Univerzitet u Banjoj Luci
Prirodno-matematički fakultet
Studijski program Matematika i informatika

k-set cover problem

Operaciona istraživanja

Studenti:

David Sabljić
Armin Kočanović

Predmetni profesor:

dr Marko Đukanović, doc.

Banja Luka, februar 2024.

Sadržaj

Uvod 3
Opis problema 4
Pohlepni algoritam 5
Metoda promjenljivih okolina (VNS) 6
Cjelobrojno linearno programiranje 9
Eksperimentalni rezultati 10
Zaključak 15
Literatura 16

Uvod

U ovom seminarskom radu istražujemo k-set cover problem (SkCP) i načine rješavanja datog problema.

Prvo dolazi generalni opis problema, pa se fokusiramo na implementaciju pohlepnog i genetskog algoritma kao i implementacije metode cjelobrojnog linearnog programiranja za k-set cover problem.

Na kraju su priloženi rezultati testiranja date tri implementacije zajedno sa zaključkom.

Cilj je bolje razumijevanje discipline operacionih istraživanja kroz primjenu SkCP, kao i, kroz implementaciju metoda za rješavanje problema i eksperimenata koji su izvođeni, dubljem upoznavanju sa tim metodama i preformansih svaki od njih.

Opis problema

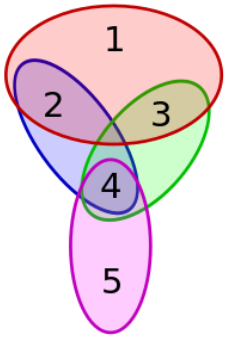
Prije opisivanja SkCP, treba da opišemo set cover problem (SCP). SkCP je posebna verzija SCP problema.

Neka je U univerzum elemenata, a S skup podskupova univerzuma U. Traži se minimalni skup podskupova $C \subseteq S$ takav da je unija svih podskupova iz C jednaka univerzumu U.

Ako imamo:

- univerzum $U = \{ 1, 2, 3, 4, 5 \}$,
- skup podskupova $S = \{ \{1, 2, 3\},\ \{2, 4\},\ \{3, 4\},\ \{4, 5\} \}$,

jedno optimalno rješenje može biti $C = \{ \{1, 2, 3\},\ \{4, 5\} \}$.



Slika 1: primjer instance za SCP

SkCP proširuje koncept SCP-a tako da uvodi dodatni uslov: svaki element mora biti **pokriven barem k puta.**

Imamo isti univerzum elemenata U i skup podskupova S, ali sada želimo pronaći minimalni skup $C \subseteq S$ takav da svaki element iz U bude pokriven barem k puta.

Ako imamo SkCP s $k = 2$, to znači da svaki element iz U mora biti pokriven najmanje dva puta. Razlika u odnosu na standardni SCP je ta što se u SkCP-u traže rješenja koja osiguravaju minimalno pokrivanje svakog elementa barem k puta.

Pohlepni algoritam

Pohlepni algoritam (eng. greedy algorithm) je predstavlja način rješavanja problema koja se bazira na donošenju optimalnih odluka na lokalnom nivou u svakom koraku, s nadom da će ukupno rješenje biti zadovoljavajuće, ako ne i optimalno.

Za svaki korak, u rješenje se uključuje ona komponenta koja donosi najveću korist u odnosu na ostale komponente. [1]

Postupak:

- **Inicijalizacija** – inicijalizujemo skupove $universe$ i $sets$ koji čuvaju vrijednost skupova univerzuma i podskupova univerzuma, respektivno. Čitamo iz tekstualnih datoteka unos inicijalnih podataka pomoću $get\_input\_from\_file()$ metode, a ispisujemo u tekstualnu datoteku rezultat rješavanja problema pomoću $print\_to\_file()$ metode.

- **Petlja za odabir podskupova** – petlja se izvršava sve dok postoji element u $universe$ koji nije dovoljno puta pokriven (manje od k puta).

- **Odabir podskupa** – metoda $get\_best\_set()$ se koristi za odabir podskupa, radi tako što se za svaki podskup računa broj elemenata koji se još uvijek trebaju pokriti, ako je pokrivenost različita od nula, u listu $cost\_per\_cover$ se dodaje trošak po pokrivanju kao vrijednost.

  $cost\_per\_cover$ za trošak uzima $\frac{cost[i]}{coverage}$. $cost[i]$ predstavlja unaprijed zadatu cijenu nekog podskupa i, a $coverage$ predstavlja ukupan broj elemenata u skupu koji još nisu pokriveni k puta.

