Универзитет у Бањој Луци

Природно-математички факултет

СЕМИНАРСКИ РАД

Предмет: Операциона истраживања

Тема: Проблем к-доминације

Ментор:

доц. др Марко Ђукановић

Студенти:

Јована Ђукић

Даница Бабић

Садржај

Увод

У овом семинарском раду разматрамо проблем к-доминације (engl. K-domination problem) који ћемо да решавамо помоћу похлепног алгоритма (engl. Greedy algorithm), метахеуристичке методе генетског алгоритма (engl. Genetic algorithm) као и помоћу алата PuLP-а који је помоћна библиотека програмског језика Python.

Прво ћемо детаљно да опишемо проблем којим се бавимо и алгоритме и алате које смо користили за решавање овог проблема. После тога ћемо да применимо алгоритме на улазни скуп инстанци и да упоређујемо добијене резултате.

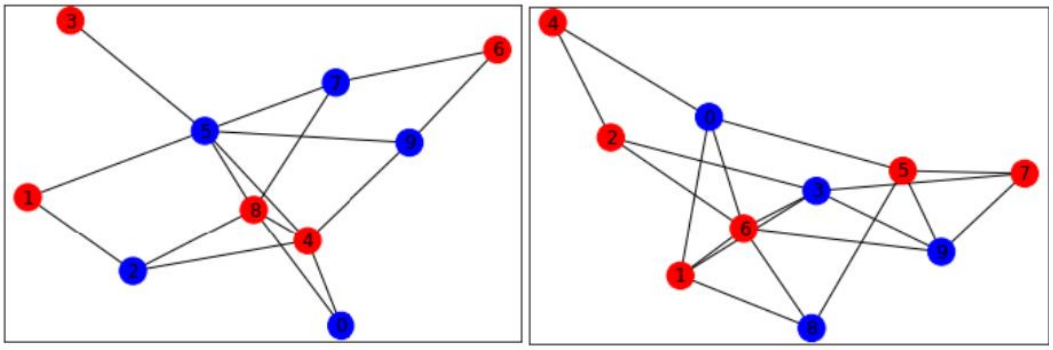
Циљ овог семинарског рада је да видимо који од алгоритама даје најбоља решења за дати проблем те да се упознамо са оптимизацијом проблема и алгоритама.

Што се тиче програмског језика, одлучили смо да користимо Python због великог броја помоћних библиотека и због једноставности синтаксе.

Опис проблема

Верује се да доминирајући скупови потичу из игре шаха, где је циљ био да се одређеним шаховским фигурама покрију или „доминирају“ различита поља шаховске табле. Проблем одређивања доминирајућег броја графа први пут се појавио 1862. године у раду de aenisch. Циљ је био да се пронађе минимални број дама на шаховској табли, тако да свако поље или заузима дама или да до њега може да дође дама у једном потезу. Доминацију као теоријску област у теорији графова су формализовали Berge 1958. и Ore 1962. године.

Формулација проблема: Дат је једноставан граф (граф који се састоји од скупа објеката који се називају врхови или темена и скупа парних односа између објеката који се називају ивице) и позитиван цео број к; проблем минималног к-доминирајућег скупа састоји се у проналажењу најмањег могућег (по кардиналности) подскупа врхова графа тако да је свако теме датог графа или елемент датог подскупа или је суседан са најмање к елемената овог скупа, што можемо и да видимо на примеру приказаном на слици 1.



Слика 1. За к = 2 (лево) и за к = 3 (десно) где црвени чворови припадају доминирајућем скупу

Доказано је да је проблем к-доминације НП-тежак, на пример у неповезаним графовима. Због тога не постоји тачан метод за израчунавање оптималног решења у разумном времену већ се користе хеуристичке методе које дају решење у разумном времену али то решење може бити субоптимално.

