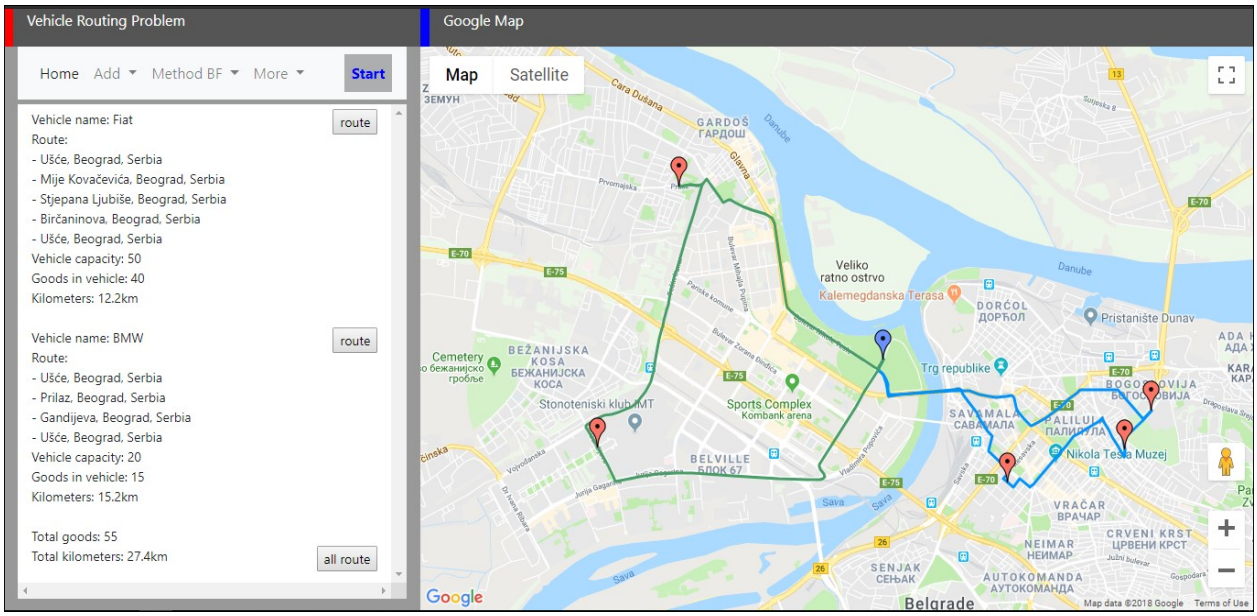
Nakon što je izvršavanje završeno, *Loader* nestaje, automatski se otvara *Result* polje u kome se nalaze informacije sa svako vozilo i to:

1. *Vehicle name* – ime vozila.
2. *Route* – lokacije u čijem ih redosledu vozilo treba da obiđe uključujuci i skladište.
3. *Vehicle capacity* – kapacitet vozila.
4. *Goods in vehicle* – količina robe koja je spakovana u vozilo.
5. *Kilometers* – kilometraža koje to vozilo treba da pređe.

Na kraju liste se nalaze podaci koji su relevantni za sva iskorišćena vozila:

