GRASP

Seminarski rad u okviru kursa
Računarska inteligencija
Matematički fakultet

Jelena Kostadinović
Sofija Višnjić

February 3, 2026

Contents

1 Uvod

GRASP (Greedy Randomized Adaptive Search Procedure) je metaheuristički algoritam koji se često koristi za rešavanje problema kombinatorne optimizacije kod kojih egzaktne metode nisu praktične. Algoritam se sastoji od dve faze: konstrukcije pohlepnog, ali randomizovanog početnog rešenja i njegove optimizacije primenom lokalne pretrage. Tokom konstrukcione faze, izbor elemenata vrši se nasumično iz ograničene liste kandidata (RCL), čime se postiže raznovrsnost generisanih rešenja. Nakon toga se primenjuje lokalna pretraga radi unapređenja kvaliteta rešenja. U okviru ovog rada obrađene su dve modifikacije GRASP algoritma, primenjene na probleme optimizacije turističke rute i projektovanja emergency network mreže. Na taj način prikazana je fleksibilnost GRASP pristupa u različitim praktičnim scenarijima.

2 Formulacija problema "Emergency network"

U situacijama vanrednih dogadjaja (npr. elementarne nepogode, prekidi infrastrukture), neophodno je uspostaviti pouzdanu komunikacionu mrežu izmdedju više komunikacionih tačaka (čvorovi). Cilj je da se definise redosled obilaska povezivanja svih komunikacionih tačaka tako da se minimizuje ukupni trošak komunikacije a da se pri tom uzme u obzir pouzdanost svake tačke u mrezi.

2.1 Algoritam grube sile

Prikazani kod implementira brute force pristup za rešavanje problema optimizacije emergency network mreže, sa ciljem pronalaženja rute minimalne ukupne cene. Ulazni parametri funkcije su matrica troškova između čvorova i penal koji predstavlja nepouzdanost svakog čvora. Algoritam generiše sve moguće permutacije čvorova i za svaku izračunava ukupnu cenu rute, uključujući troškove prelaza između uzastopnih čvorova, penal za svaki čvor i trošak povratka na početni čvor. Tokom izvršavanja pamti se ruta sa najmanjom ukupnom cenom. Zbog vremenske složenosti O(n!), ovaj algoritam je primenljiv samo za mali broj čvorova i koristi se kao referentno rešenje za poređenje sa heurističkim metodama, poput GRASP-a.

```python
import itertools
import numpy as np

def brute_force_emergency(cost_matrix, reliability_penalty):
    n = len(cost_matrix)
    nodes = list(range(n))

    best_cost = float("inf")
    best_route = None

    for perm in itertools.permutations(nodes):
        cost = 0

        for i in range(n - 1):
            cost += cost_matrix[perm[i]][perm[i+1]]
            cost += reliability_penalty[perm[i]]

        cost += cost_matrix[perm[-1]][perm[0]]
        cost += reliability_penalty[perm[-1]]

        if cost < best_cost:
            best_cost = cost
            best_route = perm

    return best_route, best_cost
```

Figure 1: Brute force

2.2 GRASP algoritam za emergency network

U ovoj implementaciji primenjen je GRASP algoritam za optimizaciju emergency communication network mreže, pri čemu je cilj minimizacija ukupnih komunikacionih troškova uz uvažavanje penalizacije nepouzdanih čvorova. Algoritam se sastoji od tri glavne faze: inicijalne konstrukcije rešenja, polupohlepne konstrukcije korišćenjem ograničene liste kandidata i lokalne pretrage zasnovane na 2-opt heuristici.

```python
def initial_configuration(cost_matrix, reliability_penalty):
    config = [[], float("inf")]
    nodes = random.sample(
        list(range(1, cost_matrix.shape[0] + 1)),
        cost_matrix.shape[0]
    )
    nodes.append(nodes[0])  # povratak na start
    config[0] = nodes
    config[1] = communication_cost(cost_matrix, config, reliability_penalty)
    return config
```