Математичка формулација проблема к-доминације

Као што смо већ поменули проблем доминације је доказано НП-тежак, тако да је и проблем к-доминације исто НП-тежак, јер се он може свести на проблем доминације, кад узмемо да је к=1. Проблем доминације се може моделовати као следећи модел целобројног линеарног програмирања (ILP).

Минимизирај: $\sum_{v\in V} y_v$,

Тако да: $y_v + \sum_{uv\in E} y_u \geq 1, \forall v \in V$,

$y_v \in \{0,1\}, \forall v \in V$.

Бинарну променљиву y_v уводимо тако да за сваки чвор v из V узима вредност 1 ако је чвор v изабран у доминирајући скуп, а вредност 0 ако није. Пошто желимо минималан број чворова у доминирајућем скупу, функција циља изгледа овако: min $\sum_{v\in V} y_v$. Како за сваки чвор мора важити да је или он у доминирајућем скупу, или да има бар једног суседа у доминирајућем скупу, ограничење за чвор v гласи: $y_v + \sum_{uv\in E} y_u \geq 1, \forall v \in V$. Ако је чвор v у доминирајућем скупу, променљива y_v ће имати вредност 1, па нам није важна вредност десног сабирка, јер је неједнакост свакако испуњена. Кад је y_v= 0, тада десни сабирак мора имати вредност већу или једнаку 1, да би неједнакост била задовољена, а то је баш еквивалентно са условом да чвор v има бар једног суседа u у доминирајућем скупу.

Пошто нама треба математичка дефиниција проблема к-доминирајућег скупа, мало смо изменили горе наведену дефиницију тако да одговара нашем проблему, те смо добили следеће. Треба нам y_v = 1 или $\sum_{uv\in E} y_u \geq k$.

Дакле, сада дефиниција за наш проблем изгледа овако:

Минимизирај: $\sum_{v\in V} y_v$,

Тако да: $k * y_v + \sum_{uv\in E} y_u \geq k$

$y_v \in \{0,1\}, \forall v \in V$.

Како се услов проблема за десни сабирак у услову допустивости променио са 1 или више на k или више (то јесте кад је y_v= 0 мора бар k суседа чвора v бити у доминирајућем скупу), то је десна страна неједнакости повећана на k. Сада за y_v= 1, то јесте када је v у доминирајућем скупу, може бити да v нема ниједног суседа у доминирајућем скупу, па је цела сума у десном сабирку једнака 0, па због тога морамо

y_v помножити са k, ако желимо да неједнакост важи и у том случају. А све остало остаје као и за проблем доминације.

PuLP

PuLP је Python библиотека која се може користити за решавање проблема линеарног програмирања. Линеарно програмирање се користи за решавање проблема оптимизације и користи се у различитим индустријама као што су производња, транспорт, исхрана итд.

Линеарно програмирање (LP), такође познато као линеарна оптимизација, је техника математичког програмирања за постизање најбољег резултата или исхода, као што је максимални профит или најмањи трошак, у математичком моделу чији су захтеви представљени линеарним односима. Линеарно програмирање је посебан случај математичког програмирања, познатог и као математичка оптимизација.

Генерално, организација или компанија има углавном два циља, први је минимизација, а други максимизација. Минимизација значи минимизирање укупних трошкова производње, док максимизација значи максимизирање њиховог профита. Дакле, уз помоћ графичке методе линеарног програмирања можемо пронаћи оптимално решење.

Основни кораци при решавању проблема коришћењем PuLP-а су:

1. Декларација варијабли
2. Дефинисање функције циља
3. Дефинисање ограничења модела
4. Методе за извоз решења проблема из модела

Инсталација ове Python библиотеке је веома једноставна. Све што нам је потребно јесте да имамо верзију Python >= 2,7 или Python>= 3,4. Пошто нам је тако било најлакше одлучили смо се за инсталацију пакета путем pip-а. користећи следећу pip наредбу у командној линији:

```
python -m pip install pulp
```

Решење проблема к-доминације помоћу PuLP-а

За почетак смо морали да уведемо PuLP библиотеку:

```
from pulp import *
```

Даље следи иницијализација проблема:

```
model = LpProblem("k_dominacija", LpMinimize)
```

где је к-доминација име проблема а LpMinimize служи за одређивање да ли се тражи максимум или минимум, у нашем случају тражимо минимално решење.