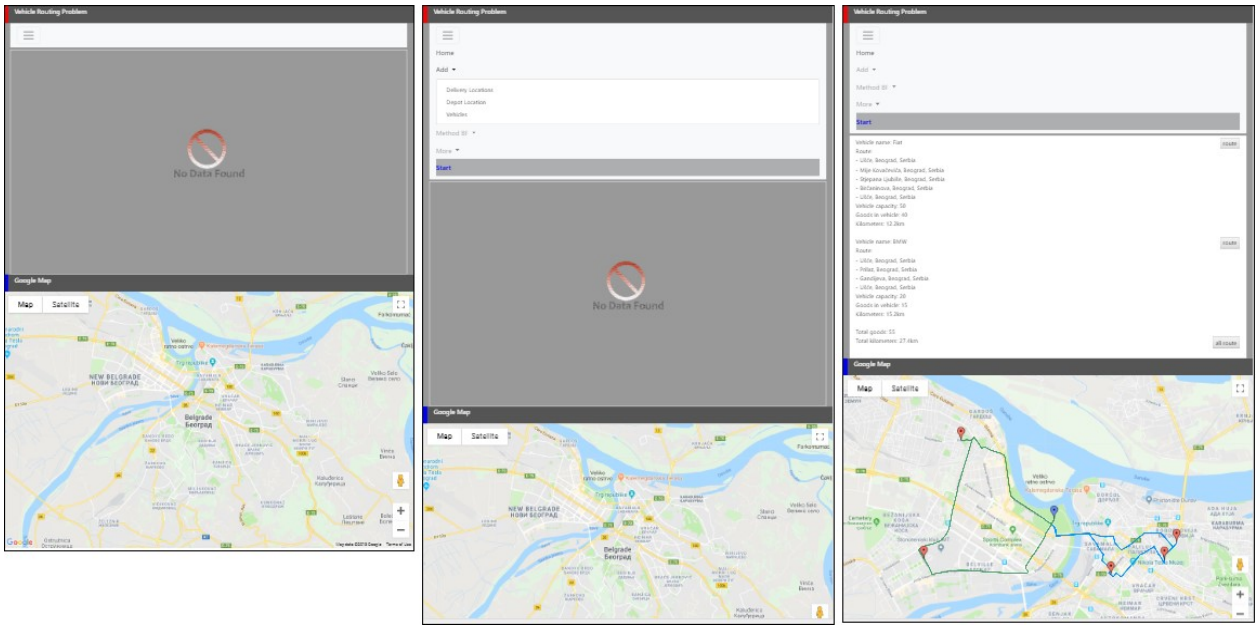
1. *Total goods* – ukupna količina robe koja se isporučuje.
2. *Total kilometers* – ukupna kilometraža koju je potrebno preći.

Takođe, na mapi se iscrtava putanja za svako vozilo i to različitom bojom da bi se lakše uočile. Ukoliko na ulazu ima veliki broj lokacija, i pored toga što je svaka putanja iscrtana različitom bojom, teško je ispratiti kretanje pojedinačnog vozila zbog preklapanja putanja. Zbog toga, pored imena svakog vozila postoji dugme *route,* koje suži za prikaz putanje samo tog vozila. Na kraju listi nalazi se dugme *all route* koje služi da se sve putanje ponovo iscrtaju. Sve navedeno je prikazano na Slici 14.



Slika 14. Prikaz rezultata

Vazno je napomenuti da je grafički interfejs prilagodljiv i mobilnim telefonima. Funkcionalnosti su iste a primer kako izlgeda na ekranima manje rezolucije je dat na Slici 15.



Slika 15. Prikaz dizajna za telefone

4.2 PHP

PHP (eng. Hypertext Preprocessor) je široko korišćen skript jezik otvorenog kôda koji je posebno pogodan u vezvoju veb programiranja i može biti ugrađen u *HTML*. U ovoj aplikaciji *PHP* se koristi na serverskoj strani. Koristi se verzija *PHP7* koja ima znatna ubrzanja u pogledu performansi u odnosu na prethodnu verziju *PHP5*. Osnovne tri oblasti gde se *PHP* koristi su sledeće:

1. *PHP na serverskoj strani.* Ovo je najtradicionalnije i osnovno polje *PHP*-a. Samo tri stvari su neophodne da se ovo obezbedi: *PHP* parser, veb server i veb pregledač. Potrebno je pokrenuti veb server sa konektovanim *PHP*-om. Zatim se izlaz *PHP* programa može videti koristeći veb pregledač.

2. *PHP kao komandna linija. PHP* skripta se može pokrenuti bez servera i veb pregledača. Potreban je samo *PHP* parser. Ova vrsta upotrebe je idealna za

skripte koje se izvršavaju koristeći *cron* (na Linux-u) ili *Task Sheduler* (na Windows-u). Ove skripte se mogu koristiti za jednostavne zadatke obrade teksta.

3. *PHP za pisanje desktop aplikacija. PHP* verovatno nije najbolji izbor jezika za kreiranje desktop aplikacija sa grafičkim korisničkim interfejsom, ali ako se *PHP* poznaje vrlo dobro, moguće je korišćenje *PHP* dodataka na klijentskoj strani aplikacija gde se može koristiti i *PHP-GTK*. Takođe, postoji mogućnost pisanja *cross* aplikacija na ovaj način. *PHP-GTK* je ekstenzija *PHP*-a koja nije dostupna u distribuciji.