Figure 2: Inicijalna konstrukcija rešenja

U konstrukcionoj fazi, početno rešenje se gradi iterativno tako što se sledeći čvor bira iz liste kandidata rangiranih prema lokalnom trošku komunikacije i penalima pouzdanosti. Stepen pohlepnosti kontroliše se parametrom greediness, koji se tokom iteracija postepeno smanjuje, čime se balansira između eksploracije i eksploatacije prostora rešenja. Nakon konstrukcije, na dobijeno rešenje se primenjuje 2-opt lokalna pretraga, kojom se proveravaju i unapređuju moguća preslaganja rute.

```python
def local_search_2_opt_emergency(cost_matrix, config, reliability_penalty):
    best = copy.deepcopy(config)
    improved = True

    while improved:
        improved = False
        for i in range(1, len(best[0]) - 2):
            for j in range(i + 1, len(best[0]) - 1):
                candidate = copy.deepcopy(best)
                #candidate[0][i:j] = reversed(candidate[0][i:j])
                candidate[0][i:j] = list(reversed(candidate[0][i:j]))
                candidate[1] = communication_cost(cost_matrix, candidate, reliability_penalty)

                if candidate[1] < best[1]:
                    best = copy.deepcopy(candidate)
                    improved = True
    return best
```

Figure 3: Lokalna pretraga

Ukupna cena rešenja računa se kao zbir komunikacionih troškova između uzastopnih čvorova i penalizacije svakog jedinstveno posećenog čvora. Algoritam se izvršava kroz unapred definisan broj iteracija ili se zaustavlja ranije ukoliko ne dođe do poboljšanja rešenja u određenom broju uzastopnih iteracija. Dobijeno rešenje predstavlja najbolju konfiguraciju pronađenu tokom celokupnog GRASP procesa.

```python
def grasp_emergency_network(cost_matrix, reliability_penalty, iterations=30):
    best_solution = initial_configuration(cost_matrix, reliability_penalty)
    no_improvement = 0

    for it in range(iterations):
        greediness = 0.9 - (it / iterations)

        candidate = restricted_candidate_list_emergency(cost_matrix, reliability_penalty, greediness)
        candidate = local_search_2_opt_emergency(cost_matrix, candidate, reliability_penalty)

        if candidate[1] < best_solution[1]:
            best_solution = copy.deepcopy(candidate)
            no_improvement = 0
        else:
            no_improvement += 1

        print(
            f"Iteration {it+1}: Candidate = {candidate[1]:.2f}, Best = {best_solution[1]:.2f}"
        )

        if no_improvement >= 15:
            print("Stopping early: no improvement in last 15 iterations.")
            break

    return best_solution
```

Figure 4: GRASP

2.3 Eksperimentalni rezultati za Algoritam grube sile

Brute-force algoritam testiran je na više instanci problema emergency network mreže sa različitim brojem čvorova i vrednostima penalizacije. Za svaku instancu algoritam je razmatrao sve moguće permutacije čvorova i pronašao rutu sa minimalnom ukupnom komunikacionom cenom, čime je garantovano pronalaženje globalno optimalnog rešenja. Testovi su sprovedeni nad mrežama sa 3 do 6 čvorova, pri čemu je uočen značajan porast složenosti sa povećanjem broja čvorova. Dobijeni rezultati korišćeni su kao referentna osnova za poređenje sa rezultatima GRASP algoritma. Zbog faktorske vremenske složenosti, primena brute-force pristupa ograničena je na male instance problema

Test	Broj čvorova	Optimalna ruta	Ukupna cena
BF1	3	(0, 1, 2)	57
BF2	4	(0, 1, 2, 3)	48
BF3	4	(0, 1, 2, 3)	42
BF4	5	(0, 1, 2, 3, 4)	64
BF5	6	(0, 1, 4, 3, 2, 5)	97

Table 1: Rezultati brute-force algoritma za emergency network problem

2.4 Eksperimentalni rezultati za GRASP

GRASP algoritam testiran je na istim malim instancama problema kao i brute-force pristup, ali i na znatno većim mrežama, kod kojih egzaktni algoritmi nisu primenljivi. Algoritam u svakoj iteraciji generiše novo polupohlepno rešenje i unapređuje ga lokalnom 2-opt pretragom, pri čemu se prati najbolje do tada pronađeno rešenje. Na manjim instancama GRASP postiže rešenja bliska optimalnim, dok na većim instancama pokazuje stabilno ponašanje i mogućnost ranog zaustavljanja kada ne dolazi do daljih poboljšanja. Dobijeni rezultati potvrđuju efikasnost GRASP algoritma u rešavanju većih instanci emergency network problema.