Правимо бинарне варијабле одлучивања:

```
y = LpVariable.dicts("y", cvorovi, 0, 1, LpBinary)
```

Функција циља:

```
model += ( lpSum([y[i] for i in cvorovi]))
```

Ограничења проблема:

```
for i in cvorovi:
    model += k*y[i] + lpSum( y[j] for j in al[i]) >= k
```

Позив функције оптимизације:

```
model.solve(PULP_CBC_CMD(maxSeconds=1000, msg=1, fracGap=0))
#status:
print("Status solve: ", LpStatus[ model.status ])
print("Vars: \n")
for v in model.variables():
    print(v.name, "===> ", v.varValue)
print("Obj value: ", value(model.objective))
```

Похлепни алгоритам

Похлепни алгоритам се дефинише као метод за решавање проблема оптимизације доношењем одлука које резултирају најочитијом и тренутном користи, без обзира на крајњи исход. Похлепни алгоритам је метода која се користи у проблемима оптимизације где је циљ да се направи локално оптималан избор у свакој фази са надом да ће се пронаћи глобални оптимум. Назива се „похлепним“ јер покушава да пронађе најбоље решење правећи најбољи избор на сваком кораку, не узимајући у обзир будуће кораке или последице тренутне одлуке.

Решење проблема к-доминације похлепним алгоритмом

Постоје различити начини одабира који чворови се разматрају. У овом раду ћемо представити два могућа начина за прављење похлепног алгоритма за проблем к-доминације.

Први начин имплементације похлепног алгоритма

У првом начину алгоритам се заснива на тактици да у решење (скуп чворова) додајемо чворове са највише суседа. Затим за сваки наредни чвор са највише суседа проверава да ли тај чвор има к суседа унутар тренутног решења. Алгоритам смо имплементирали на основу идеје из рада "Computers and Operations Research-Padraig Corcoran,Andrei Gagarin".

Други начин имплементације похлепног алгоритма

Други начин се заснива на псеудо-коду, приказаном на слици 2, који је дат у раду[1].

[1] https://epubs.siam.org/doi/pdf/10.1137/1.9781611973037.4

ALGORITHM 1. ***Greedy approximation for a k-dominating set***
Input: *$G = (V, E)$ and $k \in \mathbb{N}$.*
Output: *k-dominating Set D_k.*

1: *Initialize $D_k^0 = \emptyset$ and $i = 0$.*
2: *Give all nodes their current degree of k-domination: $d_k(v) = 0$ for all $v \in V$.*
3: ***while*** *there are nodes $v \in V$ with $d_k(v) < k$* ***do***
4: *Choose a node $g^i \in V$, $g^i \notin D_k^i$ that reduces the remaining cost $a(G, D_k^i \bigcup \{g^i\})$ in comparison to $a(G, D_k^i)$ the most.*
5: *$d_k(g^i) = k$ and $D_k^{i+1} = D_k^i \bigcup \{g^i\}$.*
6: *Raise for all $v' \in N(g^i)$ with $d_k(v') < k$ the degree of k-domination $d_k(v')$ by one, i.e., $d_k(v') = d_k(v') + 1$.*
7: *$i = i + 1$.*
8: ***end while***
9: *$D_k = D_k^i$.*
10: ***return*** *D_k*

Слика 2. Псеудо-код похлепног алгоритма за решење к-доминацијског проблема

Кораци овог алгоритма су следећи. Као улазни скуп података добијемо граф G = (V, E), где је V скуп чворова а E скуп грана датог графа, а као излазне податаке добијемо скуп D, који називамо доминирајући скуп. На почетку иницијализујемо скуп D као листу чија је дужина једнака броју чворова датог графа, чији су сви елементи једнаки -1. Затим иницијализујемо листу brojSusedauD, која има исту дужину као и листа D, и чији су првобитни елементи сви једнаки 0. У овој листи чувамо информације о томе колико сваки чвор има суседа у D, а уколико се неки чвор налази у D, онда је његова вредност у листи једнака к.