  Metoda vraća indeks skupa sa najmanjim troškom, u slučaju da 2 skupa imaju isti trošak, vraća onaj sa većim brojem pokrivenih elemenata.

- **Povratak metode** – vraća listu odabranih podskupova $selected\_sets$, ukupni trošak odabranih podskupova $selected\_cost$ i listu indeksa odabranih podskupova $selected\_ids$.

Metoda promjenljivih okolina (VNS)

VNS, kojeg su uveli Nenad Mladenović i Pierre Hansen [2], je heuristički algoritam koji se koristi za rješavanje problema optimizacije gdje je ideja da se za neko rješenje gleda okolina rješenja i prelazi se na novo samo u slučaju poboljšanja.

Implementacija VNS algoritma za SkCP radi tako da za neko rješenje dobijeno iz pohlepnog algoritma, pomoću shake algoritma razmrdavamo i vršimo lokalnu pretragu nad dobijenim rezultatom. Ako se dobije bolje rješenje, mijenjamo početno za novo i ponovo pokrećemo rad VNS algoritma, u suprotnom slučaju, povećavamo okolinu pretrage VNS algoritma za 1.

Postupak:

- **Inicijalizacija** – postavljanje parametara algoritma kao što su minimalna i maksimalna veličina okoline, vremensko ograničenje, broj iteracija bez poboljšanja, maksimalni broj iteracija, početnih vrijednosti SkCP itd.

- **Klase** $\mathbf{Individual}$:

    - Koristi pohlepni algoritam iz prethodne stavke za generisanje jedinke.

    - Sadrži metodu $fitness()$ - prolazi kroz gene jedinke, $self.genes$, i za svaki gen koji je odabran (vrijednost mu je 1), dodaje odgovarajući trošak, $cost[k]$ u ukupnu vrijednost. Konačni rezultat predstavlja prilagođenost (fitness) jedinke, tj. ukupni trošak odabranih podskupova.

    - Sadrži metodu $validate()$ za provjeravanje validnosti elemenata univerzuma, tj. da budu pokriveni od strane odabranih podskupova barem k puta.

    - Kao rezultat vraća binarni niz čiji element ima vrijednost 1 ako je uključen, 0 inače.

- **Klasa VNS**:
    - $\boldsymbol{local\_search\_FirstImprovementStrategy}()$ – prolazi kroz skup $sets$, koji predstavlja skup podskupova koji pokrivaju elemente iz unverzuma, provjerava se da li se poboljšava fitness (pomoću metode $validate()$, obj predstavlja fitness vrijednost neke instance), i ako se pronađe poboljšanje, trenutna jedinka se ažurira s tim rješenjem i pretraga se zaustavlja. Pseudokod za lokalnu pretragu:

```
def local_search_FirstImprovementStrategy(self, x):
    for i in range(len(sets)):
        x_new = x_old = x
        if x_new[j] == 0:
            x_new[j] = 1
        if x_new[j] == 1:
            x_new[j] = 0

        if x_new.validate():
            if x_old.obj < x_new.obj:
                x_old = x_new
            break
```

    - $\boldsymbol{local\_search\_BestImprovementStrategy}()$ – prolazi kroz skup $sets$, provjerava da li se poboljšava fitness, i ako se pronađe poboljšanje, trenutna jedinka se ažurira s tim rješenjem i postupak se ponavlja sve dok se ne prođe kroz sve podskupove, omogućavajući traženje najboljeg mogućeg poboljšanja u rješenju. Pseudokod za lokalnu pretragu:

```
def local_search_BestImprovementStrategy(self, x):
    has_better = True

    while has_better:
        has_better = False

        for i in range(len(sets)):
            x_new = x_old = x_best = x

            if x_new[j] == 0:
                x_new[j] = 1
            if x_new[j] == 1:
                x_new[j] = 0

            if x_new.validate():
                if x_old.obj < x_new.obj:
                    x_best = x_new
                    has_better = True

            if has_better == True:
                x_old = x_best
```

- **Procedura razmrdavanja (eng. shaking)** – $shake()$ metod simulira "razmrdavanje" u VNS algoritmu. Prima trenutnu jedinku x i parametar potresa vns_shake, koji određuje koliko gena će se razmrdati. Nasumično bira vns_shake elemenata u kopiji i mijenja njihove vrijednosti inverzijom bitova. Vraća novu konfiguraciju gena koja je rezultat potresa.

