Математички факултет
Универзитет у Београду

Мастер рад

Примена доминантних и независних скупова за конструкцију модела бежичих простих мрежа

Студент:
Марина Шљивић 1035/2017

Ментор:
др Александар Савић

септембар 2019.

Резиме

У овом раду ће бити представљене особине доминантности и независности у графу, проблеми максимум независног скупа и минимум доминантног скупа, као и њихова примена у конструкцији простих бежичних мрежа. Најпростији пример графова за изучавање модела простих бежичних мрежа су графови јединичног диска. Ипак, они се сматрају само теоријским моделима бежичних мрежа јер су тешко примењиви у стварном животу. Зато се као погоднији модел у овом раду разматрају графови ограниченог раста и потрага за максимум независним скупом и минимум доминантним скупом се одвија над графовима са овом особином.

У раду су приказани неки од алгоритама за конструкцију максималних независних скупова, док је посебно размотрен један, који максимални независни скуп за графове са особином ограниченог раста налази у времену $O(\log^* n)$, где је n број чворова графа. Разлог за разматрање брзог алгоритма за проналазак максималног независног скупа се огледа у чињеници да максимални независни скуп може послужити за конструкцију апроксимација максимум независног скупа и минимум доминатног скупа, при чему апроксимативна решења од тачних решења не одступају за више од $1 + \varepsilon$, за неку унапред задату тачност $\varepsilon > 0$. У раду ће бити размотрени и овакви апроксимативни алгоритми.

Кључне речи: Максимум независни скуп, минимум доминантни скуп, графови ограниченог раста, просте бежичне мреже

In this thesis will be discussed concepts of domination and independence in graphs, maximum independent set problem and minimum dominating set problem, as well as their application in construction of wireless Ad Hoc networks. The simplest graph model for wireless ad-hoc networks is the unit disk graph. Nevertheless, they are considered to be only theoretical models of wireless Ad Hoc networks because they are hardly applicable in real life. Therefore, bounded growth graphs are considered as a more appropriate model for Ad Hoc network, so in this thesis the problem will be analysed in the class of Growth Bounded Graphs.

The thesis presents some of the algorithms for finding maximal independent set, while one of them, which finds maximal independent set for graphs with this property in time $O(\log^* n)$, where n is number of graph nodes, is further discussed. The reason for searching for a fast algorithm for finding maximal independent set is that maximal independent set can serve for algorithms that creates a near-optimal solutions for maximum independent set and minimum dominant set, that is, solutions with an error of $1 + \varepsilon$, $\varepsilon > 0$, in efficient run time. These approximation algorithms also will be presented.

Key words: Maximum independent set, Minimum dominating set, Growth Bounded Graphs, wireless Ad Hoc networks

САДРЖАЈ

Увод

Бежичне технологије су од самог свог настанка биле једна од најатрактивнијих и најизучаванијих тема у области технологије. Разлог се огледа у огромној примени како у свакодневном животу, тако и у науци, индустрији и многим другим гранама. Услед толиког истраживања, унапређивања и потреба за побољшањем које би донело користи на разним пољима, као следећи степен развоја бежичних мрежа су настале просте бежичне мреже. Просте бежичне мреже представљају децентрализован тип бежичних мрежа. Не ослањају се на инфраструктуру, омогућавају да сваки уређај може функционисати као предајник, преносник или пријемник.

Моделовање простих бежичних мрежа се врши помоћу графова, где кључну улогу имају независни и доминантни скупови графова. Стабилни или независни скуп у графу представља подскуп скупа чворова графа где за произвољна два чвора из тог подскупа не постоји грана графа која их спаја. Другим речима, свака грана у графу ће имати бар један крај у независном скупу графа. Доминантни скуп у графу је подскуп скупа чворова графа где сваки чвор графа или припада том скупу или је суседан са неким чвором тог скупа. При формирању простих бежичних мрежа неопходна особина коју треба задовољити је успостављање комуникације - да сваки чвор посредно или непосредно може да комуницира са сваким другим чвором.

У простим бежичним мрежама доминантни скуп се у великој мери користи као виртуелна инфраструктура, односно виртуелна мрежа за пренос информација. Као оптимално решење трага се за оваквим скуповима најмање кардиналности. Многе конструкције за апроксимацију минимум доминантног скупа се заснивају на изградњи максималног независног скупа, што ће детаљније бити обрађено у овом раду.

1. Основне дефиниције и формулација проблема

1.1 Бежичне просте мреже

Просте бежичне мреже су децентрализоване мреже које се састоје од појединачних уређаја који међусобно комуницирају директно. Термин подразумева спонтану или импровизовану конструкцију јер се не ослања на постојећу инфраструктуру, као што су рутери у жичаним мрежама или приступне тачке у инфраструктурним бежичним мрежама. Због тога, многе просте бежичне мреже су локалног типа, где рачунари или други уређаји могу диретко да шаљу податке једни другима, а не кроз централизовану приступну тачку.

Идеја простих бежичних мрежа често није позната крајњим корисницима који су се сусретали само са малим стамбеним или пословним мрежама, које користе типични рутер за слање сигнала на појединачне рачунаре и уређаје. Међутим, просте мреже имају велику примену у новим типовима бежичног инжењеринга, иако је до скоро проста мрежа била прилично езотерична идеја. У подручијима где постоји мало или нимало комуникационе инфраструктуре, или је постојећа инфраструктура превише скупа или неодговарајућа да би се користила, урађаји могу комуницирати преко простих мрежа. Сваки уређај може слати податке, примати или преносити податке између друга два уређаја у простој мрежи. Ти уређаји се могу слободно кретати, па су просте мреже тип динамичких мрежа.

Неке од многих предности простих мрежа су могућност комуникације без ослањања на активну конекцију, потпуна независност од постојеће инфраструктуре, мобилност, а уз то су и јефтиније за изградњу јер не захтевају велику количину хардвера. Предност децентрализације се састоји у томе што не постоји централни уређај чијим престанком рада се потпуно губи мрежа код свих уређаја, већ ако један уређај напусти мрежу, остали уређаји остају умрежени и можда се може поново успоставити мрежа која покрива све уређаје, уколико су сви уређаји повезани и након искључивања једног од њих.

Поред многих предности постоје и мане простих мрежа које треба споменути. Иако је динамичност простих мрежа њена предност, уједно је и мана јер уређаји могу напуштати комуникационе опсеге других уређаја преко којих добијају или шаљу сигнал. Такође, великим бројем уређаја је тешко управљати без веће и конкретније инфраструктуре. Још једна мана је и безбедност, односно рањивост на различите врсте напада.

Неке од могућих примена простих мрежа су код студената који преко рачунара учествују у интерактивним предавањима, у корпоративним фирмама за дељење података између запослених, у приватним условима за размену података у одсуству интернет конекције. Једна од најпознатијих и најраспрострањенијих примена у приватним условима је блутут (енг. bluetooth) тј. технологија која омогућава бежични пренос података између уређаја који поседују исту технологију. Примењују се и у мобилним простим мрежама (познатим као MANET) које су самоорганизујуће бежичне мреже међу мобилним телефонима, бежичним простим мрежама у возилима (познатим као VANET) користећи путничке аутомобиле за конструкцију међусобне комуникације. Велика примена се огледа и у војној области, где се просте мреже користе на војним бојиштима за размену података међу војницима о стању, потенцијалним нападима и као кому-

никације са штабовима, јер на таквим местима обично не постоји могућност другачије врсте умрежавања. Додатан значај простих бежичних мрежа је у ванредним ситуацијама као што је помоћ у случају катастрофе. Брзо успостављање виртуелне инфраструктуре чини просте мреже лако употребљивим у случајевима пожара, земљотреса или других ванредних ситуација.

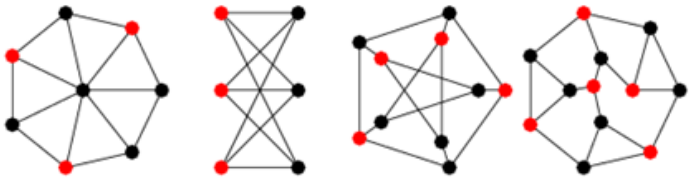
1.2 Основне дефиниције

За моделовање просте бежичне мреже користи се неоријентисани граф $G = (V, E)$ такав да су елементи скупа V чворови графа који представљају уређаје из бежичне мреже, док су елементи скупа E гране графа које представљају комуникациону везу између уређаја. Ако два уређаја могу директно да комуницирају један са другим, без посредовања других уређаја, њима одговарајући чворови у графу су повезани граном из скупа E. Илустрацију оваквог графа у простој бежичној мрежи је могуће видети на слици 1.1.

Слика 1.1. Илустрација графа у бежичној мрежи[1]

Следеће појмове независног скупа, максималног независног скупа (енг. maximal independent set) и максимум независног скупа (енг. maximum independent set) је увео Џек Едмондс у [12], док се појмови, као и следеће дефиниције могу пронаћи и у [6].

Дефиниција 1.1. *Нека је дат граф $G = (V, E)$. Подскуп чворова $V' \subseteq V$ називамо независним (стабилним) ако за свака два чвора $u, v \in V'$, не постоји грана $(u, v) \in E$.*



Слика 1.2. Примери независних скупова у графовима[2]

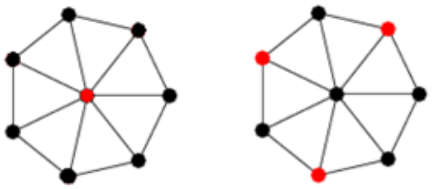
На слици 1.2. се могу видети примери независних скупова у графу. Црвеном бојом су обојени чворови који чине независне скупове у одговарајућим графовима.

Дефиниција 1.2. *Нека је дат граф $G = (V, E)$. За независни скуп кажемо да је максималан независни скуп ако губи својство независности додавањем било ког другог чвора графа, а ако је независни скуп највеће могуће кардиналности за дати граф називамо га максимум независни скуп.*

[1] Слика преузета из: [21]
[2] Слика преузета са: http://mathworld.wolfram.com/IndependentSet.html

Величина независног скупа представља број чворова које тај скуп садржи. Проблем налажења независног скупа највеће могуће кардиналности у графу називамо проблемом максимум независног скупа у графу (енг. Maximum Independent Set (Max-IS) problem) и он се сматра НП комплетним проблемом. Доказ НП комплетности Max-IS проблема је могуће пронаћи у [1]. Максимум независни скуп представља скуп добијен као решење овог проблема. Кардиналност максимум независног скупа се назива независан број и обележава се са $\alpha(G)$.

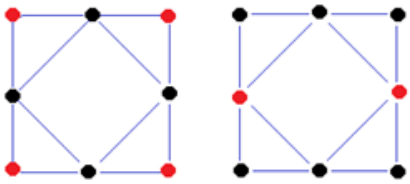
Видимо да се дефиниција максималног независног скупа не односи на кардиналност, већ само на структуру, па закључујемо да је максимум независни скуп увек и максимални независни скуп док обратно не мора да важи. То се може видети и на слици 1.3. где су црвеном бојом обојени чворови који чине максималне независне скупове за те графове. Први максимални независни скуп није и максимум за дати граф, док је други максимални скуп на слици уједно и максимум независни скуп у графу.



Слика 1.3. Пример максималног независног скупа који није максимум (слика лево) и максимум независног скупа (слика десно)

Поред независности у графу, потребно је дефинисати и доминантност. Следећу дефиницију је могуће пронаћи у [6].

Дефиниција 1.3. *Нека је дат граф $G = (V, E)$. Подскуп чворова $V' \subseteq V$ називамо доминантним ако за сваки чвор $v \in V$ важи да $v \in V'$ или да је v суседан неком чвору који припада V'.*



Слика 1.4. Пример доминантног скупа (слика лево) и минимум доминантног скупа (слика десно)

Проучавање доминације на графу је почело око 1950. године, док је сам појам доминантног скупа увео Остин Оре 1962. године [2]. Проблем одређивања доминантног скупа најмање кардиналности (енг. Minimum Dominating Set (Min-DS) problem) такође представља НП комплетан проблем, што је и могуће пронаћи у [1]. Доминантни скуп најмање кардиналности (енг. Minimum Dominating Set) називамо минимум доминантним скупом [6]. Кардиналност минимум доминантног скупа означавамо са $\gamma(G)$ и називамо доминантним бројем. На слици 1.4. је могуће видети пример доминантног скупа који није најмање кардиналности за дати граф лево и доминантног скупа најмање кардиналности за дати граф на слици десно.

Следећа теорема, коју је могуће пронаћи у [6], нам даје везу између описаних скупова:

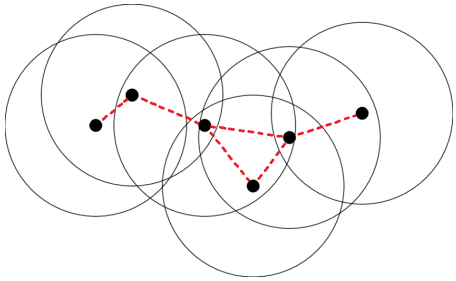
Теорема 1.1. *Сваки максимални независни скуп у графу $G = (V, E)$ је и доминантни скуп у том истом графу.*

Доказ. Нека је $V' \subseteq V$ максимални независни скуп у графу $G = (V, E)$. Претпоставимо супротно, да V' није доминантни скуп у том графу. То значи да постоји чвор $u \in V \setminus V'$ који није суседан ни са једним чвором из V'. Тада $W = V' \cup \{u\}$ представља независни скуп настао додавањем чвора на максимални независни скуп V'. Дефиниција максималног независног скупа каже да је максимални независни скуп скуп који додавањем било ког чвора графа губи својство независности, па је ово контрадикција, чиме је теорема доказана. □

1.3 Дефинисање проблема - Бежичне просте мреже и графови

Бежичне просте мреже представљају врсту мреже која није централизована, већ сваки уређај учествује у процесу преноса. Сваки уређај може да комуницира само са другим уређајем који је унутар његовог преносног опсега. У графовима за моделовање простих мрежа чворови представљају уређаје, а гране означавају комуникациону везу између уређаја. Та комуникација између уређаја може бити директна када сваки чвор може да прими информацију од сваког другог чвора или се може одвијати преко чворова преносника када су чвор пошиљалац и чвор прималац ван домета један другом. Тако настаје такозвана multi-hop комуникација, тј. комуникација која је могућа и ван комуникационог опсега уређаја. Како у простим мрежама не постоји централни сервер нити фиксна инфраструктура, оваква мрежа мора бити добро организована и динамична (чворови се могу слободно кретати).

Циљ је представити мрежу помоћу графова и геометријских региона на којима је пријем сигнала могућ. Најједноставнији и најистраживанији модел су такозвани графови облика јединичног диска (енг. Unit Disk Graph) у ком су чворови тачке у равни, а два чвора су суседна у графу ако и само ако је њихова Еуклидска удаљеност највише 1 (видети слику 1.5). Порука може бити послата било ком другом чвору у преносном опсегу. Ипак, овај модел је превише поједностављен да би био потпуно примењив у реалном животу. У природним условима уређаји скоро никада не преносе сигнале у савршеним круговима, па је потребно увести неке сложеније моделе за моделовање реалних ситуација.



Слика 1.5.[6] Пример графа облика јединичног диска

Главна идеја у анализи бежичних мрежа уз помоћ графова у овом раду је заснована на стварању подскупова скупа чворова у графу, који имају одређене повољне особине и примени добијених резултата за конструкцију модела саме просте бежичне мреже. Две основне особине које испитујемо у подскуповима графова су:

- Независност: Кажемо да подскуп чворова има ову особину ако не постоје два чвора у том скупу која су повезана међусобно граном. Претпоставка је да чворови у независном скупу не ометају једни друге током истовременог преноса.
- Доминантност: Кажемо да подскуп чворова има ову особину ако је сваки чвор из мреже уједно и чвор доминантног скупа или је повезан са бар једним чвором

доминантног скупа. Чворови доминантног скупа се могу користити за ефикасан пренос података кроз целу мрежу користећи за емитере само чворове из овог скупа.