Затим се улази у while петљу док није задовољен услов да су сви елементи листе brojSusedauD једнаки к. Ако се уђе у петљу, пролази се кроз све чворове из V/ D, и за сваки од њих се рачуна колики би био преостали трошак уколико се тај чвор дода у D. Када за сваки чвор то урадимо у D додамо онај чвор који ће да врати најмањи преостали трошак. Преостали трошак се рачуна по формули приказаној на слици 3. Од производа датог броја к и броја чворова у почетном графу одузима се сума бројева из листе brojSusedauDpomocna. Листа brojSusedauDpomocna је листа brojSusedauD којој се додају вредности за тренутни чвор, вредност тог чвора се постави на к а свим његовим суседима се вредности повећавају за 1.

DEFINITION 5.2. *Let $G = (V, E)$ be a graph, $D \subseteq V$ be a set of nodes and $k \in \mathbb{N}$. The remaining cost $a(G, D)$ for k-domination is defined as:*

$$a(G, D) = nk - \sum_{v \in V} d_k(v, D) \quad . \tag{5.2}$$

Слика 3. представља начин рачунања преосталог трошка

Сада ажурирамо податке, у D додамо дати чвор, исти чвор избацимо из листе V/ D, листу brojSusedauD ажурирамо тако што изабраном чвору поставимо вредност на к а сваком суседу тог чвора повећамо вредност за 1, осим ако је та вредност већ једнака к.

И на крају кад су сви елементи листе brojSusedauD једнаки к, као резултат вратимо листу D.

Генетски алгоритам

У рачунарским наукама и оперативним истраживањима, генетски алгоритам (ГА) је метахеуристика инспирисана процесом природне селекције која припада већој класи еволуционих алгоритама (ЕА). Генетски алгоритми се обично користе за генерисање висококвалитетних решења за проблеме оптимизације и претраге ослањајући се на биолошки инспирисане операторе као што су мутација, укрштање и селекција.

У генетском алгоритму, популација кандидата решења (назива се појединци, створења, организми или фенотипови) за проблем оптимизације се развија ка бољим решењима. Сваки кандидатски раствор има скуп својстава (његове хромозоме или генотип) који могу бити мутирани и измењени; традиционално, решења су представљена у бинарном облику као низови од 0s и 1s, али су могућа и друга кодирања.

Слика 4. Приказ структуре популације

Fitness функција (прилагођеност) претсавља бројну вредност која оцењује квалитет јединке. Циљ генетског алгоритма јесте да се кроз итерације проналазе јединке са бољом fitness функцијом које ће се даље користити у процесу селекције,укрштања, мутације како би се у наредним генерацијама добијале што квалитетније јединке.

Еволуција обично почиње од популације насумично генерисаних појединаца и представља итеративни процес, при чему се популација у свакој итерацији назива генерацијом. Нова генерација решења кандидата се затим користи у следећој итерацији алгоритма.У зависности од проблема који се решава потребно је размотрити начин како што боље одабрати чланове популације као и њену величину. Најчешћи критеријуми

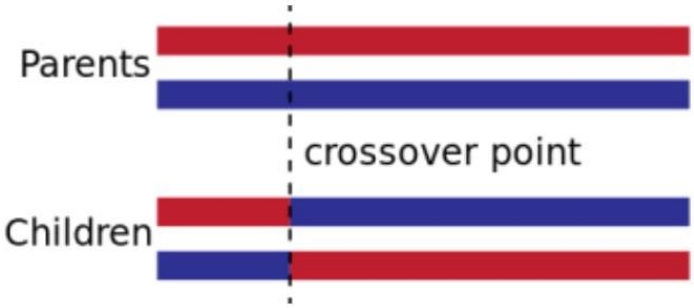
заустављања генетског алгоритма јесу достигнут максималан број итерација и временско ограничење.