- **Pokretanje** – u $run()$ metodi izvršavamo pokretanje algoritma. Inicijalizujemo osnovne parametara kao što su vremenska ograničenja, jedinku x pomoću pohlepnog algoritma, broj iteracija i drugo, nakon toga iterativno primjenjujemo VNS algoritam tako što prolazimo kroz jedinke pomoću razmrdavanja i primjenjujemo lokalnu pretragu na njoj. Ako nađemo poboljšano rješenje, ažuriramo ga. Postupak ponavljamo sve dok ne zadovoljimo uvjete (vremensko ograničenje, broj iteracija).

- **Kraj rada** – na kraju ispisujemo da li smo našli validno rješenje, i ako jesmo ispisujemo ga u izlaznu tekstualnu datoteku.

Cjelobrojno linearno programiranje

Cjelobrojno linearno programiranje (eng. Integer Linear Programming, ILP) se bavi rješavanjem optimizacionih problema gdje su varijable odluke cijeli brojevi. Korišćen je PuLP modul Python programskog jezika za implementaciju rješenja.

Opis ILP modela za SkCP [3]:

$$min \sum_{j \in J} c_j x_j$$

Ovdje je predstavljena minimizacija ciljne funkcije koja predstavlja ukupan trošak izabranih kolona.

c_j predstavlja trošak odabira kolone j kako bi pokrio red, a x_j varijabla odluke koja iznosi 1 ako je kolona j odabrana da pokrije red, 0 inače, za svaki $j \in J = \{1, \ldots, n\}$

$$\sum_{j \in J} a_{ij} x_j \geq k, i \in I, k \in Z^+$$

Ograničenja promjenjivih. Osiguravaju dobijanje približnog rješenja, tj. svaki red je pokriven sa barem k kolona, gdje $k \in Z^+$, pri čemu je a_{ij} parametar koji uzima vrijednost 1 ako kolona j pokriva red i, 0 inače.

$$x_j \in \{0, 1\}, j \in J$$

x_j predstavlja varijablu odluke.

Eksperimentalni rezultati

Za testiranje navedenih algoritama su korištene već datih 44 instanci. [3]

Testiranja su na sljedećim specifikacijama:

- Windows 10 operativni sistem
- AMD Ryzen™ 5 3550H, 2.1 GHz radni takt
- 8 GB DDR4-2400 SDRAM (2 x 4 GB)