Test	Broj čvorova	Najbolja pronađena cena
GRASP1	4	54.00
GRASP2	5	94.36
GRASP3	6	29.00
GRASP4	12	465.55
GRASP5	13	530.35
GRASP6	25	1052.13

Table 2: Rezultati GRASP algoritma za emergency network problem

Tokom izvršavanja algoritma primećeno je da se kvalitet rešenja stabilizuje nakon određenog broja iteracija, što je omogućilo primenu kriterijuma ranog zaustavljanja u slučaju izostanka poboljšanja.

3 Formulacija problema "Optimizacija turističke rute" (Sightseeing Optimization Problem)

Problem optimizacije turističke rute (Sightseeing Optimization Problem) sastoji se u pronalaženju redosleda obilaska turističkih znamenitosti unutar jednog grada tako da ukupna dužina rute bude minimalna. Svaka znamenitost modelovana je kao čvor u potpunom grafu, dok su težine grana definisane kao rastojanja između parova lokacija. Cilj problema je da se pronađe permutacija svih čvorova koja minimizuje ukupnu pređenu udaljenost.

Ovaj problem predstavlja specijalni slučaj problema trgovačkog putnika (Traveling Salesman Problem – TSP), koji je poznat kao NP-težak problem. Zbog kombinatorne prirode problema, broj mogućih rešenja raste faktorijelno sa brojem znamenitosti, što onemogućava primenu egzaktnih metoda za realistične instance problema.

3.1 Algoritam grube sile

Brute-force pristup rešavanju sightseeing optimization problema zasniva se na generisanju svih mogućih permutacija znamenitosti i izračunavanju ukupne dužine rute za svaku od njih. Najbolje rešenje se bira kao ono sa minimalnom ukupnom udaljenošću.

Iako ovaj pristup garantuje pronalaženje globalno optimalnog rešenja, njegova vremenska složenost iznosi $O(n!)$, što ga čini praktično neupotrebljivim već za relativno mali broj znamenitosti. U praksi se brute-force algoritam može koristiti isključivo za male test instance i kao referentni metod za poređenje sa heurističkim algoritmima.

3.2 GRASP algoritam za optimizaciju turističke rute

Za rešavanje sightseeing optimization problema u ovom radu primenjen je GRASP (Greedy Randomized Adaptive Search Procedure) algoritam. GRASP se sastoji od iterativnog generisanja početnih rešenja korišćenjem polupohlepne strategije, nakon čega se svako rešenje dodatno unapređuje lokalnom pretragom.

U konstrukcionoj fazi algoritma, početna turistička ruta se formira izborom sledeće znamenitosti iz ograničene liste kandidata (Restricted Candidate List – RCL). Kandidati se rangiraju na osnovu lokalnog doprinosa ukupnoj dužini rute, dok se izbor vrši nasumično, čime se obezbeđuje raznovrsnost generisanih rešenja. Stepen pohlepnosti reguliše se odgovarajućim parametrom, koji omogućava balansiranje između determinističkog i slučajnog ponašanja algoritma.

Nakon konstrukcije početne rute, primenjuje se lokalna 2-opt pretraga, kojom se iterativno ispituju i zamenjuju parovi ivica rute u cilju smanjenja ukupne udaljenosti. Lokalna pretraga se ponavlja sve dok ne dođe do daljeg poboljšanja rešenja, čime se obezbeđuje lokalni optimum.

Tokom celokupnog procesa, algoritam pamti najbolje pronađeno rešenje kroz sve iteracije. Završno rešenje predstavlja najkraću turističku rutu pronađenu GRASP procedurom.