Типичан генетски алгоритам захтева:

- генетски приказ домена решења
- функцију фитнеса за процену домена решења

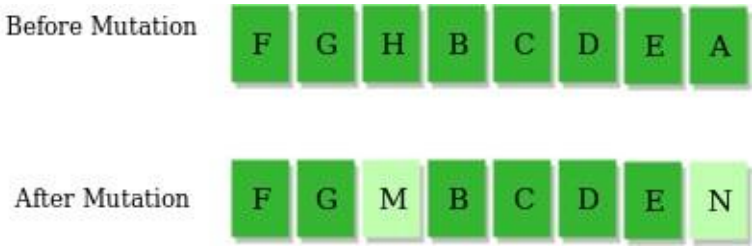
Селекција - оператор селекције бира јединке над којима ће се извршити оператор укрштања и мутације.Јединке са бољом fitness функцијом имају већу шансу да буду изабране. Постоји више врста селекција од којих су најпознатије рулет селекција, турнирска селекција, елиминацијска селекција и елитизам. Код рулет селекције вјероватноћа да ће јединка бити изабрана је пропорционална fitness функцији. Турнирска селекција итеративно бира скуп јединки са најбољим fitness функцијама. Елиминацијска селекција избацује јединке са лошим fitness функцијама. Елитизам бира најбље јединке које ће ући у наредну генерацију.

Укрштање - оператор укрштања мења гене једике тако да оне постају потенцијално боље јединке са бољом fitness функцијом. Најпознатији оператори укрштања су једнопозиционо и двопозиционо укрштање. Код једнопозиционог укрштања одређује се једна позиција гдје се врши размена гена а код двопозиционог укрштања постоје две позиције гдје се врши размена гена.



Слика 5. Једнопозиционо укрштање

Мутација - оператор мутације враћа потенцијално добар ген унутар јединке на произвољној позицији.



Слика 6. Приказ мутације

Стандардна репрезентација сваког решења кандидата је као низ битова (који се такође назива сет битова или низ битова). Низови других типова и структура могу се користити у суштини на исти начин. Главно својство које чини ове генетске репрезентације погодним је да се њихови делови лако поравнавају због њихове фиксне величине, што олакшава једноставне операције укрштања. Могу се користити и репрезентације промењљиве дужине, али је имплементација укрштања сложенија у овом случају.

Репрезентације налик стаблу се истражују у генетском програмирању, а репрезентације у облику графа се истражују у еволуционом програмирању, мешавина линеарних хромозома и стабала истражује се у програмирању генске експресије.

Решење проблема к-доминације генетским алгоритмом

Скуп података које смо обрађивали јесу чворови нумерисани бројевима (1,...,n) који чине граф. Популације су нам чиниле низ јединки фиксиране дужине формиране чворовима из овог скупа. Свака оваква јединка са низом чворова нам је представљала потенцијално решење.

У нашем алгоритму почетно смо одабрали произвољне гене за сваку јединку које смо побољшали са уметнутим генима(чворовима). Уметнуте чворове смо бирали тако да смо из скупа чворова графа одабрали n чворова са највише суседа које смо затим убацили у јединку. Остатак јединке били су произвољни чворови. Величину популације смо бирали тако да заједно са бројем итерација којима ће производити наредне генерације јединки буде задовољено извршавање у разумном времену. Величину популације као и број итерација смо мењали по потреби због побољшања решења и побољшања времена извршавања. Почетно смо популацију ограничили на 200 и пустили алгоритам а затим смо прилагођавали величину.