instance			greedy		VNS			PuLP	
	sets	universe	g_c	g_t	vns_best	vns_avg	vns_t	pulp_c	pulp_t
scp01	50	500	*10*	*0.01*	*10*	*10*	*600.0839*	***9***	*9.97225*
scp02	50	500	*9*	*0.007*	*9*	*9*	*600.0886*	***8***	*0.35329*
scp03	50	500	***8***	*0.012*	***8***	***8***	*600.1284*	***8***	*0.62309*
scp04	50	500	*10*	*0.014*	*10*	*10*	*600.2094*	***8***	*0.2914*
scp05	50	500	*9*	*0.008*	*9*	*9*	*600.1775*	***8***	*38.93199*
scp10	200	1000	*1523*	*0.24902*	*1523*	*1523*	*600.9059*	***1356***	*0.15146*
scp11	200	1000	*1330*	*0.16901*	*1330*	*1330*	*601.5545*	***1148***	*0.29434*
scp12	200	1000	*1395*	*0.18201*	*1395*	*1395*	*601.1680*	***1205***	*0.07769*
scp13	200	1000	*1366*	*0.16801*	*1366*	*1366*	*600.8196*	***1213***	*0.32468*
scp14	200	1000	*1348*	*0.16301*	*1348*	*1348*	*600.7815*	***1185***	*0.13626*
scp15	200	1000	*1412*	*0.18001*	*1412*	*1412*	*601.1191*	***1266***	*0.37005*
scp16	200	1000	*1500*	*0.16801*	*1500*	*1500*	*600.7298*	***1349***	*0.26982*
scp17	200	1000	*1311*	*0.15101*	*1311*	*1311*	*600.9141*	***1115***	*0.09974*
scp18	200	1000	*1341*	*0.16701*	*1341*	*1341*	*600.6910*	***1225***	*0.45358*
scp19	200	1000	*1639*	*0.16701*	*1639*	*1639*	*601.0222*	***1485***	*0.1192*
scp20	200	2000	*649*	*0.32102*	*649*	*649*	*603.3821*	***579***	*0.2299*
scp21	200	2000	*790*	*0.38198*	*790*	*790*	*604.4103*	***677***	*1.21664*
scp22	200	2000	*632*	*0.35503*	*632*	*632*	*604.5969*	***574***	*0.52266*
scp23	200	2000	*636*	*0.34103*	*636*	*636*	*602.9793*	***582***	*0.88718*
scp24	200	2000	*606*	*0.32102*	*606*	*606*	*603.2238*	***550***	*0.19671*
scp25	200	2000	*622*	*0.34502*	*622*	*622*	*604.6392*	***560***	*0.394*
scp26	200	2000	*778*	*0.33603*	*778*	*778*	*605.3139*	***695***	*0.37733*
scp27	200	2000	*731*	*0.30602*	*731*	*731*	*604.8157*	***662***	*0.41444*
scp28	200	2000	*746*	*0.33203*	*746*	*746*	*603.0151*	***687***	*0.96452*
scp30	400	4000	*131*	*0.89207*	*131*	*131*	*618.6460*	***122***	*21.13055*
scp31	400	4000	*144*	*1.03508*	*144*	*144*	*618.9140*	***127***	*9.44496*
scp32	400	4000	*162*	*0.95228*	*162*	*162*	*615.0102*	***138***	*9.7282*
scp33	400	4000	*137*	*0.93312*	*137*	*137*	*608.3269*	***122***	*16.45542*
scp34	400	4000	*151*	*0.97807*	*151*	*151*	*622.3339*	***130***	*19.44338*
scp40	500	5000	*54*	*1.27115*	*54*	*54*	*610.5772*	***49***	*604.65746*
scp41	500	5000	*57*	*1.30609*	*57*	*57*	*607.1313*	***51***	*1001.66968*
scp42	500	5000	*53*	*1.3031*	*53*	*53*	*615.5497*	***47***	*625.58473*
scp43	500	5000	*55*	*1.3181*	*55*	*55*	*613.8053*	***49***	*1001.60277*
scp44	500	5000	*53*	*1.26008*	*53*	*53*	*608.9332*	***49***	*361.4226685*