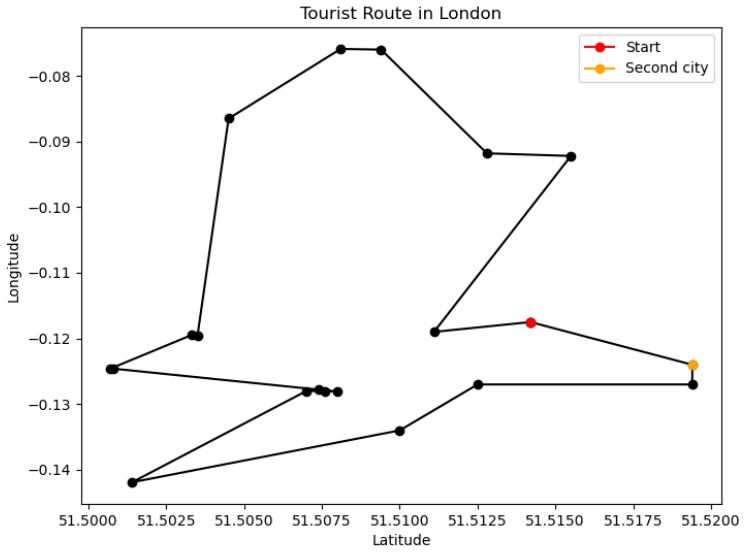

Figure 5: GRASP

3.3 Eksperimentalni rezultati za GRASP

3.3.1 Uticaj parametra greediness

Rezultati pokazuju da ekstremne vrednosti parametra greediness dovode do lošijih rešenja. Niske vrednosti uzrokuju preveliku randomizaciju, dok visoke vrednosti dovode do previše pohlepnog ponašanja algoritma. Najbolji rezultati postižu se za srednje vrednosti greediness parametra, čime se postiže balans između eksploracije i eksploatacije prostora rešenja.

Test	Greediness	Broj iteracija	Najbolja dužina rute
G1	0.1	50	12.43
G2	0.3	50	11.98
G3	0.5	50	11.42
G4	0.7	50	11.71
G5	0.9	50	12.05

Table 3: Uticaj parametra greediness na kvalitet rešenja

3.3.2 Uticaj broja iteracija

Povećanjem broja iteracija dolazi do poboljšanja kvaliteta rešenja, ali nakon određenog broja iteracija napredak postaje zanemarljiv. Ovo ukazuje na mogućnost primene kriterijuma ranog zaustavljanja, čime se dodatno optimizuje vreme izvršavanja algoritma.

Test	Broj iteracija	Najbolja dužina rute
I1	10	12.14
I2	25	11.72
I3	50	11.42
I4	100	11.39

Table 4: Uticaj broja iteracija na kvalitet GRASP rešenja

3.4 Eksperimentalni rezultati poredjenja dva algoritma i analiza

GRASP algoritam testiran je na različitim instancama sightseeing optimization problema, koje uključuju skupove od dvadeset i više turističkih znamenitosti za različite gradove. Eksperimentalni rezultati pokazuju da algoritam uspešno pronalazi smislene i kratke rute, koje su znatno bolje od nasumičnih početnih rešenja.

Poređenjem sa brute-force algoritmom na malim instancama potvrđeno je da GRASP daje rešenja bliska optimalnim, dok za veće instance predstavlja jedini praktično primenljiv pristup. Vizuelizacija dobijenih ruta dodatno potvrđuje kvalitet rešenja, pri čemu je uočeno da algoritam ne garantuje globalni optimum, ali postiže dobar kompromis između kvaliteta rešenja i vremena izvršavanja.

Algoritam	Broj znamenitosti	Dužina rute	Vreme izvršavanja
Brute force	8	5.32	12.4 s
GRASP	8	5.41	0.03 s

Table 5: Poređenje brute-force i GRASP algoritma za sightseeing optimization problem

3.5 Zaključak

Na osnovu sprovedenih eksperimenata može se zaključiti da je GRASP algoritam veoma pogodan za rešavanje problema optimizacije turističkih ruta. Njegova fleksibilnost, mogućnost skaliranja i relativno jednostavna implementacija čine ga efikasnim alatom za praktične primene, gde egzaktni algoritmi nisu izvodljivi. U poređenju sa brute-force pristupom, GRASP omogućava rešavanje znatno većih instanci problema uz prihvatljiv gubitak optimalnosti.

References