PHP može biti korišćen na mnogim operativnim sistemima uključujući *Linux,* mnoge *Unix* varijante, *Microsoft Windows, macOS, RISC OS.* On danas ima podršku na mnogim veb serverima kao što su *Apache, IIS,* i mnoge druge. Dakle, sa *PHP*-om se ima sloboda izbora u odabiru operativnog sistema i veb servera. Takođe, može se birati između procedularnog ili objektno orijentisanog programiranje ili korišćenja oba istovremeno. Jedna od najvećih i najznačajnijih osobina *PHP*-a je podrška širokom opsegu baza podataka. Pisanje veb stranice za povezanom bazom podataka je neverovatno jednostavno koristeći jedno od specifičnih ekstenzija za bazu podataka kao što je *MySQL* ili koristeći apstraktnijeg nivoa kao što je *PDO.* PHP takođe podržava komunikaciju sa drugim servisima kao što *je LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM* (na Windows-u) i bezbroj drugih. [20] U ovoj aplikaciji *PHP* je korišćen za implementaciju metode zasnovane na totalnoj enumeraciji i simulirano kaljenje kao i za ostale funkcije koje su neophodne za izvršavanje tih algoritama. Primer upotrebe je dat na Slici 3 i Slici 4.

4.3 Opis upotrebe „Google Map API" tehnologije

„Google Map API" je skup različitih API-ja. Ima ih ukupno 14. U ovom radu korišćeni su neki od njih i to:

1. *Maps JavaScript API* – omogućava podešavanje i prikaz mape na veb stranici ili mobilnom uređaju. Takođe obezbeđuje četiri tipa mapa (*roadmap, satellite,*

hybrid, terrain) koji mogu biti modifikovani koristeći slojeve i stilove, kontrole i događaje, kombinacije servisa i biblioteke [21].

2. *Places API* – je servis koji vraća informacije o mestima koristeći HTTP zahtev. Mesta su definisana u ovom API kao ustanove, geografske lokacije ili istaknute tačke interesa [22].

3. *Distance Matrix API* - je servis koji računa udaljenost i potrebno vreme da se pređe relacija između svih početnih (eng. origins) i krajnjih (eng. destinations) tačaka pojedinačno. Povratna vrednost je informacija bazirana na predloženoj ruti između ovih tačaka. Ona sadrži redove sa udaljenostima izražene u željenom formatu (kilometri ili milje) i vremenima za svaki par lokacija [23].

4. *Directions API* – je servis koji iscrtava putanju između lokacija koristeći HTTP zahtev [24].

U nastavku je objašnjen svaki servis pojedinačno kako se koristi u aplikaciji.

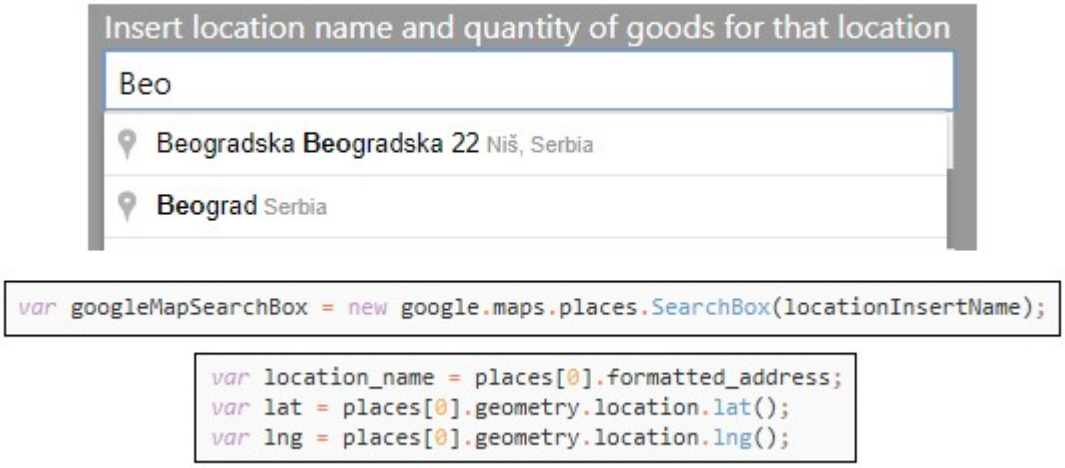
Maps JavaScript API je prvi koji je integrisan i zahtev prema njemu se upućuje pri učitavanju aplikacije jer se podrazumevano prikazuje mapa. Servis omogućava prikaz mape u obliku autokarte, satelitski, hibridni i zemljani. Pre zahteva, potrebo je definisati *HTML* element i dodeliti mu *id* koji se ovde prosleđuje da bi se znalo u kom *HTML* elementu se iscrtava mapa. Osnovni argumenti su *center* kojim se pozicionira početna pozicija mape i *zoom* kojim se podešava zum nad tom pozicijom. Korišćenje servisa je prikazano na Slici 16.

```
var map = new google.maps.Map(document.getElementById('googleMapContent'), {
    zoom: 12,
    center: {
        lat: 44.787197,
        lng: 20.457273
    }
});
```