Tabela 1: rezultati za instance, k_min

instance			greedy		VNS			*PuLP*	
	sets	universe	g_c	g_t	vns_best	vns_avg	vns_t	*pulp_c*	*pulp_t*
scp01	50	500	*171*	*0.15701*	*171*	*171*	*600.3913*	***158***	*903.05776*
scp02	50	500	*173*	*0.17001*	*172*	*172.4*	*600.2730*	***158***	*1000.48961*
scp03	50	500	*176*	*0.15901*	*176*	*176*	*600.1429*	***166***	*173.19766*
scp04	50	500	*185*	*0.16801*	*184*	*184.8*	*600.4076*	***173***	*1000.34885*
scp05	50	500	*178*	*0.16601*	*177*	*177.8*	*600.2389*	***168***	*1000.65791*
scp10	200	1000	*6103*	*0.38803*	*6103*	*6103*	*601.3878*	***5355***	*1.17375*
scp11	200	1000	*7074*	*0.42403*	*7074*	*7074*	*601.1622*	***6404***	*4.65981*
scp12	200	1000	*5099*	*0.35603*	*5099*	*5099*	*600.6996*	***4586***	*1.85178*
scp13	200	1000	*5215*	*0.39603*	*5215*	*5215*	*600.8411*	***4676***	*1.31063*
scp14	200	1000	*5295*	*0.35502*	*5295*	*5295*	*601.1574*	***4670***	*3.47199*
scp15	200	1000	*7040*	*0.44003*	*7040*	*7040*	*601.0841*	***6474***	*4.83931*
scp16	200	1000	*7149*	*0.42203*	*7149*	*7149*	*600.7611*	***6416***	*8.50217*
scp17	200	1000	*7127*	*0.46503*	*7127*	*7127*	*601.8848*	***6281***	*0.79216*
scp18	200	1000	*7145*	*0.41703*	*7145*	*7145*	*601.4677*	***6515***	*6.47427*
scp19	200	1000	*7859*	*0.46803*	*7859*	*7859*	*600.9503*	***7101***	*5.64121*
scp20	200	2000	*12142*	*1.74827*	*12142*	*12142*	*601.7526*	***11205***	*301.54122*
scp21	200	2000	*16006*	*1.80014*	*16006*	*16006*	*603.8452*	***14418***	*211.30607*
scp22	200	2000	*12315*	*1.68012*	*12315*	*12315*	*603.3608*	***11476***	*132.91663*
scp23	200	2000	*9360*	*1.44011*	*9360*	*9360*	*602.3700*	***8514***	*49.9698*
scp24	200	2000	*11872*	*1.53111*	*11872*	*11872*	*602.9201*	***10880***	*58.42347*
scp25	200	2000	*9938*	*1.51111*	*9938*	*9938*	*602.4763*	***9093***	*1000.27371*
scp26	200	2000	*14512*	*1.69614*	*14512*	*14512*	*602.7310*	***13143***	*448.59135*
scp27	200	2000	*11822*	*1.54322*	*11809*	*11819.4*	*602.9502*	***10622***	*984.34632*
scp28	200	2000	*12123*	*1.57112*	*12123*	*12123*	*603.4011*	***11042***	*134.23641*
scp30	400	4000	*41787*	*22.45531*	*41787*	*41787*	*631.9931*	***39124***	*1000.48878*
scp31	400	4000	*40863*	*23.45163*	*40863*	*40863*	*630.6897*	***38309***	*1000.41601*
scp32	400	4000	*40927*	*21.67045*	*40927*	*40927*	*630.1570*	***38615***	*1000.40847*
scp33	400	4000	*41390*	*24.94168*	*41390*	*41390*	*643.1512*	***39113***	*1000.38867*
scp34	400	4000	*42240*	*24.75018*	*42240*	*42240*	*638.2672*	***39587***	*1000.51329*
scp40	500	5000	*61717*	*87.49594*	*61717*	*61717*	*604.1281*	***59206***	*1001.48114*
scp41	500	5000	*60596*	*87.85724*	*60596*	*60596*	*603.7560*	***57466***	*1001.1107*
scp42	500	5000	*59639*	*99.84608*	*59639*	*59639*	*656.6659*	***56847***	*1001.43731*
scp43	500	5000	*58864*	*88.77179*	*58864*	*58864*	*624.9167*	***56163***	*1000.98884*
scp44	500	5000	*58742*	*88.06986*	*58742*	*58742*	*607.4108*	***55770***	*1001.035177*