Један од основних проблема који треба размотрити у овом раду је примена независних и доминантних скупова у графу за конструкцију виртуалне инфраструктуре за просте бежичне мреже. То произилази из чињенице да се за просте бежичне мреже, као виртуелна инфраструктура, обично користе доминантни скупови. Због оптимизације проблема, утрошка материјала, оптималне производње и функционалније мреже, циљ је пронаћи доминантни скуп што мање кардиналности - минимум доминантни скуп. Као што је раније наведено, проблем конструкције минимум доминантног скупа је НП комплетан проблем. Зато се у овом раду прибегава алгоритмима који ће у полиномијалном времену пронаћи скуп који је доминантан и чија се кардиналност од кардиналности минимум доминантног скупа не разликује за више од $1+\varepsilon$, где је $\varepsilon > 0$ унапред задата тачност.

У наредном поглављу ће бити представљени неки од алгоритама за проналажење максималног независног скупа, као основе за многе апроксимативне алгоритме. Даље ће бити размотрене особине графова које можемо узети као погодне за графове који представљају моделе простих бежичних мрежа.

Као централни део, у четвртом поглављу ће бити представљен брз и стабилан алгоритам за проналажење максимланог независног скупа у графу са особином ограниченог раста. Алгоритам ће бити анализиран уз добијене експерименталне резултате. За сам алгоритам ће бити дат и предлог имплементације у програмском језику C# . У наредном поглављу ће бити представљене примене овог алгоритма за конструкцију независних и доминантних скупова у графу чије се кардиналности од кардиналности максимум независног и минимум доминантног скупа не разликују за више од $1+\varepsilon$, где је $\varepsilon > 0$.

Како је већ наведено, појам доминантног скупа увео је Остин Оре 1962. године. Ипак, проучавање доминације у графу се интензивније проучава од краја 1950. и почетка 1960. године[10]. Врло брзо након почетка разматрања проблема доминације, јавља се и идеја о оптимизацији овог проблема која доводи до минимум доминантног скупа. Овај проблем је 1977. разматран у [11]. Као што је већ поменуто, године 1964. Џек Едмондс у [12] дефинише појмове максималан независни скуп и максимум независни скуп.

Када је у питању примена доминантних и независних скупова у простим бежичним мрежама, велики број радова узима у обзир само графове облика јединичног диска као графове за моделовање простих бежичних мрежа ([13], [27], [28]). Ипак, колико је познато, чак и у случају графа облика јединичног диска, са датом геометријском репрезентацијом, полиномијални алгоритам за решавање проблема Max-IS и Min-DS није познат у литератури [14].

2. Похлепни алгоритми за контрукцију максималних независних скупова

Како је већ речено, конструкције максимум независног скупа и минимум доминантног скупа представљају НП комплетне проблеме. Ако код независних скупова оставимо по страни услов највеће кардиналности, за конструкцију максималних независних скупова постоје бројни алгоритми. У овом поглављу ћемо размотрити конструкцију максималних независних скупова у графу помоћу неких грамзивих алгоритама. Грамзиви (енг. Greedy) алгоритам је сваки алгоритам који решава проблем бирањем локално оптималног решења у сваком кораку, у нади да ће на тај начин доћи до глобалног оптимума. Наведимо најпре дефиницију околине, коју је могуће пронаћи у [6]:

Дефиниција 2.1. *Под отвореном околином чвора $v \in V$ графа $G = (V, E)$ подразумевамо скуп $\Gamma(v) := \{u \in V | (u, v) \in E\}$, док под затвореном околином чвора v подразумевамо скуп $\overline{\Gamma}(v) := \{u \in V | (u, v) \in E\} \cup \{v\}$.*

2.1 Секвенцијални алгоритам и његова модификација

За решавање проблема проналаска максималног независног скупа у неоријентисаном графу је могуће користити похлепни секвенцијални алгоритам који ће бити представљен у овом поглављу као први алгоритам. Алгоритам пролази кроз чворове датог графа произвољним редоследом додајући чвор резултујућем скупу ако претходно није додат ниједан његов сусед. На тај начин се као резултат добија максимални независни скуп. Овај алгоритам је први пут описан у [6].

Ако пођемо од саме дефиниције независног скупа, како тражимо скуп такав да за свака два чвора тог скупа не постоји грана која их спаја, јасно се намеће идеја додавања несуседних чворова тренутном решењу све док то не постане немогуће. Описани поступак се може приказати следећим псеудо кодом:

Алгоритам 1.

Улазни параметар: Неоријентисани граф $G = (V, E)$
1. Иницијализовати $I := \emptyset$. Ако постоје изоловани чворови додати их у I
2. **If** V празан скуп **then** ићи на корак 5.
   else изабрати $v \in V$
   end if
3. $I = I \cup \{v\}$, $V = V \setminus \overline{\Gamma}(v)$
4. Вратити се на корак 2
5. Скуп I је максимални независни скуп

У кораку иницијализације полазимо од празног скупа I коме, уколико постоје, додајемо све изоловане чворове графа, тј. чворове који немају ниједан суседни чвор. Током алгоритма, скупу I додајемо чворове из скупа V који нису већ додати, али тако

да не кршимо особину независности. Описаним поступком скуп I, који добијамо као резултат рада алгоритма, је независан и максималан зато што се чворови из скупа V, а који имају суседа у скупу I, не разматрају. Временска сложеност овог алгоритма је $O(m+n)$, где је $m = |E|, n = |V|$.

Оно што није прецизно дефинисано у алгоритму 1 је избор чвора v у кораку 2. Постоји више начина за избор овог чвора. У [15] се чворови не бирају на случајан начин, већ су сви чворови уређени према одређеном правилу, док се избор врши према том уређењу.

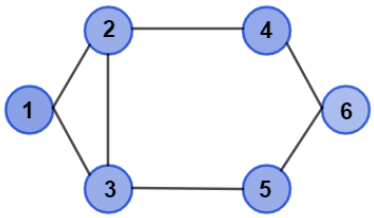
У наставку ће бити представљена друга модификација, која поредак међу чворовима формира на основу степена самих чворова, а у кораку 2 за чвор v бира чвор најмањег степена. Степен чвора v представља број његових суседа и означава се са d_v. У случају да постоји више чворова са истим степеном, један од њих се бира на случајан начин. Овакав избор чворова има за циљ да у свакој итерацији из скупа чворова који се посматрају избаци што мањи број чворова, са тежњом да резултујући скуп буде што је могуће веће кардиналности.

На овај начин добијамо незнатно модификовани алгоритам:

Алгоритам 2.

Улазни параметар: Неоријентисани граф $G = (V, E)$
1. Иницијализовати $I := \emptyset$. Ако постоје изоловани чворови додати их у I
2. **If** V празан скуп **then** ићи на корак 5.
   else изабрати $v \in V$ тако да је v чвор најмањег степена у V
   end if
3. $I = I \cup \{v\}$, $V = V \setminus \overline{\Gamma}(v)$
4. Вратити се на корак 2
5. Скуп I је максимални независни скуп

Пример 1. Погледајмо рад алгоритма 2 на конкретном примеру графа задатог на слици 2.1.

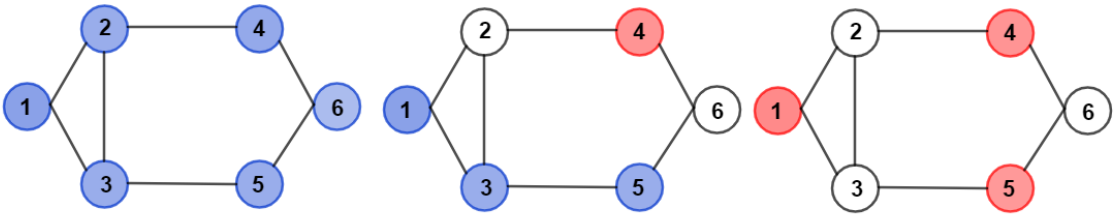


Слика 2.1.

Нека чвору 1 одговара ознака v_1, чвору 2 ознака v_2 и за остале чворове аналогно. Како граф нема изолованих чворова, нека је након иницијализације скуп I празан. У првој итерацији за чвор v у кораку 2 можемо узети неки од чворова v_1, v_4, v_5 или v_6 јер су степена 2. Како чворови имају исти степен, избор једног од њих се изводи на случајан начин.

У првом случају, нека је чвор v_4 на случајан начин добио приоритет у односу на друге поменуте чворове истог степена. На тај начин добијамо $I = \{v_4\}, V = \{v_1, v_3, v_5\}$. Како V није празан, а чворови v_1 и v_5 су степена 1, у наредне две итерације та два чвора улазе у I. Крајњи резултат је $I = \{v_1, v_4, v_5\}$ што је и приказано слици 2.2. Слика лево представља полазни граф, слика у средини представља стање након прве итерације,

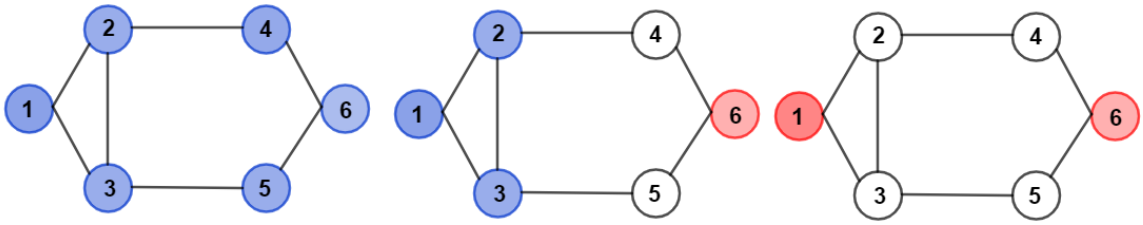
где је чвор v_4 ушао у резултујући скуп и слика десно представља резултат, где чворови обојени црвеном бојом чине максимални независни скуп полазног графа.



Слика 2.2.

На овом једноставном примеру, видимо да смо добили не само максимални независни скуп већ и максимум независни скуп. Ипак, приказани алгоритам не даје увек максимум независни скуп, већ може гарантовати само да је добијени скуп максимални независни скуп у графу. То можемо видети применом истог алгоритма ако у првој итерацији у кораку 2 на случајан начин изаберемо неки други чвор који ће имати приоритет у односу на остале чворове истог степена.

Ако у првом проласку уместо чвора v_4 изаберемо чвор v_6, истим поступком као излазни параметар добијамо максимални независни скуп који није маскимум (видети слику 2.3). На слици лево је поново дат почетни граф, док је на слици у средини приказано стање након прве итерације, где чвор v_6, обојен црвеном бојом, улази у резултујући скуп. На слици десно је приказан резултат након друге итерације, а уједно и крајњи резултат, где чворови обојени црвеном бојом чине резултујући скуп - максимални независни скуп који није максимум.



Слика 2.3.

2.2 Паралелни алгоритам са случајним приоритетом - Лубијев алгоритам

Размотримо сада један такође једноставан али ефикаснији алгоритам за добијање максималног независног скупа у графу. Овај алгоритам је први предложио 1986. године Михаел Луби [16].

У сваком проласку кроз петљу, сваком чвору текућег скупа V се додељује случајна вредност из интервала [0,1]. Све чворове $v \in V$ који имају мању случајну вредност од свих својих суседа додајемо независном скупу, док из скупа чворова који се даље разматрају бришемо све изабране чворове $v \in V$ као и све њихове суседе. Брисањем суседних чворова се постиже особина независности.

Како се алгоритам извршава паралелно, а не секвенцијално, потребно је нагласити да није могућ ни улаз два суседна чвора у истој итерацији у резултујући скуп, јер улази само чвор који има строго најмању случајну вредност у односу на све своје суседне чворове. Ако два суседна чвора добију исту случајну вредност, ниједан неће ући у резултујући скуп у тој итерацији, јер случајна вредност тих чворова тада сигурно није мања од случајних вредности свих суседних чворова. На основу овога је јасно да је

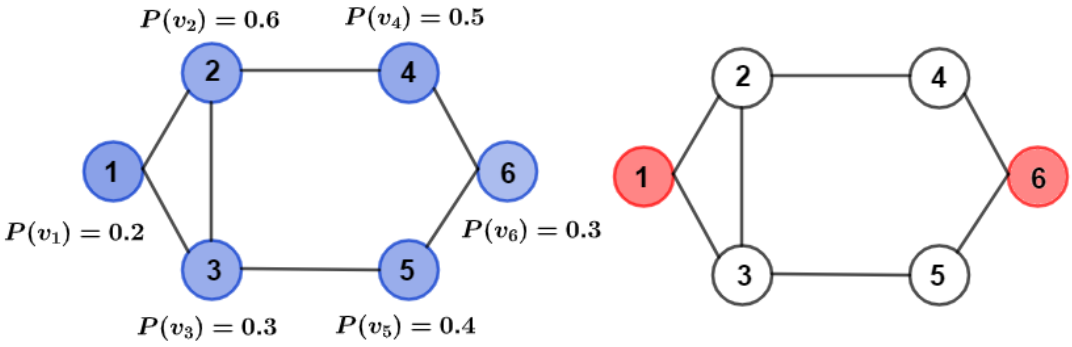
ситуација уласка два суседна чвора паралелно у резултујући скуп немогућа. Дефинишимо алгоритам:

Алгоритам 3.

Улазни параметар: Неоријентисани граф $G = (V, E)$
1. Иницијализовати $I := \emptyset$. Ако постоје изоловани чворови додати их у I
2. **If** V празан скуп **then** ићи на корак 6
   end if
3. **For** свако $v \in V$
   изабрати случајну вредност $P(v) \in [0, 1]$
   end for
4. Паралелно за све v за које $P(v) < P(w), \forall w \in \Gamma(v) : I = I \cup \{v\}, V = V \setminus \overline{\Gamma}(v)$
5. Вратити се на корак 2
6. Скуп I је максимални независни скуп

У односу на секвенцијалне алгоритме, у овом алгоритму је могуће додавање више чворова резултујућем скупу истовремено, што је предност алгоритма. Коректност алгоритма је аналогна као и код претходних, јер увек додајемо независне чворове, па скуп који добијамо има својство независности, а било који други чвор који би додали би нарушио то својство јер је суседан са неким од раније додатих чворова. Показано је у [16] да је временска сложеност овог алгоритма $O(\log n)$. Погледајмо рад алгоритма 3 на конкретном примеру графа задатог на слици 2.1.

Пример 2. Као и у првом примеру, нека чвору 1 одговара ознака v_1 и за остале чворове аналогно. Како граф нема изоловане чворове, након иницијализације скуп I је празан. У првој итерацији се у кораку 3 свим чворовима додељује случајна вредност (видети слику 2.4 лево).



Слика 2.4.

У кораку 4 се паралелно упоређују додељене случајне вредности свих чворова. Како је $P(v_1) < P(v_2)$ и $P(v_1) < P(v_3)$, чвор v_1 има најмању случајну вредност међу свим својим суседима. Паралелно важи да је и $P(v_6) < P(v_4)$ и $P(v_6) < P(v_5)$ па и чвор v_6 има најмању случајну вредност међу свим својим суседима. На тај начин чворови v_1 и v_6 истовремено улазе у резултујући скуп I током прве итерације у кораку 4. Та два чвора, као и сви њихови суседи, се уклањају из скупа чворова за даље разматрање. Како више нема чворова за даље разматрање, алгоритам се завршава. На слици 2.4 десно је могуће видети добијени максимални независни скуп за задати граф где су чворови овог скупа обојени црвеном бојом.

3. Графови ограниченог раста

Као што је већ речено, најпростији и најизучаванији пример за моделовање простих комуникационих мрежа су графови познати под именом графови облика јединичног диска. За граф $G = (V, E)$ кажемо да је граф облика јединичног диска ако се за сваки чвор $v \in V$ може поставити тачка p_v у Еуклидској равни $\mathbb{R}^2$ тако да грана $(u, v) \in E$ постоји ако и само ако је растојање $\| p_v - p_u \|_2 \leq 1$. Каже се да они представљају теоријске моделе бежичних мрежа. Ипак, они нису најбоље решење за моделовање простих мрежа, јер представљају идеалну ситуацију, тешко у потпуности примењиву у стварном животу. Иако су се графови јединичног диска показали као делимично добар модел комуникационих мрежа, главни недостатак представља услов строге геометријске структуре, а посебно зато што мрежна инфраструктура скоро никада нема потпуно исти комуникациони опсег међу уређајима.