Slika 16 – Maps JavaScript API

Places API se koristi za dobijanje punog imena lokacije prilikom unošenja iste, odnosno pojavljuje se lista lokacija koja sadrži dotadašnju unetu reč. Nakon toga

potrebno je dobiti koordinate za tu lokaciju. Da bi automatska dopuna bila omogućena i da bi se dobilo puno ispravno ime lokacije potrebno je *HTML* element za unos povezati sa *SearchBox-om* kao na slici. Nakon toga da bi se dobile koordinate (eng. *latitude, longitude*) za unetu lokaciju, potrebo je nad dobijenim objektom od *SearchBox-a* pozvati *getPlaces* nakon čega se konkretno tim poljima pristupa kao na Slici 17.



Slika 17. Places API

Distance Matrix API se koristi za dobijanje rastojanja i vremena koje je potrebno da se pređe to rastojanje između svih lokacija pojedinačno. Pre zahteva, potrebno je obezbediti dva niza lokacija, početne tačke (eng. origins) i krajnje tačke (eng. destinations). Lokacije su predstavljene koordinatama. Moguće je koristiti i dodatne argumente kao što su *transportOptions, drivingOptions, unitSystem, avoidHighways i avoidTolls.* Servis ima ograničenje po pitanju broja početnih i krajnjih tačaka i broja elemenata po zahtevu:

1. Maksimum je 25 početnih ili 25 krajnjih tačaka po zahtevu.
2. Maksimalno 100 elemenata po zahtevu.

S obzirom na to da je potrebno izračunati rastojanja između svih lokacija isporuke i između svih lokacija isporuke i skladišta, nekada je potrebno slati više od jednog zahteva.

Primer 5. *Neka je potrebno dobiti rastojanja između lokacija za Test Primer 6. S obzirom na to da ima 50 lokacija isporuke, to znaci da ima 2500 elemenata, rastojanje od svake lokacije do svake. Pošto je 100 elemenata dozvoljeno po zahtevu, upućuje se 25 zahteva za dobijanje rastojanja između lokacija isporuke. Dalje, potrebno je dobiti rastojanja između lokacija isporuke i skladišta. Ovde nije problem broj elemenata ali je problem broj lokacija gde je maksimum 25 a u ovom primeru ih ima 50. U tom slučaju biće upućena 2 zahteva. Ukupno je potrebno 27 zahteva samo za rastojanja između lokacija.*

Ako je potrebno slati više zahteva za dobijanje rastojanja, dobijene matrice se pakuju u jednu i izvršavanje aplikacije se nastavlja nad tom matricom. Povratna vrednost može biti u formatu JSON[4] ili XML. U ovom slučaju koristi se JSON format. Odgovor sadrži dosta podataka koji nisu releventni za ovu aplikaciju. Zbog toga je potrebno izdvojiti samo podatke o udaljenostima između svih lokacija izražene u kilometrima. Korišćenje servisa je prikazano na Slici 18.

```
var distanceBetweenLocations = distanceMatrixBetweenLocations(gmResponse);
service.getDistanceMatrix({
    origins: depot,
    destinations: locations,
    travelMode: 'DRIVING',
    unitSystem: google.maps.UnitSystem.METRIC,
}, function(gmResponse, gmStatus) {
    if (gmStatus !== 'OK') {
        alert("depot " + gmStatus);
    } else {
```