Tabela 2: rezultati za instance, k_med

instance			greedy		VNS			PuLP	
	sets	universe	g_c	g_t	vns_best	vns_avg	vns0_t	pulp_c	pulp_t
scp01	50	500	*388*	*0.32702*	*386*	*387.6*	*600.2331*	***366***	*0.29503*
scp02	50	500	*386*	*0.33902*	*384*	*385.4*	*600.3401*	***364***	*4.33141*
scp03	50	500	*414*	*0.36801*	*410*	*413*	*600.2512*	***393***	*0.26568*
scp04	50	500	*438*	*0.38503*	*438*	*438*	*600.3773*	***421***	*0.27732*
scp05	50	500	*416*	*0.36703*	*415*	*415.6*	*600.2948*	***398***	*0.25584*
scp10	200	1000	*13098*	*0.58004*	*13098*	*13098*	*600.9399*	***11607***	*1.95962*
scp11	200	1000	*20114*	*0.75406*	*20114*	*20114*	*601.1000*	***18265***	*0.75609*
scp12	200	1000	*13679*	*0.59304*	*13679*	*13679*	*600.9977*	***12360***	*18.75408*
scp13	200	1000	*11459*	*0.54304*	*11459*	*11459*	*600.8842*	***10396***	*0.78192*
scp14	200	1000	*11472*	*0.55304*	*11472*	*11472*	*601.2458*	***10393***	*37.73804*
scp15	200	1000	*20617*	*0.73905*	*20617*	*20617*	*601.1137*	***18856***	*0.54863*
scp16	200	1000	*17053*	*0.66205*	*17053*	*17053*	*601.4813*	***15394***	*7.75459*
scp17	200	1000	*17060*	*0.68435*	*17060*	*17060*	*600.8891*	***15233***	*7.09537*
scp18	200	1000	*20429*	*0.72305*	*20429*	*20429*	*601.2331*	***18602***	*5.38236*
scp19	200	1000	*18058*	*0.66305*	*18058*	*18058*	*600.6789*	***16558***	*4.10447*
scp20	200	2000	*38249*	*3.06632*	*38249*	*38249*	*603.1114*	***35670***	*1000.27778*
scp21	200	2000	*49207*	*3.3662*	*49207*	*49207*	*603.7268*	***45396***	*32.66176*
scp22	200	2000	*39452*	*3.21124*	*39452*	*39452*	*602.7421*	***36330***	*1000.37034*
scp23	200	2000	*30766*	*2.7202*	*30765*	*30765.8*	*603.8029*	***28017***	*216.43997*
scp24	200	2000	*35865*	*2.9355*	*35865*	*35865*	*604.6102*	***32779***	*462.45328*
scp25	200	2000	*32476*	*2.69547*	*32476*	*32476*	*604.1757*	***29618***	*1000.23114*
scp26	200	2000	*45601*	*3.21548*	*45601*	*45601*	*603.6273*	***41930***	*372.04536*
scp27	200	2000	*34718*	*2.8162*	*34641*	*34702.6*	*603.0098*	***32320***	*345.38655*
scp28	200	2000	*36716*	*2.90248*	*36716*	*36716*	*603.8242*	***33584***	*482.3513*
scp30	400	4000	*154203*	*48.24955*	*153844*	*154124.8*	*646.1887*	***145078***	*1000.48902*
scp31	400	4000	*153871*	*48.75991*	*153871*	*153871*	*600.7345*	***144218***	*1000.46725*
scp32	400	4000	*149826*	*50.29554*	*149628*	*149786.4*	*646.2058*	***140702***	*1000.47941*
scp33	400	4000	*153110*	*51.92879*	*153033*	*153088.4*	*641.3666*	***143598***	*1000.54286*
scp34	400	4000	*156046*	*50.73019*	*155988*	*156034.4*	*604.7194*	***146374***	*1000.61193*
scp40	500	5000	*234848*	*175.14983*	*234606*	*234780.4*	*634.8468*	***224053***	*1001.70098*
scp41	500	5000	*236193*	*186.31337*	*236103*	*236175*	*645.0258*	***225011***	*1001.21007*
scp42	500	5000	*231265*	*185.33963*	*231090*	*231204.6*	*640.6056*	***220344***	*1001.15861*
scp43	500	5000	*228762*	*182.84078*	*228691*	*228747.8*	*637.5328*	***218279***	*1001.34611*
scp44	500	5000	*228118*	*176.90567*	*228118*	*228118*	*640.5333*	***218422***	*1000.775216*

Tabela 3: rezultati za instance, k_max

Vrijeme se mjeri u sekundama. k_min iznosi 2, k_max je maksimalni broj pokrivanja svih elemenata i k_med je $\frac{k\_min+k\_max}{2}$.

Opis rezultata:

Za bilo koje od navedenih k, PuLP se daje najbolje rezultate.

k_min:

- VNS i pohlepni algoritmi daju ista rješenja
- razlika je ~10-15% između heuristike optimalnog.

k_med:

- Za 11.76% instancih je bolji VNS, za ostale ista rješenja kao i za pohlepni algoritam.
- razlika je ~5-10% između heuristike optimalnog.

k_max:

- Za 41.18% instancih je bolji VNS, za ostale ista rješenja kao i za pohlepni algoritam.
- razlika je ~5-10% između heuristike optimalnog.

Zaključak

U ovom radu smo opisali SkCP, kao i pohlepni pristup, metodu promjenljivih okolina i metodu cjelobrojnog linearnog programiranja.

Odluka izbora nekog od navedena 3 pristupa zavisi od veličina instancih, kao i potrebe (da li nam treba optimalno rješenje, ali u zamjenu za potrošeno vrijeme dolaska do tog rješenja, ili približno dobro rješenje, ali značajno brže).

Kako je bilo i za pretpostaviti, pohlepni pristup daje rezultate sa najboljim vremenom, ali sa neoptimalnim troškom. Za male, kao i srednje i velike instance PuLP daje najbolje rješenje dok pohlepni i VNS algoritmi daju u većini slučajeva iste rezultate. VNS poboljša rješenje u malom broju slučajeva u odnosu na pohlepni pristup, ali za znatno veće iskorišteno vrijeme.

Literatura

[1] M. Đukanović and D. Matić, Uvod u operaciona istraživanja, Banja Luka, 2022.

[2] N. Mladenović and P. Hansen, Variable neighborhood search, Computers and Operations Research, 1997.

[3] A. Salehipour, "A Heuristic Algorithm for the Set k-Cover Problem," in *Optimization and Learning* , 2020, pp. 98-112.