Погоднију структуру графова у односу на графове облика јединичног диска налазимо код графовима ограниченог раста (енг. Bounded growth graphs). Ова уопштенија фамилија графова је заснована на претпоставци да чворови који се налазе у непосредној близини морају бити у међусобном опсегу преноса. Да би дефинисали граф ограниченог раста, дефинишимо прво r-околину. Следеће дефиниције су преузете из [6].

Дефиниција 3.1. *Са $d_G(u, v)$ означимо број грана које чине најкраћи пут између чворова u, v у графу $G = (V, E)$. За чвор v његову r-околину, за $r > 0$, дефинишемо са:* $\Gamma_r(v) := \{u \in V | d_G(u, v) \leq r\}$.

У теорији графова, граф је ограниченог раста ако је број независних чворова у r-околини ограничен, односно формално дефинисано:

Дефиниција 3.2. *За граф $G = (V, E)$ кажемо да је p-ограниченог раста ако постоји ограничавајућа функција $p(r)$ таква да свака $\Gamma_r(v)$ околина за свако $v \in V$ садржи највише $p(r)$ независних чворова.*

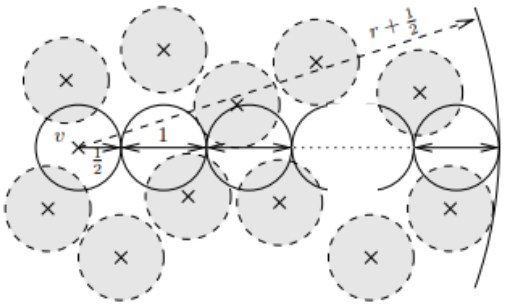
Дефиниција 3.3. *За граф $G = (V, E)$ p-ограниченог раста кажемо да је полиномијално ограничен растом ако је функција раста, за неку константу $c \geq 1$, ограничена полиномом максималног степена c, односно $p(r) = O(r^c)$.*

Више о графовима полиномијално ограниченог раста је могуће видети у [29]. Како из дефиниција видимо, ако је граф G p-ограничен растом тада постоји функција $p(r)$ таква да за свако $v \in V$ у $\Gamma_r(v)$ постоји највише $p(r)$ независних чворова, тј. максимум независни скуп у околини $\Gamma_r(v)$ има највише $p(r)$ елемената, за свако $r \geq 0$, где је r из скупа природних бројева. Функцију $p(.)$ називамо функцијом раста и приметимо да она не зависи од броја чворова графа G већ само од удаљености r. Односно, за константно r, број независних чворова у r-околини је константан.

У последње време, при моделовању бежичних простих мрежа, све више се користе графови ограниченог раста као графови који добро представљају моделе бежичних простих мрежа ([4], [7], [17], [23]). Предност моделовања простих мрежа коришћењем графова ограниченог раста на одређеним географским положајима се огледа у томе што уређаји који су у непосредној близини лако примају сигнале један од другог и тако

омогућавају комуникацију између уређаја веће међусобне удаљености међу којима је директна комуникација често онемогућена услед ограниченог домета сингала.

Покажимо сада да су графови облика јединичног диска полиномијално ограниченог раста са функцијом раста $p(r) = O(r^2)$. Изаберимо произвољно $v \in V$ и произвољан независни скуп $I \subseteq \Gamma_r(v)$ за $r \geq 0$. Како су у I сви чворови независни, раздаљина између свака два чвора у I мора бити већа од 1. Тако свака тачка која одговара чвору у I заузима диск полупречника $\frac{1}{2}$ и површине $P = (\frac{1}{2})^2\pi$. Област у којој сви дискови леже због r-околине мора бити полупречника $r + \frac{1}{2}$, са центром у изабраном чвору v. Објашњење за ово видимо на слици 3.1.



Слика 3.1[6]. Дистанца графа облика јединичног диска у r-околини

Тако долазимо до неједначине $|I| \leq \frac{(r+\frac{1}{2})^2\pi}{(\frac{1}{2})^2\pi}$, односно $|I| \leq (2r+1)^2$.

3.1 Независни и доминантни скупови у графовима ограниченог раста

Посматрајмо граф $G = (V, E)$ p-ограниченог раста. Следеће две теореме, формулисане и доказане у [6], показују да максимални независни скуп представља апроксимацију константног фактора за проблеме максимум независног и минимум доминантног скупа у случају графа ограниченог раста.

Теорема 3.1. *Нека је дат граф $G = (V, E)$ p-ограниченог раста и нека је I максимални независни скуп, а I^* максимум независни скуп у G. Тада важи:*

$$|I^*| \geq |I| \geq \frac{|I^*|}{p(1)}.$$

Доказ. Прва неједнакост $|I^*| \geq |I|$ очигледно важи на основу дефиниција максималног и максимум независног скупа. За доказ друге неједнакости посматрајмо произвољно $v \in I^*$. Тада v или припада I или је сусед неког чвора из I. Ово следи из Теореме 1.1. односно на основу чињенице да је максимални независни скуп уједно и доминантан. Како за оне чворове v за које $v \notin I$ мора постојати њихов сусед који је у I, у циљу упоређивања кардиналности скупова I и I^* формирајмо функцију бројања за $v \in I^*$ на следећи начин: ако је $v \in I$ бројимо чвор на основу њега самог, у супротном ако чвор $v \notin I$, бројимо га на основу његовог суседа који је у I.

За било који чвор $v \in V$ ако посматрамо околину сачињену од његових директних суседа, односно $\Gamma_1(v)$ на основу дефиниције графова ограниченог раста, закључујемо да та околина може имати највише $p(1)$ независних чворова. Дакле, сваки чвор $v \in I$ може бити избројан на описани начин највише $p(1)$ пута за чворове из оптималног решења I^*. Одатле следи друга неједнакост $|I| \geq \frac{|I^*|}{p(1)}$ чиме је ова теорема и доказана. □

На сличан начин је могуће показати да у минималном доминантом скупу D^* сваки чвор може доминирати над највише $p(1)$ чворова из I, односно аналогно са претходним тврђењем можемо доказати и следећу теорему, чији ће доказ из тог разлога бити изостављен:

Теорема 3.2. *Нека је дат граф $G = (V, E)$ p-ограниченог раста и нека је I максимални независни скуп, а D^* минимум доминантни скуп у G. Тада важи:*

$$p(1) \cdot |D^*| \geq |I| \geq |D^*|.$$

Претходне Теореме нам нису само показале апроксимацију са константним фактором проблема максимум независног и минимум доминантног скупа преко максималног независног скупа већ закључујемо да су кардиналности максималног и максимум независног скупа различите само за фактор од $p(1)$, а како је максимум независни скуп такође и максималан, овај однос се одржава и између оптималних решења проблема максимум независног и минимум доминантног скупа на истом графу.

Дакле, закључили смо да нам код графова ограниченог раста проблем максималног независног скупа може играти битну улогу у апроксимацији решења НП комплетних проблема Max-IS и Min-DS, а сада је показано и да скупови добијени као решења ова два проблема имају кардиналности које се међусобно разликују само за фактор $p(1)$. Напоменимо још само да код неких специфичних класа графова ограниченог раста можемо добити и прецизније процене кардиналности од процена које су изведене у теореми 3.1. и теореми 3.2.

4. Максимални независни скуп у графовима ограниченог раста

Као што је већ истакнуто, максимални независни скупови имају важну улогу за конструисање модела виртуелне инфраструктуре простих бежичних мрежа. То је разлог за разматрање неких бржих, стабилних и сигурних алгоритама за проналажење максималног независног скупа у графу. Како се показало да графови ограниченог раста добро представљају особине бежичних мрежа, фокусираћемо се на графове са овом особином у даљем тексту.

Алгоритам који ће бити представљен су предложили Јоханес Шнајдер и Роџер Ватенхофер у свом раду [8]. Роџер и Јоханес су професори на Циришком универзитету науке, технологије и математике и обојица се баве истраживањем проблема информационих технологија и утицаја тих технологија на савремени живот. Алгоритам који су они предложили завршава свој рад у времену $O(\log^* n)$, где је n број чворова графа, а функција $\log^* n$ одређена следећом дефиницијом:

Дефиниција 4.1. *Функција* $\log^*()$ *је дефинисана рекурзивно формулом:* $\log^* n = 1 + \log^* \lceil \log n \rceil$ *за* $n > 2$, *где је* $\log^* 0 = \log^* 1 = \log^* 2 = 0$.

Напоменимо да је логаритам ($\log n$) овде битовски логаритам, са основом 2.

4.1 Алгоритам за конструкцију максималног независног скупа

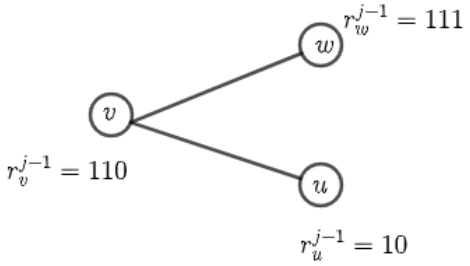
4.1.1 Основна идеја

На почетку алгоритма сваком чвору графа се додељује јединствен серијски број - ID. Он омогућава јединствену идентификацију чвора и представља се у бинарном облику. ID-еви чворова се састоје од l битова, где је $l \geq 0$, и имају следећи облик: $x^l x^{l-1} x^{l-2} ... x^1$, $x^i \in \{0, 1\}, i = 1, .., l$. Напоменимо да је алгоритам униформан, односно да не захтева информацију о броју чворова у графу као и да је робустан, у смислу да је коректан за произвољне графове, не само за графове ограниченог раста. За графове који немају особину ограниченог раста се не може гарантовати временска сложеност као у случајевима када ова особина постоји.

Сваки чвор током рада алгоритма учествује у низу такмичења са себи суседним чворовима. Идеја је да кроз такмичења, по одређеним правилима, неки чворови престају да буду разматрани, све док на тај начин у графу не остану само чворови који чине максимални независни скуп. Размотримо како изгледа такмичење између чворова. За ту потребу уводимо променљиву r_v^j која представља резултат такмичења чвора v са својим суседима у итерацији j. Резултати такмичења, попут ID-ева, се такође представљају у бинарном облику. Резултат такмичења у једној итерацији представља основу за следећу итерацију такмичења. На почетку такмичења вредности резултата такмичења сваког чвора су једнаке вредностима њихових ID-ева, односно: $r_v^0 := ID_v$ за сваки чвор v.

Такмичење чвора v са његовим суседима у некој итерацији j подразумева најпре проналажење суседног чвора u са најмањим резултатом из претходне итерације r_u^{j-1} од свих суседних чворова, а затим и формирање резултата r_v^j такмичења за чвор v. Резултат такмичења у j-тој итерацији, r_v^j, за чвор v се формира на следећи начин: ако

је $r_v^{j-1} \leq r_u^{j-1}$ то значи да ниједан чвор суседан чвору v нема строго мањи резултат такмичења у односу на резултат такмичења придружен чвору v, па у том случају $r_v^j := 0$. Ако услов није испуњен, односно $r_v^{j-1} > r_u^{j-1}$, тада резултат добија бинарну вредност највеће позиције (прве позиције гледано са лева на десно) на којој се битови резултата такмичења на тим позицијама разликују. Покажимо то и на примеру графа задатог на слици 4.1.



Слика 4.1. Пример рачунања резултата такмичења чвора v са својим суседима u, w

Ако посматрамо део графа који је приказан на слици 4.1, чвор v у такмичењу са својим суседима u, w прво проналази суседни чвор најмањег резултата и то је u. Како $r_v^{j-1} > r_u^{j-1}$, тражимо бит на највећој позицији на којој се ова два резултата разликују. Пошто два резултата немају исти број битова, онај са мањим бројем битова допуњујемо нулама на позицијама највеће тежине. Тако долазимо до поређења резултата 110 и 010. Позиција највеће тежине на којој се битови разликују је 3, што у бинарном облику представља запис 11. Дакле $r_v^j = 11$.

Формалним записом резултат такмичења се добија на следећи начин:

$$r_v^j = \begin{cases} max\left\{k | y_v^k > y_u^k\right\} \text{ у бинарном запису}, & r_v^{j-1} > r_u^{j-1} \\ 0, & \text{иначе} \end{cases},$$

где су резултати r_v^j облика: $r_v^j = y_v^t y_v^{t-1} ... y_v^1$, $y_v^i \in \{0, 1\}, i = 1, .., t$, $t \leq l$.

Поред резултата и ID-а сваки чвор v у сваком тренутку алгоритма има и своје стање s_v. У алгоритму постоји пет стања: **такмичар, доминатор, доминиран, владар, савладан** (енг. competitor, dominator, dominated, ruler, ruled). На почетку сви чворови имају стање такмичара и кроз алгоритам само чворови са таквим стањем учествују у такмичењу. На крају сваке итерације такмичења улази се у део алгоритма (функција $PromenaStanja()$) који мења стања чворовима на основу резултата добијених у тој итерацији тамичења. То се дешава по следећим правилима: Ако чвор v у итерацији j има резултат r_v^j који је строго мањи од резултата свих суседних чворова у итерацији j, тада чвор v добија статус доминатора. У супротном, проверава се да ли је резултат r_v^j мањи или једнак од резултата свих суседних чворова. Ако то јесте, чвор v постаје владар. Дакле, чвор добија стање доминатора ако је његов резултат мањи од резултата свих суседних чворова и међу суседима не постоји чвор који има исти резултат, док стање владара добија ако међу суседним чворовима постоји бар један који има исти резултат, док сви остали суседни чворови имају строго већи резултат. Чворови који су суседни чвору у стању доминатор добијају стање доминиран. Доминатори и доминирани чворови више не учествују у даљим такмичењима. Када се алгоритам заврши сви чворови у графу имају стање или доминатор или доминиран и сви они који су у стању доминатора граде максимални независни скуп полазног графа.

Када неки чвор добије стање владара не учествује у такмичењима све до наредне итерације дела алгоритма који називамо фазом. Суседи владара који су у стању такмичара постају савладани. Сваки чвор који још увек активно учествује у алгоритму у

оквиру фазе мења своје стање. Фаза се за један чвор завршава када у његовој околини више не постоје чворови који су у стању такмичара. На почетку нове фазе, за чворове који још увек учествују у алгоритму, односно нису ни доминатори ни доминирани, итерације се ресетују и почетни резултати такмичења опет добијају вредности ID-ева чворова, док сви чворови који су у стању владара прелазе у стање такмичара. Напоменимо да ID-еви јединствено одређују чворове и њихове вредности се не мењају током рада алгоритма.

4.1.2 Алгоритам

Како смо представили основне идеје алгоритма, у овом одељку следи псеудо код самог алгоритма као и илустрација његовог рада кроз пример.