На основу резултата похлепног алгоритма које смо претходно тестирали и који нам је дао кардиналност скупа решења том метдом,бирали смо број гена који ће се наћи унутар јединке. С обзиром на то да смо имплементирали похлепни алгоритам, који према претходно поменутим научним радовима даје јако добра решења, могли смо проценити гдје нам се налазе потенцијално добра решења. Горња граница је била резултат похлепног алгоритма над графом за којег смо тражили боље решење. Благо смо се кретали према доњој граници све док нисмо више добијали добра решења.

Дакле, резултате похлепног алгоритма смо побољшали и тестирали смо генетски алгоритам над таквом популацијом а након тога уколико нам је fitness функција таквог скупа јединки била једнака нули онда смо радили још додатно сажимање скупа гена и тако све док не нарушимо оптимално решење. Тестирање смо првобитно спровели четири пута за сваку инстанцу, итеративно.

За сваку јединку смо рачунали fitness функцију која нам је процењивала колико нам је добро решење тј. колико је наше тренутно решење одступало од унапред процењеног оптималног рјешења. Fitness функција помоћу матрице суседства рачуна за сваки чвор да ли има к суседа унутар одабране јединке или се чвор налази у јединки. За сваки чвор додаје penal = 1 уколико чвор нема к суседа унутар дате популације односно penal = 0 уколико има. Крајња fitness функција се рачуна по формули *fitness*= $\sum_n penal,$ n=1,…,l , где је n величина популације.

У сваку наредну популацију су улазиле јединке које су преживеле у процесу Турнирске селекције коју смо бирали због ефикасности извршавања. Ова селекција нам је бирала итеративно између двије суседне јединке ону са бољом fitness функцијом. Над таквом популацијом, вршили смо укрштање гена. Користили смо једнопозиционо укрштање тако што смо на произвољној позицији двију популација замјенили гене.

Резултат укрштања су биле двије једнике са новим генима. Недопустива рјешења која смо добијали укрштањем смо аутоматски избацивали. Мутација нам је омогућила да заменимо ген на произвољној позицији унутар јединке. Користили смо два опетатора мутације. Први оператор мутације са фактором 0.4 дефинише мутацију која се дешава када је случајно генерисани број мањи од задате вероватноће 0.4. Други оператор мутације са фактором 0.2 дефинише мутацију која се дешава када је случајно генерисани број мањи од задате вероватноће 0.2. Када ће се користити први а када други оператор смо мењали али

смо најчешће први оператор користили у првој половини извршених корака а други оператор у другој половини извршених корака. Функције мутације смо имплементирали тако да су онемогућена недопустива решења. Генерише се произвољна позиција гена који треба да се замени и генерише се произвољни број који нам даје позицију у низу тј. чвор који ће заменити стари чвор. Уколико је генерисани чвор(ген) већ био унутар јединке генерисао се нови број и тако редом.

Резултати

У овом делу ћемо приказати резултате претходно поменутих алгоритама за наш улазни скуп података. Циљ овог поглавља је да се упореди ефикасност свих алгоритама и извуче закључак који алгоритам даје најбоља решења у датом времену.

Што се тиче величина инстанци, алгоритме смо тестирали на графовима са по 50, 100, 150 чворова. То су инстанце које смо сами направили. А затим смо алгоритме тестирали на великим инстанцама које имају 500 и више чворова, које нам је проследио предметни професор.

Табеле резултата

Табела резултата за мале инстанце. То су инстанце са 50 чворова. Тестирамо за вредности к = 1, к = 2 и к = 3.