Slika 18. Distance Matrix API

Directions API se koristi za iscrtavanje putanja na mapi. Za svako vozilo je već određeno koje lokacije obilazi. Početna i krajnja tačka su uvek iste, a ta tačka je skladište. Sve ostale lokacije, odnosno lokacije isporuke se dodaju u niz *waypoints.* Putanje se iscrtavaju različitom bojom za svako vozilo podešavanjem *strokeColor* opcije. Ovaj servis iscrtava svoje markere nad svakom lokacijom ali je to ugašenom opcijom *suppressMarkers: true* jer su markeri već postavljeni pri unosu lokacija i obojeni odgovarajućim bojama u zavisnosti da li je u pitanju lokacija isporuke ili skladište. Korišćenje ovog servisa je prikazano na Slici 19.

[4] JSON – JavaScript Object Notation. JSON je sintaksa za čuvanje i razmenu podataka između pretraživača i servera. Sadržaj koji se šalje i prima može biti samo tekst.

```
function renderDirections(result, color) {

    var directionsDisplay = new google.maps.DirectionsRenderer({
        polylineOptions: {
            strokeColor: color,
        }
    });
    directionsDisplay.setDirections(result);
    directionsDisplay.setOptions({
        suppressMarkers: true
    });
    directionsDisplay.setMap(map);
    g_currentDirections.push(directionsDisplay);

}

function requestDirections(start, end, waypts, color) {
    var directionsService = new google.maps.DirectionsService;
    directionsService.route({
        origin: start,
        destination: end,
        waypoints: waypts,
        travelMode: google.maps.DirectionsTravelMode.DRIVING
    }, function(result) {
        renderDirections(result, color);
    });
}
```

Slika 19. Directions API

5. ZAKLJUČAK

U ovom radu je opisana implementacija dva algoritma za rešavanje problema rutiranja vozila koji ima veliku primenu u praksi. Ovaj problem pripada grupi NP-teških problema. Prvi algoritam, metoda zasnovana na totalnoj enumeraciji pripada grupi naivnih algoritama koji u ovom slučaju uvek daje najbolje moguće rešenje ali nije izvršiv u razumnom vremenu za veći broj ulaznih lokacija. Drugi algoritam, heuristika simulirano kaljenje prevazilazi problem prvog algoritma ali nije sigurno da uvek daje najbolje rešenje. Algoritmi su testirani na veštačkom skupu instanci veličine 5, 10, 12, 15, 25 i 50. Rezultati su upoređeni tabelarno i dijagramski gde se može uočiti da se prvi algoritam brže izvršava na prvom test primeru i daje bolji rezultat u odnosu na drugi algoritam ako se posmatra prosečna dobijena vrednost. Drugi test primer je prelazna tačka gde se prvi algoritam izvršava duže ali daje tačnije rešenje u poređenju sa drugim algoritmom koji se izvršava brže ali se ne garantuje uvek tačno rešenje. U ostalim test primerima prvi algoritam se ne izvršava u razumnom vremenu zbog pojave kombinatorne eksplozije, pa je zbog toga njegovo vreme izvršavanja ograničeno na 3600s. Takođe, u sklopu algoritama opisana je funkcija cilja, kriterijumi zaustavljanja, kodiranje rešenja i ostali detalji. Funkcija cilja služi za izračunavanje kilometraže koju je potrebno preći kandidatom rešenja. Kriterijum zaustavljanja je uslov na osnovu koga se algoritam završava cime se smatra da je algoritam dostigao dovoljno dobro ili tačno rešenje. U ovom slučaju izabran je kriterijum „Dostignuta minimalna temperatura". Kodiranje rešenja je način formiranja i čuvanja rešenja koje sadrži informacije koja su vozila iskorištena, koji je raspored obilaska lokacija isporuke za iskorištena vozila i koja je kilometraža potrebna da bi se sve lokacije posetile i vozila vratila u skladište. Implementirana je veb aplikacija koja pruža grafički korisnički interfejs. U tekstu je opisan način upotrebe tog interfejsa i implementacija značajnih delova. Za prikazivanje mape, putanja, računanja udaljenosti korišćen je *Google Map API.* Korišćen je *Maps JavaScript* servis koji omogućava prikaz mape, *Place* servis koji sadrži informacije o mestima, *Distance Matrix* servis koji računa udaljenost između početnih i krajnjih

lokacija i *Directions* servis koji služi za iscrtavanje putanja. Svi servisi koji su korišćeni iz ovog *API-ja* su detaljnije opisani u tekstu.