Алгоритам такмичења [8]

Улазни параметар: Неоријентисани граф $G = (V, E)$
1. За сваки чвор $v \in V$ **repeat**:
2. {Почетак етапе} $s_v :=$ такмичар
3. **repeat**:
4. {Почетак фазе} $r_v^0 := ID_v$
5. **If** $s_v =$ владар **then** $s_v :=$ такмичар
end if
6. $j := 0$
7. **repeat**:
8. {Почетак такмичења} $j := j + 1$
9. **If** $s_v =$ такмичар **then** изабрати такмичара $u \in \Gamma(v)$ таквог да $r_u^{j-1} = \min_{w \in \Gamma(v)} r_w^{j-1}$ и поставити $r_v^j := \max(\{k | (y_v^k > y_u^k) \wedge (r_v^{j-1} > r_u^{j-1})\} \cup \{0\})$
end if
10. $PromenaStanja(s_v)$ {Завршетак такмичења}
11. **until** $\nexists u \in (\Gamma(v) \cup v)$ такво да $s_u =$ такмичар {Завршетак фазе}
12. **until** $\nexists u \in (\Gamma(v) \cup v)$ такво да $s_u =$ владар {Завршетак етапе}
13. **until** $s_v \in$ {доминатор, доминиран}

Алгоритам $PromenaStanja(s_v)$

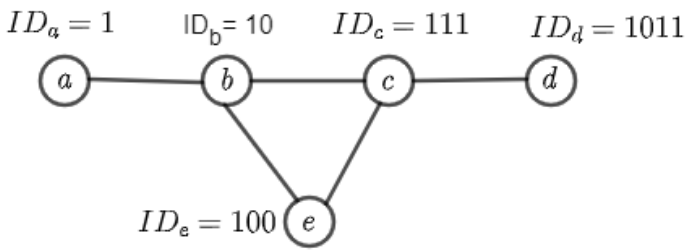
---

1. **If** $s_v =$ такмичар **then**
2. **If** $\forall t \in T$ ($T \subseteq \Gamma(v)$ скуп свих чворова са стањем такмичар) важи $r_t^{j-1} > r_v^{j-1}$ **then** $s_v :=$ доминатор
3. **else if** $\forall t \in T$ важи $r_t^{j-1} \geq r_v^{j-1}$ **then** $s_v :=$ владар
end if
end if
4. **If** $\exists t \in \Gamma(v)$ такво да $s_t =$ доминатор **then** $s_v :=$ доминиран
5. **else if** ако $(s_v \neq$ владар$) \wedge (\exists t \in \Gamma(v)$ такво да $s_t =$ владар) **then** $s_v :=$ савладан
6. **end if**

Потребно је нагласити да чвор изводи етапе, фазе и такмичења симултано са својим суседима. Да би се то извело потребно је синхронизовати резултате - глобално или локално. Код глобалне синхронизације чека се да сваки чвор заврши своје кораке па

се тек онда прелази на следећу фазу за све чворове. Ово гарантује синхронизацију на глобалном нивоу али и непотребно успоравање рада алгоритма. Ако се синхронизација врши локално, поруке се могу размењивати потпуно асинхроно. Једино ограничење је што чворови морају чекати док њихови суседи не буду спремни за следећу итерацију. Ово може довести до ситуације да синхронизација није постигнута на глобалном нивоу, али је довољна њена испуњеност на локалном нивоу.

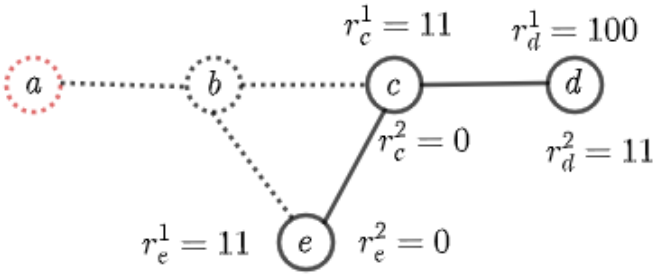
Илуструјмо рад алгоритма на примеру. Посматрајмо део графа представљен на слици 4.2.:



Слика 4.2.

На почетку алгоритма сви чворови добијају статус такмичара на почетку етапе и почетни резултат $r^0 = ID$ на почетку фазе. Пошто у почетном тренутку нема владара улази се у део алгоритма назван такмичење. За чвор a нови резултат је $r_a^1 = 0$ јер $r_a^0 < r_b^0$, а b је једини сусед чвора a. При такмичењу чвора b његов сусед који је такмичар са најмањим резултатом је чвор a, њихови резултати се разликују на другом биту, па је $r_b^1 = 10$. Аналогно $r_c^1 = 11, r_e^1 = 11$ и $r_d^1 = 100$. Алгоритам се наставља кроз функцију $PromenaStanja()$.

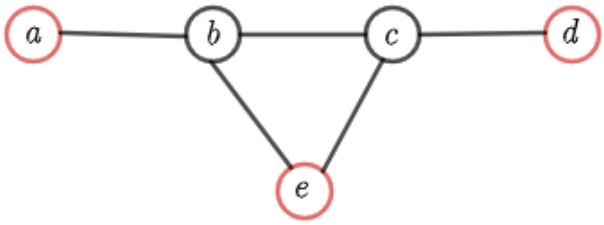
Оно што можда није тако очигледно на основу псеудо кода алгоритма, а треба истаћи, нови резултати се користе тек у следећој итерацији. Ако итерације бројимо од прве, за део алгоритма који се назива $PromenaStanja()$ у првој итерацији користимо иницијалне резултате r^0, у другој итерацији користимо резултате срачунате у првој и тако даље. У првој итерацији тако у делу $PromenaStanja()$ чвор a постаје доминатор а чвор b доминиран. Остали чворови не мењају стање и за њих се алгоритам наставља док чворови a и b више не учествују у раду алгоритма. У следећој итерацији такмичења добијени резултати су приказани на слици 4.3. Чвор a је обојен црвеном бојом јер је постао доминатор, док је чвор b тиме постао доминиран.



Слика 4.3.

У другој итерацији такмичења делу, $PromenaStanja()$ чворови c и e постају владари јер $r_c^1 < r_d^1$ и $r_c^1 = r_e^1$. Чвор d постаје савладан и неко време остаје миран. Како више нема чворова са статусом такмичара фаза се завршава, почиње нова фаза и у њој сви чворови који су владари поново постају такмичари са почетним резултатима једнаким њиховим ID-евима. Видимо да савладан чвор тренутно не учествује. Понављајући

поступак за чворове c и e чвор e постаје доминатор, а c доминиран. Како нема више такмичара завршава се фаза, а како нема ни владара завршава се и етапа. За преостали чвор d алгоритам почиње поново и он у њему постаје доминатор, зато што има најамњи ID међу суседима, чиме се алгоритам завршава. Описаним поступком пронађена су 3 доминатора која уједно чине и максимални независни скуп. Резултат рада алгоритма 3 је илустрован на слици 4.4. Чворови који чине максимални независни скуп обојени су црвеном бојом.



Слика 4.4. Резултат рада алгоритма такмичења

Очигледно је да време извршавања зависи од почетних података, односно ID-ева чворова, а начин на који су они додељени није прецизно дефинисан у самом алгоритму. Неки од примера додељивања ID-ева чворовима ће бити размотрени у даљем тексту.

4.1.3 Поређење рада алгоритма такмичења са радом других алгоритама

У овом одељку ћемо размотрити експерименталне резултате рада представљеног алгоритма такмичења и рада неких других алгоритама. Као први поредбени алгоритам ћемо узети познати Лубијев алгоритам (алгоритам 3), представљен у другом поглављу овог рада. За други поредбени алгоритам уводимо алгоритам који представља поједностављену верзију алгоритма такмичења: чвор постаје део максималног независног скупа ако има најмањи ID међу својим суседима, његови суседи се тада бришу и не разматрају даље.

Сва три алгоритма су испрограмирана у програмском језику C#. За потребе тестирања алгоритама генерисани су само неоријентисани графови. За сваки генерисани граф њему одговарајућа матрица суседства је приложена као засебна инстанца у виду текстуалног фајла. Инстанце, код и резултати тестирања представљају саставни део овог рада и налазе се на линку http://alas.matf.bg.ac.rs/~ml12008/. Чворови графа су имплементирани помоћу C# класе *Node*, док је за репрезентацију листе свих чворова графа коришћена C# класа *GraphNodes*. Након што су подаци учитани и након што је формирана листа суседа чворова, врши се идентификација чворова.

При креирању сваког чвора потребно је доделити му јединствени ID. То је омогућено формирањем листе природних бројева од 1 до n, где је n број чворова у графу. Сваки чвор на случајан начин добија један број из ове листе, након чега се тај број брише из листе, како би била обезбеђена јединственост ID-ева. Репрезентација тог броја у бинарном запису представља серијски број чвора односно његов јединствени ID. Поред ID-а, сваки чвор добија и листу себи суседних чворова. Тиме је процес иницијализације и формирања листе чворова за сам алгоритам завршен.

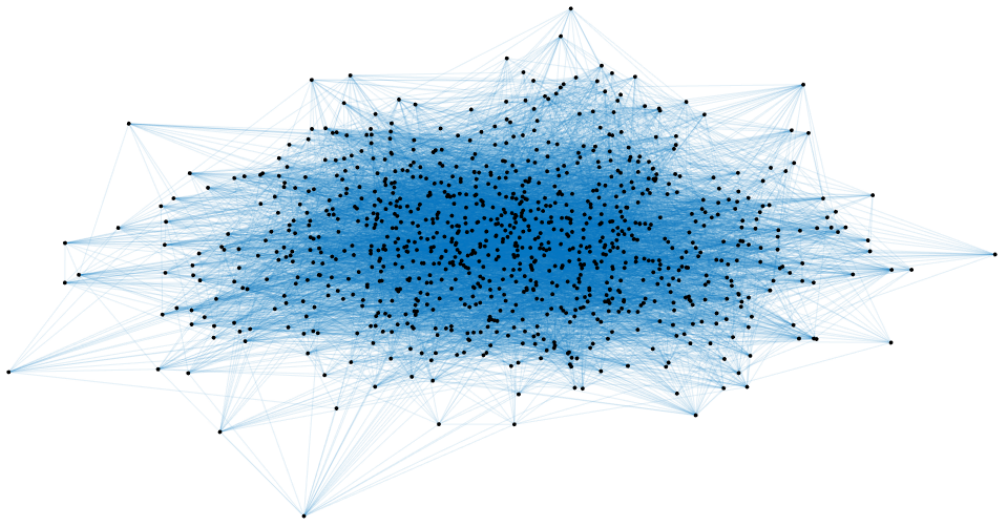
На основу претходне дискусије видимо да је за представљени алгоритам такмичења потребна једна итерација за креирање листе чворова и додељивање ID-ева. Исту ситуацију имамо и код друга два алгоритма: формирање и иницијализацију ID-ева код поједностављеног алгоритма и само формирање листе чворова код Лубијевог алгоритма.

За само такмичење сваког чвора у представљеном алгоритму такмичења потребно је извршити међусобну размену резултата претходног такмичења међу суседним чворовима, како би се срачунали резултати такмичења за наредну итерацију, промену стања самог чвора и промену стања његових суседа. Код поједностављеног алгоритма у једној итерацији имамо једну операцију мање, јер је у промени стања потребно само разменити ID-еве и на основу њих означити суседе чвора који улази у максимални независни скуп да више не учествују. Код Лубијевог алгоритма у току једне итерације је слична ситуација као код поједностављеног алгоритма, две операције су довољне - за размену случајних вредности суседних чворова и њихов улазак, односно испадање из максималног независног скупа.

У првој итерацији алгоритам такмичења и његова поједностављена верзија се понашају налик на Лубијев алгоритам, јер су вредности које карактеришу чворове изабране на случајан начин додавањем ID-ева. Након прве итерације алгоритам такмичења губи ову сличност. Главна разлика се огледа у чињеници да прва два алгоритма у сваком проласку дају бар један чвор који улази у максимални независни скуп. Код Лубијевог алгоритма, како у свакој итерацији чвор као своју вредност добија случајну вредност из интервала $[0, 1]$, могући су итеративни пролази који не дају ни један чвор за улазак у максимални независни скуп.

За поређење рада ових алгоритама као улазне параметре ћемо у првом случају користити насумичне графове за које је алгоритам коректан, али се не гарантује представљена временска сложеност, док ћемо у другом случају користити графове за које је овај алгоритам и осмишљен - графове ограниченог раста.

За конструкцију насумичних графова постоји пуно модела ([24], [25], [26]). Насумични графови које овде користимо су генерисани преко Ердос-Рени модела [18]. Ердос-Рени $\mathcal{G}(n, p)$ модел је један од најпознатијих модела за генерисање насумичних графова. На почетку се генерише n неповезаних чворова и пролази преко свих могућих грана, додајући их са вероватноћом p, $0 \leq p \leq 1$. Пример насумичног Ердос-Рени графа за $n = 1000$ и $p = 0.02$ је могуће видети на слици 4.5.

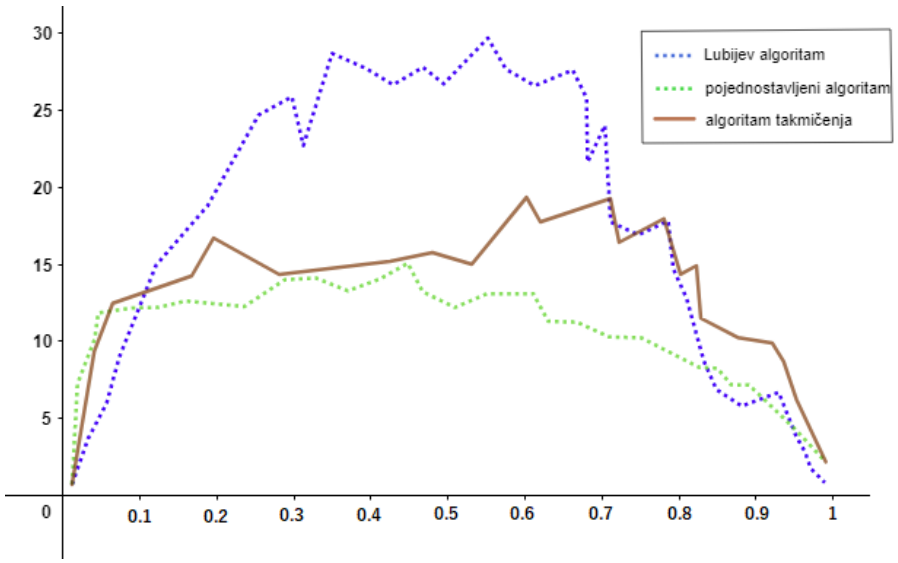


Слика 4.5. Пример насумичног Ердос-Рени графа са $n = 1000$ чворова и вероватноћом везивања $p = 0.02$

Насумични графови које користимо су конструисани у Matlab-у преко поменутог Ердос-Рени модела. Као повратна вредност се добија матрица суседства димензије $n \times n$

конструисаног графа. Та матрица представља улазни податак за тестирање представљених алгоритама. Број чворова за графове на којима вршимо тестирање је фиксиран на $n = 1000$, док вероватноћу p мењамо у интервалу $[0, 1]$ чиме добијамо различите насумичне графове са 1000 чворова. Вероватноћа p је рачуната случајним избором вредности из интервала $[0, 1]$. За потребе тестирања конструисано је 30 графова. Очигледно је да за $p = 0$ граф који добијамо нема ниједну грану, само изоловане чворове, као и да за $p = 1$ добијамо комплетан граф. Ово су тривијални случајеви где за $p = 0$ сви чворови графа чине максимални независни скуп, а за $p = 1$ само један, било који чвор графа, чини максимални независни скуп. Поред тога максимални независни скуп у оба случаја је и максимум независни скуп.

Размотримо сада експерименталне резултате добијене радом ова три алгоритма са насумично генерисаним Ердос Рени графовима са 1000 чворова. Као што је већ речено, граф се изгенерише помоћу Matlab-a и служи као улазни податак за сва три алгоритма. На следећем графику је приказано поређење броја потребних итерација алгоритама за проналазак максималног независног скупа, где је на x-оси представљена вероватноћа постојања гране између два чвора, док y-оса показује број итерација потребних да алгоритам дође до решења.



Слика 4.6. Поређење броја потребних итерација алгоритама за проналазак решења на Ердос Рени графовима од 1000 чворова.

На слици 4.6 испрекиданом плавом линијом је представљен број итерација потребних да Лубијев алгоритам дође до решења у односу на вероватноћу постојања гране p, испрекиданом зеленом број итерација потребних да поједностављени алгоритам дође до решења и пуном браон линијом број потребних итерација да алгоритам такмичења дође до решења у односу на променљиву p. График је добијен формирањем линеарног сплајна на основу 30 добијених вредности.

Прво што уочавамо на слици 4.6. је да Лубијевом алгоритму у већем делу тестирања био потребан већи број итерација у односу на друга два алгоритма, док је поједностављеном алгоритму у већем делу тестирања требало мање итерација да дође до решења у односу на друга два алгоритма. Ситуације у којима се поједностављени алгоритам понаша драстично лошије у односу на Лубијев и алгоритам такмичења је присутна код графова који имају дуге ланце међусобно суседних чворова са равномерним смањивањем ID-ева, односно низове међусобно суседних чворова који имају такве ID-еве који се равномерно смањују па само по један чвор може бити са најмањим резултатом. Код таквих графова поједностављени алгоритам додаје чвор по чвор решењу, повећавајући

тиме број итерација драстично, док се алгоритам тамичења и Лубијев алгоритам не сусрећу са већим успоравањем у оваквој ситуацији. Посматрајући алгоритам такмичења и његов поједностављени алгоритам, јасно је да поједностављени алгоритам у потпуности зависи од почетног додељивања ID-ева, док алгоритам такмичења потенцијално неповољно додељивање ID-ева на почетку коригује током свог рада.