Мале инстанце	Број к	Број чворова	Број грана	Похлепни Први начин		Похлепни Другии начин		PuLP		Генетски	
				Резултат	Време	Резултат	Време	Резултат	Време	Резултат	Време
Инстанце 1	1	50	623	5	0.10991	3	0.01894	3	0.61419	3	17.25580
Инстанце 2	1	50	734	3	0.08193	3	0.02344	3	2.84935	3	10.399988
Инстанце 3	1	50	872	3	0.0639	2	0.02194	2	0.4872	2	1.08937
Инстанце 4	1	50	855	3	0.04496	3	0.0299	2	0.98447	2	226.41928
Инстанце 5	1	50	980	2	0.0379	2	0.02493	2	0.4257	2	1.27321
Инстанце 1	2	50	623	6	0.06395	5	0.03989	5	0.6894	**4**	902.3486
Инстанце 2	2	50	734	5	0.04396	5	0.04939	4	0.39157	4	2305.2239
Инстанце 3	2	50	872	4	0.0509	4	0.04587	4	3.15476	**3**	2620.6496
Инстанце 4	2	50	885	5	0.0659	4	0.0497	4	1.9115	**2**	1530.7906
Инстанце 5	2	50	980	4	0.0379	3	0.0389	3	0.22889	**2**	32.2580
Инстанце 1	3	50	623	8	0.05396	7	0.06147	7	5.25398	**6**	7540.501
Инстанце 2	3	50	734	8	0.022004	6	0.05485	6	3.6904	**4**	619.8149
Инстанце 3	3	50	872	6	0.03697	5	0.0592	5	0.54566	**4**	40.44395
Инстанце 4	3	50	885	7	0.05493	6	0.0718	5	0.49132	**4**	39.52450
Инстанце 5	3	50	980	6	0.04694	4	0.04886	4	0.42513	4	46.31879

Табела резултата за средње инстанце. То су инстанце са 100 чворова. Тестирамо за вредности к = 1, к = 2 и к = 3.

Мале инстанце	Број к	Број чворова	Број грана	Похлепни Први начин		Похлепни Другии начин		PuLP		Генетски	
				Резултат	Време	Резултат	Време	Резултат	Време	Резултат	Време
Инстанце 1	1	100	2453	5	0.20687	4	0.25676	4	25.34816	4	902.8437
Инстанце 2	1	100	3021	4	0.30575	3	0.20102	3	6.183884	3	2305.3870
Инстанце 3	1	100	3449	3	0.26580	3	0.223544	3	11.3771	3	2620.6743
Инстанце 4	1	100	3451	4	0.27480	2	0.175714	2	0.667647	2	1530.844
Инстанце 5	1	100	3931	3	0.24582	2	0.182692	2	0.554012	2	32.2757
Инстанце 1	2	100	2453	7	0.388720	7	0.386752	6	23.52531	**5**	22.62395
Инстанце 2	2	100	3021	7	0.317750	5	0.356417	5	9.811712	**4**	88.32239
Инстанце 3	2	100	3449	5	0.398714	4	0.289814	4	4.933583	4	1.57312
Инстанце 4	2	100	3451	6	0.275803	4	0.314923	4	8.94137	4	103.7906
Инстанце 5	2	100	3931	5	0.328762	4	0.3418216	3	0.6482958	3	28.41625
Инстанце 1	3	100	2453	11	0.404731	8	0.4702954	8	38.0827	8	24.92039
Инстанце 2	3	100	3021	8	0.353746	7	0.454690	6	9.92376	6	85.21117
Инстанце 3	3	100	3449	7	0. 395719	6	0.423874	5	0.973119	5	37.17157
Инстанце 4	3	100	3451	7	0. 312798	6	0.386257	6	22.56396	6	94.79218
Инстанце 5	3	100	3931	6	0. 390719	5	0.376031	5	6.732303	**4**	108.3689

Табела резултата за средње инстанце. То су инстанце са 150 чворова. Тестирамо за вредности к = 1, к = 2 и к = 3.