Ovaj rad bi se delimično mogao unaprediti u pogledu grafičkog korisničkog interfejsa i implementaciji algoritama. Grafički korisnički interfejs je implementiran da zadovolji potrebe predstavljenih algoritama. Poželjno je dodati mogućnost izmene unesenih podataka. Moguće je poboljšati sam dizajn glavnog menija. Aplikacija podrazumeva jedno skladište, poželjno je dodati mogućnost više skladišta. Takođe, poželjno je unaprediti algoritam u pogledu pakovanja robe. Omogućiti da više vozila može posetiti jednu lokaciju ukoliko jedno vozilo ne može da dostavi svu robu na tu lokaciju, odnosno ukoliko je najveći kapacitet vozila manji od najveće zahtevane količine robe na nekoj lokaciji. Trenutno je funkcija cilja pronalaženje najmanje kilometraže, može se dodati izbor između najmanje ukupne kilometraže ili najmanje potrebnog vremena da se obiđu sve lokacije. Takođe, rad bi se mogao delimično unaprediti izborom optimizovanijeg programskog jezika za implementaciju algoritama kao što je *C++.*

Doprinos ovoga rada je u izračunavanju optimalnih putanja za isporuku robe od skladišta do određenih lokacija isporuke gde optimalna putanja podrazumeva minimalnu putanju po kilometrazi. Glavna upotreba ove aplikacije je u trgovini za isporuku različite vrste robe do klijenata.

LITERATURA

[1] Dantzig GB, Ramser JH. The truck dispatching problem. 1959.

[2] Clarke G, Wright JV. Scheduling of vehicles from a central depot to a number of delivery points. 1964.

[3] Christofides N, Mingozzi A, Toth P. Exact algorithms for the vehicle routing problem based on the spanning tree and shortest path relaxations. 1981.

[4] Christofides N, Mingozzi A, Toth P. State-space relaxation procedures for the computation of bounds to routing problems. 1981.

[5] Laporte G, Desrochers M, Nobert Y. Two exact algorithms for the distance-constrained vehicle routing problem. 1984.

[6] Baldacci R, Christofides N, Mingozzi A. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. 2008.

[7] Fukasawa R, Longo H, Lysgaard J, Poggi de Araga´o M, Reis M, Uchoa E, Werneck RF. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. 2006.

[8] Gilbert Laporte, Paolo Toth, Daniele Vigo. Vehicle routing: historical perspective and recent contributions., March 2013.

[9] F. Glover. Future paths for integer programming and links to artificial intelligence. Computers and Operations Research, 1986.

[10] F. Glover. Tabu search-part I. ORSA journal on Computing. 1989.

[11] Pisinger and Ropke. A general heuristic for vehicle routing problems. 2007.

[12] Vidal et al. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. 2012.

[13] Adam N. Letchford Andrea Lodi. The traveling salesman problem, October 2007.

[14] Toth, P., Vigo, D., eds. The Vehicle Routing Problem. Monographs on Discrete Mathematics and Applications. 2002

[15] Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi. Optimization by simmulated annealing.

[16] Stopping Conditions for the Algorithm Simulated Annealing, https://www.mathworks.com/help/gads/how-simulated-annealing-works.html#bq2g2yi-25

[17] Angular, https://docs.angularjs.org/api

[18] Bootstrap, https://getbootstrap.com/docs/4.1/getting-started/introduction/

[19] Bootstrap GitHub repozitorijum, https://github.com/twbs/bootstrap

[20] PHP, https://secure.php.net/

[21] JavaScript Maps API, https://developers.google.com/maps/documentation/javascript/tutorial

[22] Places API, https://developers.google.com/places/web-service/intro

[23] Distance Matrix API, https://developers.google.com/maps/documentation/distance-matrix/intro

[24] Directinos API, https://developers.google.com/maps/documentation/directions/intro