Као што је већ речено, алгоритам такмичења није конструисан за графове који немају особину ограниченог раста. У случајевима графова без ове особине просечно време извршавања које се гарантује је $O(n)$, односно алгоритму ће требати највише $O(n)$ итерација да дође до решења. Имајући ову чињеницу на уму, можемо закључити да је алгоритам тамичења дао добре резултате у тестирању са насумичним графовима.

Друго поређење које изводимо је на основу величине добијеног решења, односно кардиналности добијеног максималног независног скупа. Као улазни податак коришћено је 30 генерисаних графова од 1000 чворова са различитим вероватноћама везивања. У Табели 1 су представљени резултати добијеног тестирања, где се у првој колони налази ознака графа. У другој колони су представљене вредности промељиве p, односно вероватноћа везивања конкретног графа. У трећој колони се налази број грана графа. Четврта, пета и шеста колона садрже кардиналности, односно број чворова добијених максималних независних скупова радом алгоритма такмичења, поједностављеног алгоритма и Лубијевог алгоритма, респективно. Напоменимо да је алгоритам извршаван више пута, а да су у Табели 1 дате најфреквентније вредности.

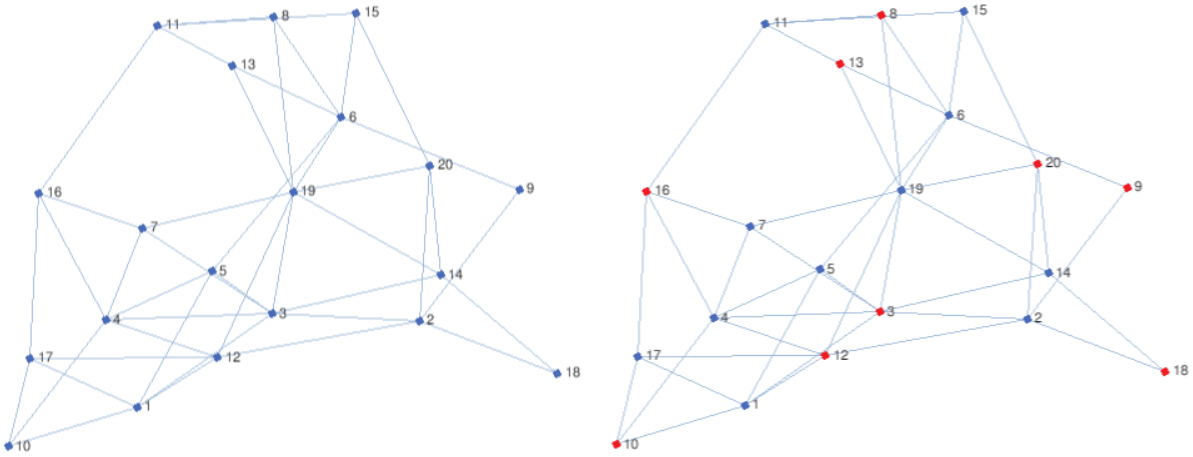
Табела 1. Кардиналности максималних независних скупова добијених радом представљених алгоритама са Ердос Рени графовима од 1000 чворова

	Вероватноћа постојања гране	Број грана	Алгоритам такмичења	Поједностављен алгоритам	Лубијев алгоритам
Граф1	0,03	14793	**123**	115	117
Граф2	0,05	29257	70	68	**71**
Граф3	0,08	43046	**52**	47	49
Граф4	0,11	56427	42	**43**	38
Граф5	0,14	69372	**38**	33	31
Граф6	0,16	82346	26	**31**	28
Граф7	0,19	94416	**27**	**27**	24
Граф8	0,21	106031	24	**25**	24
Граф9	0,22	106039	**23**	**23**	22
Граф10	0,25	129349	**22**	18	19
Граф11	0,28	140509	19	**20**	19
Граф12	0,30	151355	**16**	12	14
Граф13	0,31	151367	**15**	12	14
Граф14	0,35	171258	**15**	11	14
Граф15	0,36	171272	**16**	12	15
Граф16	0,39	181459	**15**	12	11
Граф17	0,41	190327	**12**	11	11
Граф18	0,44	208428	**12**	10	11
Граф19	0,45	208438	**13**	8	9
Граф20	0,48	217578	**10**	9	**10**
Граф21	0,51	233193	**10**	**10**	8
Граф22	0,52	233213	12	8	**13**
Граф23	0,55	248662	**10**	8	7
Граф24	0,56	248673	**9**	**9**	8
Граф25	0,59	256197	**11**	**11**	**11**
Граф26	0,64	263025	**9**	**9**	8
Граф27	0,66	269766	**9**	6	**9**
Граф28	0,69	276735	**9**	7	8
Граф29	0,71	276742	**8**	7	7
Граф30	0,75	289893	**8**	7	**8**

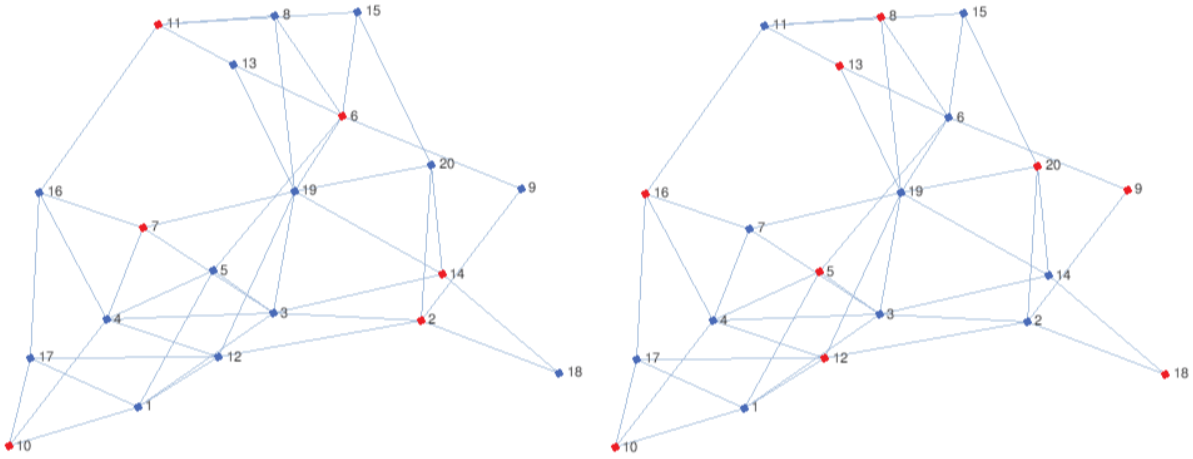
На основу резултата приказаних у Табели 1 можемо закључити да је алгоритам такмичења у већем делу тестирања налазио максималне независне скупове веће кардиналности од максималних независних скупова добијених радом друга два алгоритма

над истим графовима. За 15 графова алгоритам такмичења је као резултат добио скуп веће кардиналности од преостала два алгоритма, док је код 9 од 30 тестираних графова резултат био исте кардиналности као и резултат неког од преостала два алгоритма.

Погледајмо још добијене резултате рада сва три алгоритма на једном конкретном графу са мањим бројем чворова. Граф који посматрамо је произвољни Ердос Рени граф настао са вероватноћом везивања $p = 0.29$ и 20 чворова. Матрица суседства овог графа је дата у прилогу (тест граф).



Слика 4.7. Посматрани граф (слика лево) и максимални независни скуп добијен алгоритмом такмичења (слика десно)



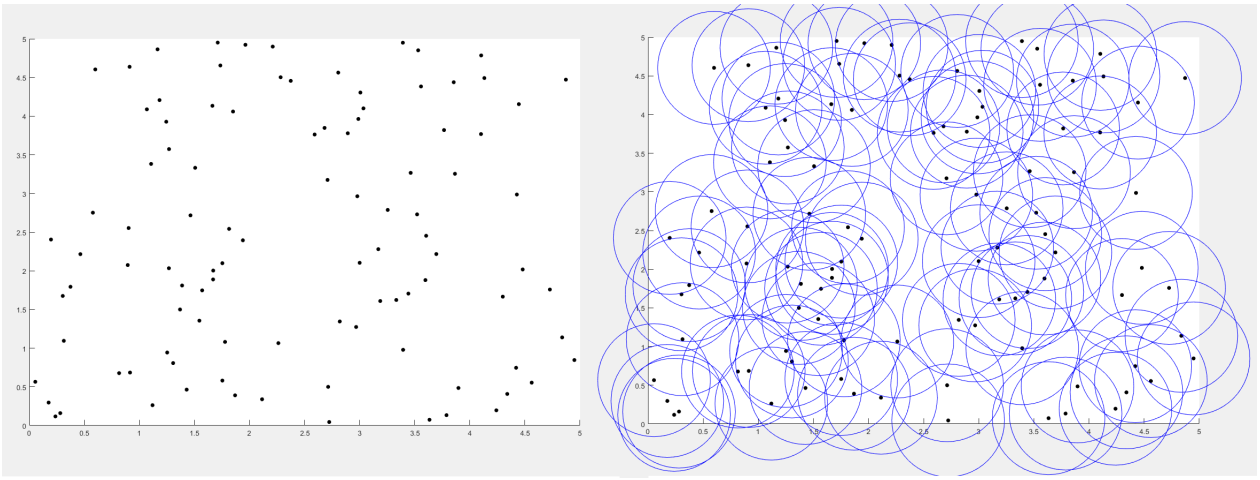
Слика 4.8. Максимални независни скуп добијен поједностављеним алгоритмом (слика лево) и Лубијевим алгоритмом (слика десно)

Као што видимо на сликама 4.7 и 4.8, сва три алгоритма су пронашла максималне независне скупове. Чворови који чине ове скупове су на сликама обојени црвеном бојом. У сва три резултата чворови који су у скупу су несуседни, што доказује независност и додавањем било ког чвора ова особина би била нарушена, што доказује максималност. Поједностављени алгоритам је добио максимални независни скуп који се састоји од 6 чворова, док су друга два алгоритма дошла до различитих максималних независних скупова, али оба кардиналности 9. Поред тога што је резултат максимални независни скуп, алгоритам такмичења и Лубијев алгоритам су у овом примеру дали и максимум независне скупове.

Размотримо сада рад алгоритама на графовима са особином ограничености раста, за које је алгоритам такмичења и конструисан. У трећем поглављу је показано да графови облика јединичног диска имају ову особину, додатно графови облика јединичног диска су полиномијално ограниченог раста. За алгоритам такмичења, да би се гарантовала показана брзина рада, потребна и довољна особина је да граф буде ограниченог раста. Полиномијална ограниченост не мора бити испуњена. Из тог разлога, као пример графова ограниченог раста за теситрање, узимамо графове облика јединичног диска.

За дати, произвољни граф без геометријске репрезентације, одређивање да ли се може представити као граф облика јединичног диска је НП комплетан проблем [3]. Из тог разлога, креирамо граф са геометријском репрезентацијом тако да обезбедимо да је граф облика јединичног диска и узимамо његову матрицу суседства као улазни податак. Такође, истакнути проблеми Max-IS и Min-DS остају НП комплетни проблеми чак и у случају графова облика јединичног диска [9]. Више о конструкцији оваквих графова је могуће видети у [30].

За потребе нашег тестирања графове облика јединичног диска конструишемо помоћу Matlab-а на следећи начин: у Еуклидовој равни $\mathbb{R}^2$ изаберемо низ случајно изабраних тачака са својим координатама (x, y). Те тачке ће представљати чворове генерисаног графа и за сваки нови граф који учествује у тестирању, тачке које представљају чворове тог графа ће изнова бити биране на случајан начин. Како се за сваки граф тачке бирају изнова на случајан начин, тако се и вредност променљиве r мења за сваки граф. За тестирање, као и у случају са Ердос Рени графовима, број чворова, а тиме и број произвољних тачака, ћемо фиксирати на $n = 1000$. Међутим, ради лакше и једноставније илустрације конструкције, објашњење ће бити дато на графовима са мањим бројем чворова.



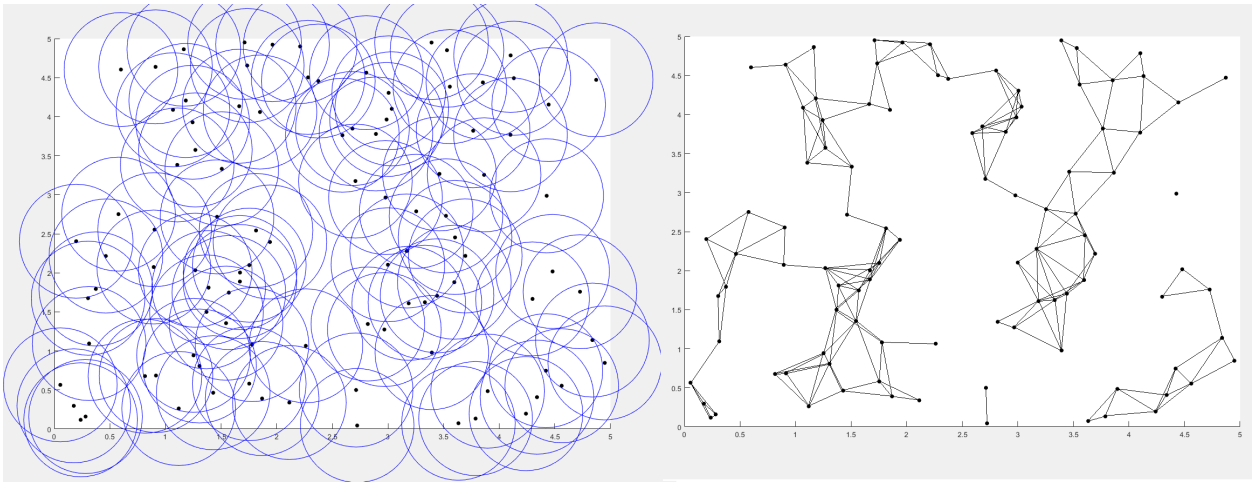
Слика 4.8. Пример 100 случајно изабраних тачака у Еуклидовој равни (слика лево) и кругови полупречника r креирани тако да те тачке буду центри кругова (слика десно)

На слици 4.8 видимо случајно изабране тачке и кругове око тих тачака са полупречником r. На основу дефиниције графова облика јединичног диска, између чворова постоји грана ако је њихово растојање највише 1. Међутим, да би добили већи број графова, са различитим вероватноћама постојања грана између њих, уместо услова да растојање буде мање или једнако 1, користићемо услов да између два чвора постоји грана уколико је Еуклидско растојање између њих мање или једнако r. Ово је могуће урадити зато што је овако добијен граф са геометријском репрезентацијом изоморфан графу облика јединичног диска. Због ове чињенице често се графовима облика јединичног диска називају сви графови састављени од колекције чворова (тачака) у

Еуклидској равни, где су два чвора суседна ако је њихово растојање мање од неког фиксираног прага (броја). При креирању графова облика јединичног диска за насумичан избор тачака се често користи Поисонов процес [20].

Након ове дискусије, видимо да ће грана између два чвора постојати ако се чворови (односно тачке које их представљају) налазе унутар круга описаног око оног другог, односно два чвора су суседна уколико се тачке које их представљају у равни налазе на растојању мањем од изабраног полупречника кругова. Након креирања насумичних тачака и кругова са центром у њима за изабрани полупречник, посматрамо све парове тачака. Ако једна тачка припада кругу описаном око неке друге тачке, односно растојање између њих је мање од одређеног полупречника, у матрици суседства за те чворове постављамо вредност на 1. Ако је растојање веће од задатог полупречника, матрица суседства на одговарајућим местима има нуле. На тај начин добијамо граф без геометријске репрезентације за који знамо да је граф облика јединичног диска, са особином ограничености раста.