Мале инстанце	Број к	Број чворова	Број грана	Похлепни Први начин		Похлепни Другии начин		PuLP		Генетски	
				Резултат	Време	Резултат	Време	Резултат	Време	Резултат	Време
Инстанце 1	1	150	5528	7	0.86036	4	0.706220	4	69.53537	4	7540.50
Инстанце 2	1	150	6736	4	0.839395	3	0.669721	3	14.30148	3	38752.28
Инстанце 3	1	150	7795	4	0.816416	3	0.6807055	3	15.31862	3	43646.41
Инстанце 4	1	150	7740	4	0.853386	3	0.865585	3	14.73341	3	698.9294
Инстанце 5	1	150	8871	4	0.759472	2	0.645916	2	1.95640	2	782.05931
Инстанце 1	2	150	5528	9	0.9743204	7	1.205726	6	64.40580	6	889.2862
Инстанце 2	2	150	6736	7	0.945343	5	1.03478	5	23.454149	5	1871.5780
Инстанце 3	2	150	7795	5	0.90934	5	1.252428	4	12.085137	4	10.75005
Инстанце 4	2	150	7740	5	0.88536	5	1.224639	4	20.13502	4	1352.8969
Инстанце 5	2	150	8871	6	0. 771464	4	0.933326	4	18.94331	4	57.077
Инстанце 1	3	150	5528	11	1.126212	10	1.751379	**8**	79.25949	9	1699.670
Инстанце 2	3	150	6736	10	1.219144	8	1.77246	**7**	86.85478	8	18110.968
Инстанце 3	3	150	7795	6	1.161168	6	1.539255	6	49.940716	6	1495.690
Инстанце 4	3	150	7740	8	0. 98627	7	2.120601	6	53.220038	6	751.98622
Инстанце 5	3	150	8871	6	0. 93730	6	1.831238	**5**	16.62131	6	860.6444

Закључак

Као што можемо закључити из приложених резултата најгоре резултате је дао први начин имплементирања похлепног алгоритам без обзира на величину коришћених инстанци.

Оно што нас је изненадило јесте то што је други начин имплементације похлепног алгоритма давао, скоро исто добре резултате за наше инстанце, као и PuLP.

Као орентацију у генетском алгоритму смо користили резултате добијене првим похлепним алгоритмом јер други тада није давао ништа боља решења од првог. За сваки улазни скуп података, генетски алгоритам је пружио боље резултате у односу на први похлепни алгоритам. С обзиром на то да генетски алгоритам користи резултате добијене првим похлепним алгоритмом и као резултат даје боља решења, верујемо да би исто било и да смо користили резултате другог похлепног алгоритма.

Тестирање генетског алгоритма смо почели са малим скупом јединки унутар популације. Очекивано, алгоритам је давао лоша решења и брза извршавања. Затим смо популацију фиксирали на 100 јединки, што је успорило извршавање алгоритма али су решења била добра. Тактика додавања одређеног броја чворова са највише суседа побољшала је резултате генетског алгоритма.

Генетски алгоритам смо покретали четири пута по једној инстанци. А ограничење за заустављање алгоритма је да 20 пута направи итерацију, а затим да врати оно што је до тада нашао. Затим смо по потреби мењали број корака у зависности од квалитета претходног решења.

Литература

1. Увод у операциона истраживања - др Марко Ђукановић, др Драган Матић
2. https://epubs.siam.org/doi/pdf/10.1137/1.9781611973037.4 Approximating Fault-Tolerant Domination in General Graphs Klaus-Tycho Foerster
3. https://computationalsocialnetworks.springeropen.com/articles/10.1186/s40649-020-00078-5 Solving the k-dominating set problem on very large-scale networks
4. https://www.geeksforgeeks.org/dominant-set-of-a-graph/
5. https://sci-hub.ee/10.1016/j.cor.2021.105368 Heuristics for k-domination models of facility location problems in street networks
6. https://browse.arxiv.org/pdf/2111.07885.pdf Heuristics for k-domination models of facility location problems in street networks Padraig Corcoran Andrei Gagarin