На слици 4.9 је приказан граф облика јединичног диска настао случајним избором тачака у равни (слика 4.8.) где је претходна дискусија приказана графички, тј. може се видети да су чворови суседни само уколико се тачке које их представљају налазе унутар круга описаног око оне друге.



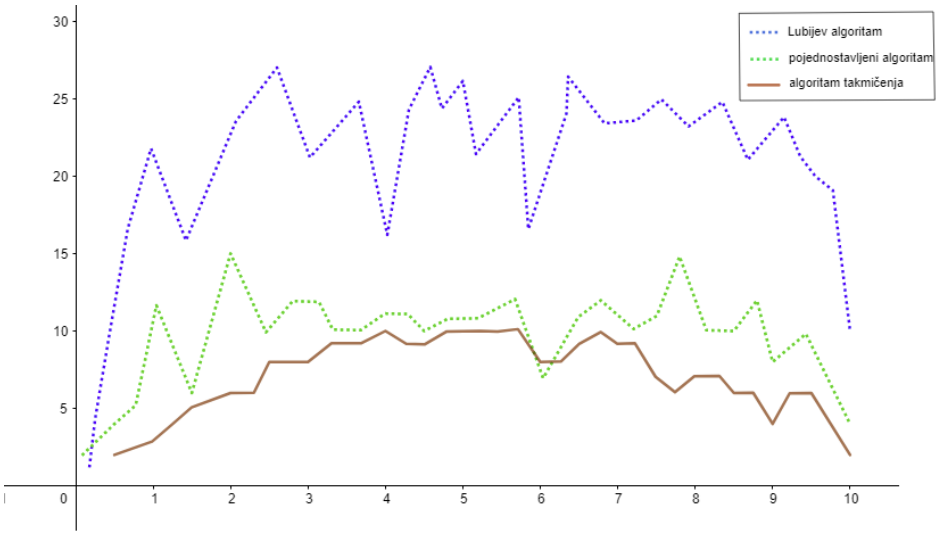
Слика 4.9. Пример графа облика јединичног диска датог са геометријском репрезентацијом (слика лево) и тог истог графа без геометријске репрезентације(слика десно)

Јасно је да повећавањем вредности промењљиве r повећавамо и површину у којој чворове који се налазе сматрамо суседним чворовима, односно повећавамо шансу за постојање гране између два чвора, док смањивањем вредности промењљиве r смањујемо и вероватноћу постојања гране између два чвора. За потребе овог тестирања, креирао је додатних 30 различитих графова за фиксирани број чворова $n = 1000$, променама промењљиве r.

Генерисане графове ограниченог раста користимо за друго тестирање описаних алгоритама. За разлику од првог тестирања код ког алгоритам такмичења није могао да гарантује брзину бољу од $O(n)$, сада гарантује проналазак решења за највише $O(\log^* n)$ итерација.

На слици 4.10 су представљени експериментални резултати тестирања рада изабрана три алгоритма. Потребан број итерација за проналазак максмималног независног скупакод генерисаних графова ограниченог раста је приказан на y-оси, док x-оса приказује промене полупречника r кругова унутар којих чворове сматрамо суседним. Испрекиданом плавом линијом је представљен број итерација потребних да Лубијев алгоритам дође до решења у односу на промењљиву r, испрекиданом зеленом број итерација

потребних да поједностављени алгоритам дође до решења и пуном браон линијом број потребних итерација да алгоритам такмичења дође до решења у односу на променљиву r. График је добијен формирањем линеарног сплајна на основу 30 добијених вредности.



Слика 4.10. Поређење броја итерација алгоритама на графовима облика јединичног диска од 1000 чворова.

Приликом тестирања рада изабрана три алгоритма са графовима ограниченог раста, Лубијевом алгоритму је био потребан највећи број итерација да дође до решења, док су алгоритам такмичења и поједностављени алгоритам са мањим бројем итерација долазили до максималног независног скупа и имали донекле слично понашање. Ипак, алгоритам такимчења је током скоро читавог теста захтевао најмањи број итерација за завршетак свог рада. Иако смо видели да алгоритам такмичења за сваки чвор захтева при једном пролазу једну операцију више у односу на друга два алгоритма, укупан број потребних итерација је скоро у сваком тренутку теста мањи у односу на потребан број за друга два алгоритма.

Поред броја потребних итерација, алгоритам такмичења је и константнији - захтевао је сличан број итерација током тестирања, са мањим осцилацијама у односу на друга два алгоритма, што је последица случајних избора који су у друга два алгоритма доминантни, а у алгоритму такмичења се радом алгоритма потенцијално лош случајни избор направљен на почетку алгоритма превазилази.

Друго поређење које изводимо је на основу кардиналности добијеног решења. Као улазни податак је узето 30 графова ограниченог раста од 1000 чворова. Матрице суседства ових графова су приложене као додатак раду у одговарајућим текстуалним датотекама, где име датотеке представља ознаку графа.

Табела 2 представља резултате добијеног тестирања, где се у првој колони налази ознака графа а у другој број грана графа. Трећа, четврта и пета колона садрже кардиналности, односно број чворова, добијених максималних независних скупова радом алгоритма такмичења, поједностављеног алгоритма и Лубијевог алгоритма респективно. Напоменимо да је алгоритам извршаван више пута, а да су у Табели 1 дате најфреквентније вредности.

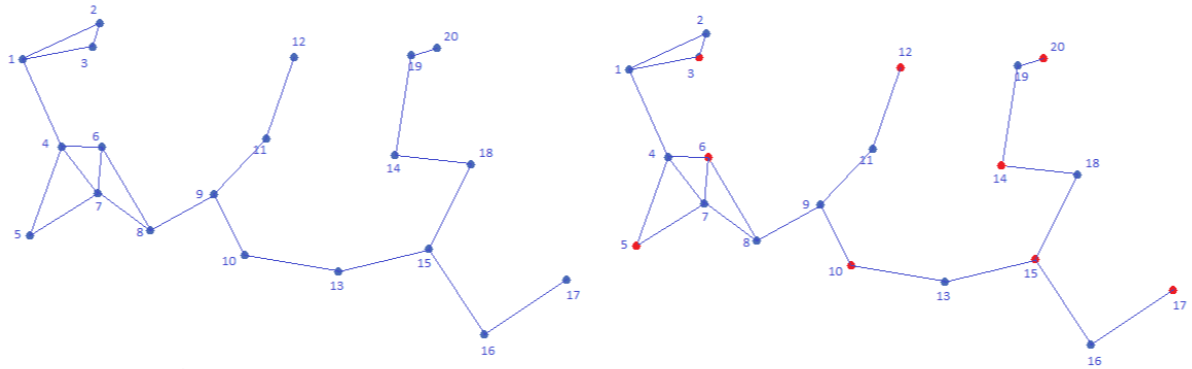
На основу резултата приказаних у Табели 2 можемо закључити да је алгоритам такмичења у већем делу тестирања налазио максималне независне скупове веће кардиналности од максималних независних скупова добијених радом друга два алгоритма над истим графовима.Код 22 од 30 тестираних инстанци алгоритам такмичења је као резултат добио скуп веће кардиналности од преостала два алгоритма, док је у 5 од 30

случајева резултат био исте кардиналности као и резултат неког од односу на друга два алгоритма. Скупове чворова који чине максималне независне скупове представљене у Табели 2 је могуће пронаћи у прилогу.

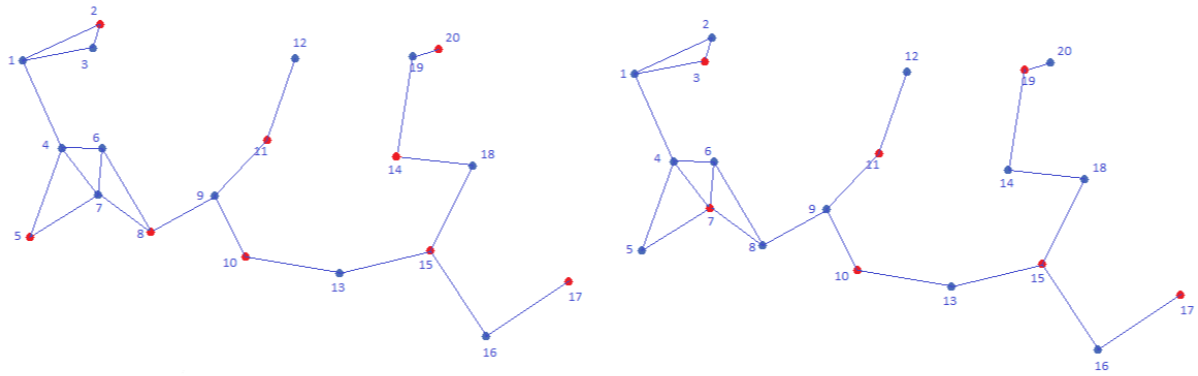
Табела 2. Кардиналности максималних независних скупова добијених радом представљених алгоритама са графовима облика јединичног диска од 1000 чворова

	Број грана	Алгоритам такмичења	Поједностављен алгоритам	Лубијев алгоритам
ГрафОР1	9834	159	**160**	156
ГрафОР2	19572	**98**	97	97
ГрафОР3	29257	**73**	64	66
ГрафОР4	29271	**70**	65	69
ГрафОР5	56427	**45**	42	43
ГрафОР6	65081	**38**	37	34
ГрафОР7	65087	**39**	38	**39**
ГрафОР8	82346	**32**	29	31
ГрафОР9	90346	**27**	26	26
ГрафОР10	90362	**28**	27	23
ГрафОР11	98431	**28**	25	21
ГрафОР12	122048	**20**	18	19
ГрафОР13	132882	**21**	20	18
ГрафОР14	143980	**19**	18	17
ГрафОР15	154344	**18**	16	17
ГрафОР16	164654	**17**	16	15
ГрафОР17	174290	**14**	13	13
ГрафОР18	174302	**14**	**14**	**14**
ГрафОР19	193772	**15**	14	12
ГрафОР20	202712	**14**	13	11
ГрафОР21	202723	**14**	12	13
ГрафОР22	211835	10	11	**13**
ГрафОР23	227711	**11**	9	10
ГрафОР24	236139	**9**	**9**	8
ГрафОР25	236152	**11**	8	9
ГрафОР26	243717	**11**	9	**11**
ГрафОР27	258159	**10**	8	9
ГрафОР28	258176	**9**	**9**	**9**
ГрафОР29	285971	**9**	8	7
ГрафОР30	285987	8	7	**9**

Погледајмо сада добијене резултате рада сва три алгоритма на једном конкретном графу ограниченог раста са мањим бројем чворова. Граф који посматрамо је граф облика јединичног диска са 20 чворова. Матрица суседства овог графа је дата у прилогу (grafTestUGG).



Слика 4.11. Посматрани граф облика јединичног диска (слика лево) и максимални независни скуп добијен алгоритмом такмичења (слика десно)



Слика 4.12. Максимални независни скуп добијен поједностављеним алгоритмом (слика лево) и Лубијевим алгоритмом (слика десно)

На сликама 4.11 и 4.12 видимо да су сва три алгоритма пронашла максималне независне скупове и у случају графова ограниченог раста. Чворови који чине максималне независне скупове су приказани црвеном бојом. У сва три резултата чворови који су у скупу су несуседни, што доказује независност и додавањем било ког чвора ова особина би била нарушена, што доказује максималност. Лубијев алгоритам је добио максимални независни скуп који се састоји од 7 чворова, док су друга два алгоритма дошла до различитих максималних независних скупова, али оба кардиналности 9. Иако три алгоритма која посматрамо гарантују само проналазак максималних независних скупова, алгоритам такмичења и поједностављени алгоритам су у овом случају пронашли и максималне независне скупове највеће кардиналности, односно максимум независне скупове.

Алгоритам такмичења и у случају произвољних графова и у случају графова ограниченог раста је захтевао мањи број итерација за проналазак решења од Лубијевог алгоритма (видети слике 4.6. и 4.10.). Алгоритам такмичења је захтевао већи број итерација од поједностављеног код произвољних графова, док је захтевао мањи број итерација у случају графова ограниченог раста. Ипак, предности рада алгоритма такмичења у односу на његов поједностављени алгоритам су већ истакнуте у тексту. Оно што је највећа предност алгоритма такмичења је гарантовање доста боље брзине проналаска решења у случајевима графова ограниченог раста. Поједностављени алгоритам може гарантовати само брзину $O(n)$. На основу Табеле 1 и Табеле 2 можемо извести закључак да је алгоритам такмичења и у случају произвољних графова и у случају графова ограниченог раста налазио максималне скупове веће кардиналности у односу на максималне скупове које су пронашла друга два алгоритма.

У [8] је могуће видети да је представљени алгоритам такмичења и асимптотски оптималан, односно у најгорем случају се понаша за константан фактор горе од најбољег могућег алгоритма. При томе тај константан фактор не зависи од величине графа. Доказ коректности и брзине представљеног алгоритма је могуће наћи у [8].

4.2 Повезани доминантни скуп

Повезани доминантни скуп у повезаном графу G је подскуп скупа G који је доминантан и чије гране формирају повезани подграф, односно за свака два чвора из доминатног скупа постоји пут од једног до другог само преко грана које припадају повезаном доминантном скупу. Проналазак минималног повезаног доминантног скупа такође представља НП комплетан проблем [1].

Уз помоћ максималног независног скупа добијеног представљеним алгоритмом такмичења је могуће формирати повезани доминантни скуп на следећи начин. Ако за

сваки чвор $v \in I$, где је I максимални независни скуп, изаберемо најкраће растојање до сваког чвора u, где је чвор u такав да важи $u \in \Gamma_3(v) \cap I$ и $ID_u < ID_v$, као резултат добијамо повезани доминантни скуп. ID-еви који су овде поменути су исти ID-еви додељени чворовима у алгоритму такмичења, а функција $p(.)$ функција раста. Како је кардиналност скупа $\Gamma_3(v) \cap I$ највише $p(3)$ и дужина било ког изабраног пута највише 3, повезани доминантни скуп ће имати највише $3 \cdot p(3) \cdot |I| + |I|$ чворова. Како смо у трећем поглављу показали да за графове ограниченог раста максимални независни скуп представља константну апроксимацију минимум доминантног скупа, константна апроксимација минимум повезаног доминантног скупа је могућа у времену $O(\log^* n)$. Више о овоме се може пронаћи у [8].

5. Апроксимативни алгоритми

Како смо већ навели, максимални независни скуп има широку примену и представља кључни део при решавању неких других проблема. У овом поглављу ће бити размотрени полиномијални апроксимативни алгоритми за креирање максимум независног скупа и минимум доминантног скупа који за фиксирано $\varepsilon > 0$ конструишу независни скуп, односно доминантни скуп, који не одступа од максимум независног скупа, односно минимум доминантног скупа, за више од $1 + \varepsilon$. На овај начин долазимо до апроксимативног решења НП комплетних проблема конструкције максимум независног скупа и минимум доминантног скупа. Време извршавања алгоритма за фиксирано ε мора бити полиномијално и једини услов који захтевамо је да граф буде полиномилално ограниченог раста. Сада, поред ограничености раста улазног графа, захтевамо и да функција раста буде полиномијална. Кроз оба алгоритма ће бити илустрована употреба алгоритма такмичења који је представљен у претходном поглављу.

Сложеност оба алгортима за граф полиномијално ограниченог раста са n чворова је $O(T_{MIS} + \frac{\log^* n}{\varepsilon^{O(1)}})$, где T_{MIS} представља време потребно за конструкцију маскималног независног скупа. Доказ сложености је могуће пронаћи у [5]. Како је $\varepsilon > 0$ фиксно, јасно је да битну улогу игра време потребно да се конструише максмимални независни скуп. Ово је разлог потребе за ефикасним и стабилним алгоритмом за проналажење максималног независног скупа и из тог разлога користимо алгоритам такмичења представљен у претходном поглављу.

Уведимо још скраћене ознаке које ћемо на даље користити. У околини $\Gamma_r(v)$ неког чвора $v \in V$ доминантан скуп $D(\Gamma_r(v))$ и независни скуп $I(\Gamma_r(v))$ обележимо са D_r и I_r респективно.

5.1 Максимум независни скуп

За апроксимацију максимум независног скупа као улазни податак имамо граф полиномијално ограниченог раста. Основна идеја је конструисати одређене помоћне структуре, пођи од произвољног чвора $v \in V$ и посматрати редом околине $\Gamma_r(v)$ за $r = 0$, 1, 2,... и у њима тражити оптимална решења $I_r \subseteq \Gamma_r$. Околину чвора повећавамо све док важи неједнакост:

$$|I_{r+1}| > (1 + \varepsilon) \cdot |I_r|.$$

Обележимо са $\bar{r}$ најмање r за које је ова неједнакост не важи. Битно запажање је да је повећавање околине ограничено, односно опсег највеће околине коју треба да разматрамо је ограничен константом која зависи само од функције раста и $\varepsilon > 0$, а не зависи од броја чворова самог графа. Покажимо то формално следећом лемом:

Лема 5.1. *Нека је дат граф $G = (V, E)$ $p-$полиномијално ограниченог раста. Тада постоји константа $c = c(\varepsilon)$ таква да је опсег r било које околине Γ_r коју разматрамо у алгоритму за конструкцију $1+\varepsilon$ апроксимације максимум независног скупа ограничен са c, односно $r \leq c$.*

Доказ. Нека је $r < \bar{r}$. Тада за r важи:

$$|I_r| > (1 + \varepsilon)|I_{r-1}| > \cdots > (1 + \varepsilon)^r |I_0|.$$

Будући да I_0 укључује само један изабрани чвор, следи да је $|I_0| = 1$, па долазимо до неједнакости: $|I_r| > (1+\varepsilon)^r$. Да би комплетирали доказ искористимо још чињеницу да је ово граф полиномијално ограниченог раста, односно да важи $|I_r| \leq p(r)$. Коначно:

$$(1+\varepsilon)^r < |I_r| \leq p(r).$$

Па очигледно: $(1+\varepsilon)^r < p(r)$. Овим смо доказ леме завршили. □

Из ове леме произилази и закључак да се за произвољни чвор v, чије околине посматрамо, оптимални независни скуп може наћи у времену $n^{O(p(c)^2)}$, где је n број чворова, а p полиномијална функција раста. Формулацију леме, доказ и закључак је могуће пронаћи у [6].

Погледајмо сада псеудо код апроксимативног алгоритма за креирање максимум независног скупа у графу:

Апроксимативни алгоритам - максимум независни скуп[5]

Улазни параметар: Неоријентисани граф $G = (V, E)$ полиномијално ограниченог раста, $\varepsilon > 0$, $\bar{c} := c(\varepsilon) + 2$
1. Наћи максимални независни скуп I у графу
2. Уз помоћ максималног независног скупа I формирати кластеровани граф $\bar{G}$ који се састоји од кластера $\Gamma_{2\bar{c}}(v)$, $v \in I$
3. Извршити бојење графа $\bar{G}$ са $\Delta_{\bar{G}} + 1$ боја
4. Иницијализовати $\bar{I} := \emptyset$
5. **For** $k = 1$ **to** $\Delta_{\bar{G}} + 1$ **do**:
6. **For each** чвор $v \in I$ боје k **do**:
7. **while** $\Gamma(v) \cap V \neq \emptyset$ **do**:
8. За неко $u \in \Gamma(v) \cap V$ тражити локално оптимални независни скуп све док повећавањем околине не дође до ситуације: $|\bar{I}_{\bar{r}+1}| \leq (1+\varepsilon)|\bar{I}_{\bar{r}}|$
9. $\bar{I} := \bar{I} \cup \bar{I}_{\bar{r}}(u)$, $V := V \setminus \Gamma_{\bar{r}+1}(u)$
10. **end while**
11. **end for**
12. Добијено $\bar{I}$ је $1+\varepsilon$ апроксимација максимум независног скупа I^*

Како овај алгоритам као предуслов има да улазни граф буде полиномијално ограниченог раста, могуће је искористити представљени алгоритам такмичења за рачунање максималног независног скупа у првом кораку. На тај начин брзо и сигурно добијамо максимални независни скуп потребан за овај апроксимативни алгоритам. Добијени максимални независни скуп своју примену налази већ у следећем кораку за конструкцију кластер графа. Кластерграф полупречника c дефинишемо као граф $\bar{G} = (\bar{V}, \bar{E})$ где за произвољне чворове $u, v \in \bar{V}$ грана $(u, v) \in \bar{E}$ ако и само ако је Еуклидско растојање између чворова u и v највише c. Више о формирању кластерграфа је могуће пронаћи у [5], где је показано и да је могуће ефикасно извршити $\Delta_{\bar{G}} + 1$ бојење, односно могуће је извршити бојење са највише $\Delta_{\bar{G}} + 1$ боја, где је $\Delta_{\bar{G}}$ највећи степен неког чвора у $\bar{G}$. Доказ за ефикасно бојење кластерграфа је могуће пронаћи и у [22].

Након креираних помоћних структура, алгоритам паралелно креира локалне независне скупове у околинама чворова обојених истом бојом. Они су означени са $\bar{I}_r$, за околину r неког чвора изабраног у кораку 8. На овај начин се формирају локално оптимална решења, тежећи ка глобалном решењу. Како је наглашено пре самог алгоритма,

локално оптимална решења је могуће наћи у времену $n^{O(p(c)^2)}$, па је локално оптимална решења могуће брзо наћи. Како алгоритам нема практичан значај, већ само теоријски, у овом тренутку се не наводе специфични агоритми за тражење локално оптималног решења, довољно је постојање таквих, а више о томе је могуће пронаћи у [5]. Сваки пут када неједнакост $|\bar{I}_{r+1}| > (1+\varepsilon)|\bar{I}_r|$ буде нарушена за избрани чвор из корака 8, из разматраног графа G склањамо околину разматраног чвора Γ_{r+1} а $\bar{I}_r$ додајемо у $\bar{I}$ како би било део глобалног решења. Нагласимо да када $\bar{I}_r$ улази у решење $\bar{I}$ како из преосталог скупа за разматрање избацујемо и све директне суседе од $\bar{I}_r$ следећи креирани независни скуп неће нарушавати независност у односу са претходним. Процес се наставља све док не искористимо све чворове из графа G. Независни скуп који се добије као резултат извршавања овог алгоритма је $1+\varepsilon$ апроксимација максимум независног скупа, односно за $\bar{I}$ које добијемо као резултат рада алгоритма важи да је независан у датом графу и да $|I^*| \leq (1+\varepsilon)|\bar{I}|$ где је I^* максимум независни скуп датог графа.

5.2 Минимум доминантни скуп

Аналогно алгоритму који проналази $1+\varepsilon$ апросксимацију максимум независног скупа, сада ће бити представљен алгоритам који у полиномијалном времену проналази $1+\varepsilon$ апросксимацију минимум доминатног скупа. Захтев да улазни граф буде полиномијално ограниченог раста је присутан и овде. Током рада алгоритма за конструкцију $1+\varepsilon$ апросксимације минимум доминатног скупа, конструишу се помоћне структуре и затим се траже локална решења.

За изабрани чвор v се посматрају редом околине $\Gamma_r(v)$ за $r = 0,\ 1,\ 2...$ и у њима траже оптимална решења, све док важи неједнакост:

$$|D_{r+2}| > (1+\varepsilon) \cdot |D_r|.$$

Поново, са $\bar{r}$ обележимо најмање r за које ова неједнакост не важи. Аналогно ограничености највеће околине коју посматрамо за претходни алгоритам и сада је највећа околина коју треба разматрати за потребе алгоритма ограничена константом која не зависи од величине самог графа, односно формално:

Лема 5.2. *Нека је дат граф $G = (V, E)$ $p-$полиномијално ограниченог раста. Тада постоји константа $c = c(\varepsilon)$ таква да је опсег r било које околине Γ_r коју разматрамо у алгоритму за конструкцију $1+\varepsilon$ апроксимације минимум доминантног скупа ограничен са c, односно $r \leq c$.*

Доказ. Нека је $r < \bar{r}$. Тада за r важи:

$$|D_r| > (1+\varepsilon)|D_{r-2}| > \cdots > (1+\varepsilon)^{\frac{r}{2}}|D_0|.$$

Како је $|D_0| = 1$ тада $|D_r| > (\sqrt{1+\varepsilon})^r$. Сваки максимум независни скуп је истовремено и максимални независни скуп, самим тим је и доминантан, па за скуп D_r који је доминантан у Γ_r важи неједнакост $|D_r| \leq |I_r|$, а како на основу ограничености раста $|I_r| \leq p(r)$ лако долазимо до неједнакости $p(r) > (\sqrt{1+\varepsilon})^r$ која комплетира доказ леме. □

Лему као и последицу да је за сваки чвор v у околинама $\Gamma_r(v)$ оптималне доминантне скупове могуће наћи у времену $n^{O(p(r)^2)}$ је могуће наћи у [6].

Када се неједнакост $|D_{r+2}| > (1+\varepsilon) \cdot |D_r|$ наруши за $\bar{r}$, глобалном решењу D додајемо скуп $D_{\bar{r}+2}$, док из преосталог скупа за разматрање избацујемо $\Gamma_{\bar{r}+2}$. За разлику од апроксимативног алгоритма за максимум независни скуп, сада се не посматрају све

околине, већ након посматрања околине $\Gamma_{\bar{r}}$, следећа околина за разматрање је $\Gamma_{\bar{r}+2}$. Разлог тога се огледа у 2-сепарацији (видети [6]) која омогућава да локално доминантни скупови не наруше особину доминације уласком у резултујући скуп. Локално решење $D_{\bar{r}+2}$ доминира скупом $\Gamma_{\bar{r}+2} \cap V$ који елиминишемо за даље разматрање, тако да глобално решење мора задржати особину доминације. За добијени скуп D који се добија као резултат рада алгоритма, поред особине доминације важи и тражена тачност $|D| \leq (1+\varepsilon)|D^*|$, где D^* представља минимум доминантни скуп у датом графу. Како је значај овог апроксимативног алгоритма теоријски, овде је представљено само да је локално оптимална решења могуће пронаћи брзо, а нису детаљно разматрани алгоритми којима би се то постизало, за више информација погледати [5].

Погледајмо сада псеудо код алгоритма за креирање $1+\varepsilon$ апроксимације минимум доминантног скупа у графу:

Апроксимативни алгоритам - минимум доминантни скуп[5]

Улазни параметар: Неоријентисани граф $G = (V, E)$ полиномијално ограниченог раста, $\varepsilon > 0$, $\bar{c} := c(\varepsilon) + 2$
1. Наћи максимални независни скуп $\bar{I}$ у графу
2. Уз помоћ максималног независног скупа $\bar{I}$ формирати кластеровани граф $\bar{G}$ који се састоји од кластера $\Gamma_{2\bar{c}}(v)$, $v \in I$
3. Извршити бојење графа $\bar{G}$ са $\Delta_{\bar{G}} + 1$ боја
4. Иницијализовати $D := \emptyset$
5. **For** $k = 1$ **to** $\Delta_{\bar{G}} + 1$ **do**:
6. **For each** чвор $v \in \bar{I}$ боје k **do**:
7. **If** $\Gamma(v) \cap V \neq \emptyset$ **then**
8. За неко $u \in \Gamma(v) \cap V$ тражити локално оптимални доминантни скуп све док повећавањем околине не дође до ситуације: $|D_{\bar{r}+2}| \leq (1+\varepsilon)|D_{\bar{r}}|$
9. $D := D \cup D_{\bar{r}+2}(u)$, $V := V \setminus \Gamma_{\bar{r}+2}(u)$
10. **end if**
11. **end for**
12. Добијено D је $1+\varepsilon$ апроксимација минимум доминантног скупа

За конструкцију максималног доминантог скупа у првом кораку можемо поново користити алгоритам тамичења, јер је испуњен услов ограничености раста графа. За конструкцију кластерованог графа и бојење важи исто разматрање као и у претходно представљеном алгоритму за конструкцију апроксимације максимум независног скупа.

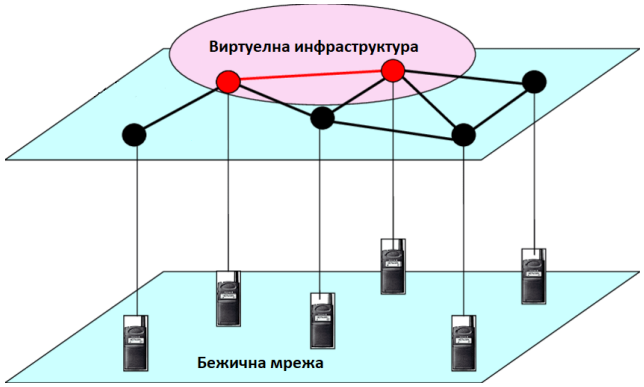
Апроксимативни алгоритми су представљени због свог теоријског значаја иако немају практичну примену. Опште инфомарције о њима су дате у овом поглављу, а више о њима, као и доказ коректности и доказ да оба алгоритма завршавају у полиномијалном времену, може се наћи у [5] и [19]. Тиме је завршено разматрање алгоритама који у полиномијалном времену конструишу апроксимације решења НП комплетних проблема конструкције минимум доминантног скупа и максимум независног скупа у графу полиномијално ограниченог раста, тачности $(1+\varepsilon)$ за унапред задато $\varepsilon > 0$. Такође, показано је да време рада ових алгоритама у великој мери зависи од времена рада изабраног алгоритма за конструкцију максималног независног скупа. Зато је битно изабрати брз и сигуран алгоритам за проналазак максмималног независног скупа, као што је алгоритам такмичења, представљен у претходном поглављу.

6. Конструкција модела простих бежичних мрежа - закључак

У овом раду су представљене особине независности и доминације у графу, као и повезаност ових особина са бежичним простим мрежама. Као што је већ речено, особине доминације и независности у графу имају битну улогу у конструкцији модела бежичних простих мрежа. Наглашено је да чворови унутар независног доминантног скупа не ометају једни друге у паралелном преношењу информација, док доминантан скуп може послужити да ефикасно допреми информације до сваког чвора у мрежи. Притом, максимум независни скуп задовољава оба својства.

Постоји много различитих перспектива из којих можемо посматрати доминантне и независне скупове и алгоритме за њихову конструкцију. Узимајући у обзир кардиналност, добијамо оптимизационе проблеме максимум независни скуп и минимум доминантни скуп чија конструкција представља НП комплетне проблеме. У раду су представљени алгоритми који у полиномијалном времену проналазе доминанти скуп и независни скуп који не одступају од минимум доминантног и максимум независног скупа за више од $1 + \varepsilon$.

Ако за виртуелну инфраструктуру простих бежичних мрежа као модел узмемо доминантни скуп, тада чворови који припадају доминантном скупу могу да даље прослеђују информације до свих осталих чворова у графу. Чворови који нису у доминантном скупу су сигурно суседни са бар једним чвором доминантног скупа, тако да ће сигурно моћи да приме информације.



Слика 6.1. Пример конструкције модела за реалну просту бежичну мрежу уз помоћ доминантног скупа

Као оптимизациони проблем намеће се потреба да што мање чворова преноси информације, али да и даље сви чворови могу да добију информацију. Идеја је користити минимум доминантне скупове као виртуелну инфраструктуру. Како је истакнуто проблеми Max-IS и Min-DS су НП-комплетни. У петом поглављу рада су представљени апроксимативни алгоритми за решавање ових проблема. Неопходан алгоритам за представљене апроксимативне алгоритме је алгоритам који проналази максимални независни скуп, па су у раду размотрена три алгоритма за овај подпроблем. Такође сложеност апроксимативних алгоритама зависи од алгоритма за проналазак максималног независног скупа, па је у раду испитана и ефикасност три разматрана алгоритма за решење поменутог подпроблема.

На слици 6.1. је могуће видети пример виртуелне инфраструктуре за бежичну мрежу. Оно што је мана оваквог модела је да за мрежу представљену преко повезаног графа, чворови доминантног скупа не морају бити повезани. Ако се ограничимо да чворови који нису у доминантном скупу могу да комуницирају само са својим суседима који јесу у доминантном скупу, може доћи до ситуације да једна информација не може доћи до свих чворова из мреже.

Ова потешкоћа се може превазићи тако што ће се користити повезани доминантни скупови, односно минимум повезани доминантни скупови. Тако информација може доћи до свих чворова који припадају доминантном скупу а и преко чворова из доминантног скупа до свих осталих чворова, без обзира из ког првог чвора да информација потекне.

Овим је дискусија о конструкцији простих мрежа уз помоћ доминантних и независних скупова у овом раду комплетирана.

ЛИТЕРАТУРА

[1] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-completeness, Freeman, 1979.

[2] O. Ore, Theory of graphs, American Mathematical Society, 1962.

[3] H. Breu and D. G. Kirkpatrick, Unit disk graph recognition is NP-hard., Computational Geometry 9, 3-24, 1998.

[4] F. Kuhn, T. Moscibroda, T. Nieberg, R. Wattenhofer, Fast deterministic distributed maximal independent set computation on growth-bounded graphs, Proceeding DISC'05 Proceedings of the 19th international conference on Distributed Computing, 273-287, 2005.

[5] F. Kuhn, T. Moscibroda, T. Nieberg, R. Wattenhofer, Local Approximation Schemes for Ad Hoc and Sensor Networks, In Proc. of the 3 rd ACM Joint Workshop on Foundations of Mobile Computing (DIALM-POMC), 2005.

[6] T. Nieberg, Independent and Dominating Sets in Wireless Communication Graphs, University of Twente, The Netherlands, 2006.

[7] J. L. Hurink, T. Nieberg, Approximating Minimum Independent Dominating Sets in Wireless Networks, Information Processing Letters 109, 155–160, 2008.

[8] J. Schneider, R. Wattenhofer, An Optimal Maximal Independent Set Algorithm for Bounded-Independence Graphs, Distributed Computing 22, (5–6): 363–379, 2010.

[9] B. N. Clark, C. J. Colburn, D. S. Johnsonr, Unit disks graphs, Discrete Mathematics 86, 165–177, 1990.

[10] C. Berge, Theory of Graphs and its Applications, Methuen, London, 1962.

[11] E. J. Cockayne, S. T. Hedetniemi, Towards a theory of domination in graphs, Networks 7, 247-261, 1977.

[12] Jack Edmonds, Minimum Partition of a Matroid Into Independent Subsets, Journal of research of the National Bureau of Standards-B. Mathematics and Mathematical Physics Vol. 69B, Nos. 1 and 2, 1965.

[13] X.Y. Li, Algorithmic, geometric and graphs issues in wireless networks, Wireless Communications and Mobile Computing 3, 119-140, 2003.

[14] P. Crescenzi, V. Kann, A compendium of NP optimization problems, http://www.nada.kth.se/ viggo/problemlist/

[15] G.E. Blelloch, J. T. Fineman, J. Shun, Greedy sequential maximal independent set and matching are parallel on average, Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures, 308-317, 2012.

[16] M. Luby, A Simple Parallel Algorithm for the Maximal Independent Set Problem, SIAM Journal on Computing, 15 (4), 1036-1053, 1986.

[17] F. Kuhn, T. Moscibroda, R. Wattenhofer, On the locality of bounded growth, In 24th ACM Symposium on the Principles of Distributed Computing (PODC), 60-68, 2005.

[18] P Erdős, A Rényi, On the evolution of random graphs, Publ. Math. Inst. Hung. Acad. Sci, 17-60, 1960.

[19] Y Sun, X Gu, J Qian, Construction of Virtual Backbone on Growth-Bounded Graph with Variable Transmission Range, WSEAS transactions on computers 7, 32-38, 2008.

[20] J.F.C Kingman, Poisson Processes, Oxford University Press, 1993.

[21] J. Bauermeister, M. Zimmerman, M. Johns, P. Glowacki, S. Stoddard, E. Volz, Innovative recruitment using online networks: Lessons learned from an online study of alcohol and other drug use utilizing a web-based, Respondent-Driven Sampling (webRDS) strategy, Journal of Studies on Alcohol and Drugs 73(5), 2012.

[22] R. Cole and U. Vishkin, Deterministic coin tossing with applications to optimal parallel list rankin, Information and Control, 70(1):32–53, 1986.

[23] B. Gfeller, E.Vicari, A faster distributed approximation scheme for the connected dominating set problem for growth-bounded graphs, In International Conference on Ad-Hoc Networks and Wireless, 59-73, 2007.

[24] W.Aiello, F. Chung, L. Lu, A random graph model for massive graphs, In STOC Vol. 2000, 1-10, 2000.

[25] W.Aiello, F. Chung, L. Lu, A random graph model for power law graphs, Experimental Mathematics 10.1, 53-66, 2001.

[26] M. E. Newman, Random graphs with clustering, Physical review letters 103.5, 2009.

[27] F. Kuhn, R. Wattenhofer, Y. Zhang, Geometric ad-hoc routing: Of theory and practice, In Proceedings of the twenty-second annual symposium on Principles of distributed computing 63-72, 2003.

[28] F. Kuhn, R. Wattenhofer, Y. Zhang, Ad hoc networks beyond unit disk graphs, Wireless Networks 14.5, 715-729, 2008.

[29] I. W. Trofimov, Graphs with polynomial growth, Mathematics of the USSR-Sbornik, 51(2), 405, 1985.

[30] F. Kuhn, R. Wattenhofer, T. Moscibroda, Unit disk graph approximation, In Proceedings of the 2004 joint workshop on Foundations of mobile computing, 17-23, 2004.

ПРИЛОГ

```
using System.Collections.Generic;
using System.Linq;

namespace MisAlgorithm.BusinessLayer
{
        public class Node
        {
                #region Properties

                public string Name { get; set; }

                public int ID { get; set; }

                public Dictionary<int, int> R { get; set; } = new Dictionary<int, int>();

                public NodeStatus Status { get; private set; }

                /// <summary>
                /// Delayed Status that is calculated to be applied.
                /// </summary>
                public NodeStatus? NextStatus { get; set; }

                public List<Node> Neighbors = new List<Node>();

                public int J { get; set; }

                public List<Node> CompeteNeighbors
                {
                        get
                        {
                        return Neighbors.Where(node => node.Status == NodeStatus.Competitor).ToList();
                        }
                }

                public int CurrentIterationValue
                {
                        get
                        {
                                return R[J];
                        }
                        set
                        {
                                R[J] = value;
                        }
                }

                public int PreviousIterationValue
                {
                        get
                        {
                                return R[J - 1];
                        }
                }

                #endregion Properties

                public Node(string name, int id)
                {
                        Name = name;
                        ID = id;
                }

                 public Node(int id)
                 {
                                ID = id;
                 }

        #region Methods

        public void ApplyNextStatus()
                {
                        if (NextStatus != null)
                        {
                                Status = NextStatus.Value;
                                NextStatus = null;
                        }
                }

                public static void ApplyNextStatuses(IEnumerable<Node> nodes)
                {
                        foreach (Node node in nodes)
                        {
                                node.ApplyNextStatus();
                        }
                }
                #endregion Methods
        }
}

namespace MisAlgorithm.BusinessLayer
{
        public enum NodeStatus
        {
                Competitor,
                Ruler,
                Ruled,
                Dominator,
                Dominated
        }
}


using System;
```

```
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text.RegularExpressions;

namespace MisAlgorithm.BusinessLayer
{
    class SimplifiedAlgorithm
    {
        public List<Node> ExecuteSimplified(string textFile)
        {
            List<Node> GraphNodes = IntializeNodesSimplified(textFile);
            List<Node> result = new List<Node>();

            while (GraphNodes.Count > 0)
            {
                foreach (Node node in GraphNodes)
                {
                    if (node.Neighbors.Count == 0)
                        node.NextStatus = NodeStatus.Dominator;
                    else
                    {
                        List<Node> list = node.Neighbors.Where(x => x.NextStatus == null).ToList();
                        if (list.Count > 0)
                        {
                            double smallestNeighborID = list.Select(x => x.ID).Min();
                            if (node.ID < smallestNeighborID)
                                node.NextStatus = NodeStatus.Dominator;
                        }
                        else
                            node.NextStatus = NodeStatus.Dominator;
                    }
                }

                List<Node> enteringResult = GraphNodes.Where(x => x.NextStatus == NodeStatus.Dominator)
.ToList();
                if (enteringResult.Count > 0)
                {
                    foreach (Node enteringNode in enteringResult)
                    {
                        result.Add(enteringNode);
                        GraphNodes.Remove(enteringNode);
                        foreach (Node notEnteringNeighbord in enteringNode.Neighbors)
                        {
                            GraphNodes.Remove(notEnteringNeighbord);
                            notEnteringNeighbord.NextStatus = NodeStatus.Dominated;
                        }
                    }
                }
            }
            return result.OrderBy(x => PadNumbers(x.Name)).ToList();
        }

        private List<Node> IntializeNodesSimplified(string textFile)
        {
            List<Node> list = new List<Node>();
            var rand = new Random();
            if (File.Exists(textFile))
            {
                string[] lines = File.ReadAllLines(textFile);
                List<int> listOfIds = Enumerable.Range(1, lines.Length).OrderBy(i => rand.Next()).ToList();
                for (int index = 1; index <= lines.Length; index++)
                    list.Add(new Node(string.Format("node{0}", index), listOfIds[index - 1]));

                int nodeNumber = 0;
                foreach (string line in lines)
                {
                    string[] indicators = line.Split(',');
                    for (int j = 0; j < indicators.Length; j++)
                        if (indicators[j] == "1")
                            list[nodeNumber].Neighbors.Add(list[j]);
                    nodeNumber++;
                }
            }
            return list;
        }

        private string PadNumbers(string input)
        {
            return Regex.Replace(input, "[0-9]+", match => match.Value.PadLeft(10, '0'));
        }
    }
}

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text.RegularExpressions;

namespace MisAlgorithm.BusinessLayer
{
    class LubyAlgorithm
    {
        public List<NodeForLuby> ExecuteLuby(string textFile)
        {
            List<NodeForLuby> GraphNodes = IntializeLubyNodes(textFile);
            List<NodeForLuby> result = new List<NodeForLuby>();

            var rand = new Random();
            while (GraphNodes.Count > 0)
            {
                foreach (NodeForLuby node in GraphNodes)
```

```
                    node.Probability = rand.NextDouble();

                foreach (NodeForLuby node in GraphNodes)
                {
                    if (node.Neighbors.Where(x => !x.Excluded).ToList().Count == 0)
                        node.EnteringResult = true;
                    else
                    {
                        double smallestNeighborProb = node.Neighbors.Where(x => !x.Excluded)
.Select(x => x.Probability).Min();
                        if (node.Probability < smallestNeighborProb)
                            node.EnteringResult = true;
                    }
                }

                List<NodeForLuby> enteringResult = GraphNodes.Where(x => x.EnteringResult == true).ToList();
                if (enteringResult.Count > 0)
                {
                    foreach (NodeForLuby enteringNode in enteringResult)
                    {
                        result.Add(enteringNode);
                        enteringNode.Excluded = true;
                        GraphNodes.Remove(enteringNode);
                        foreach (NodeForLuby notEnteringNeighbord in enteringNode.Neighbors)
                        {
                            GraphNodes.Remove(notEnteringNeighbord);
                            notEnteringNeighbord.Excluded = true;
                        }
                    }
                }
            }
            return result.OrderBy(x => PadNumbers(x.Name)).ToList();
        }
        private string PadNumbers(string input)
        {
            return Regex.Replace(input, "[0-9]+", match => match.Value.PadLeft(10, '0'));
        }

        private List<NodeForLuby> IntializeLubyNodes(string textFile)
        {
            List<NodeForLuby> list = new List<NodeForLuby>();
            if (File.Exists(textFile))
            {
                string[] lines = File.ReadAllLines(textFile);
                int n = lines.Length;
                for (int index = 1; index <= n; index++)
                    list.Add(new NodeForLuby(string.Format("node{0}", index)));

                int nodeNumber = 0;
                foreach (string line in lines)
                {
                    string[] indicators = line.Split(',');
                    for (int j = 0; j < indicators.Length; j++)
                        if (indicators[j] == "1")
                            list[nodeNumber].Neighbors.Add(list[j]);
                    nodeNumber++;
                }
            }
            return list;
        }

    }
    public class NodeForLuby
    {
        #region Properties

        public string Name { get; set; }

        public double Probability { get; set; }

        public bool EnteringResult { get; set; }

        public bool Excluded { get; set; }

        public List<NodeForLuby> Neighbors = new List<NodeForLuby>();

        #endregion Properties

        #region Constructor

        public NodeForLuby(string name)
        {
            Name = name;
        }

        #endregion Constructor
    }
}

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text.RegularExpressions;

namespace MisAlgorithm.BusinessLayer
{
    class LubyAlgorithm
    {
        public List<NodeForLuby> ExecuteLuby(string textFile)
        {
            List<NodeForLuby> GraphNodes = IntializeLubyNodes(textFile);
            List<NodeForLuby> result = new List<NodeForLuby>();
```

```
            var rand = new Random();
            while (GraphNodes.Count > 0)
            {
                foreach (NodeForLuby node in GraphNodes)
                    node.Probability = rand.NextDouble();

                foreach (NodeForLuby node in GraphNodes)
                {
                    if (node.Neighbors.Where(x => !x.Excluded).ToList().Count == 0)
                        node.EnteringResult = true;
                    else
                    {
                        double smallestNeighborProb = node.Neighbors.Where(x => !x.Excluded)
.Select(x => x.Probability).Min();
                        if (node.Probability < smallestNeighborProb)
                            node.EnteringResult = true;
                    }
                }

                List<NodeForLuby> enteringResult = GraphNodes.Where(x => x.EnteringResult == true).ToList();
                if (enteringResult.Count > 0)
                {
                    foreach (NodeForLuby enteringNode in enteringResult)
                    {
                        result.Add(enteringNode);
                        enteringNode.Excluded = true;
                        GraphNodes.Remove(enteringNode);
                        foreach (NodeForLuby notEnteringNeighbord in enteringNode.Neighbors)
                        {
                            GraphNodes.Remove(notEnteringNeighbord);
                            notEnteringNeighbord.Excluded = true;
                        }
                    }
                }
            }
            return result.OrderBy(x => PadNumbers(x.Name)).ToList();
        }
        private string PadNumbers(string input)
        {
            return Regex.Replace(input, "[0-9]+", match => match.Value.PadLeft(10, '0'));
        }

        private List<NodeForLuby> IntializeLubyNodes(string textFile)
        {
            List<NodeForLuby> list = new List<NodeForLuby>();
            if (File.Exists(textFile))
            {
                string[] lines = File.ReadAllLines(textFile);
                int n = lines.Length;
                for (int index = 1; index <= n; index++)
                    list.Add(new NodeForLuby(string.Format("node{0}", index)));

                int nodeNumber = 0;
                foreach (string line in lines)
                {
                    string[] indicators = line.Split(',');
                    for (int j = 0; j < indicators.Length; j++)
                        if (indicators[j] == "1")
                            list[nodeNumber].Neighbors.Add(list[j]);
                    nodeNumber++;
                }
            }
            return list;
        }

    }
    public class NodeForLuby
    {
        #region Properties

        public string Name { get; set; }

        public double Probability { get; set; }

        public bool EnteringResult { get; set; }

        public bool Excluded { get; set; }

        public List<NodeForLuby> Neighbors = new List<NodeForLuby>();

        #endregion Properties

        #region Constructor

        public NodeForLuby(string name)
        {
            Name = name;
        }

        #endregion Constructor
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MisAlgorithm.BusinessLayer
{
        public class MisAlgorithmCore
        {
```

```
#region Properties

public List<Node> GraphNodes { get; set; }

#endregion Properties

#region Events

public EventHandler OnGlobalParallelStepEnded;

#endregion Events

#region Methods

public void Execute()
{
        // Initialize
        do
        {
                bool thereArePendingCompetitions = AreTherePendingCompetitions();

                // Stage start
                foreach (Node node in GraphNodes)
                {
                        // Wait for other competitions to end.
                        if (thereArePendingCompetitions)
                                break;

                        if (IsNodeFinalized(node))
                                continue;

                        // If there are more Phases, don't start new Stage for node.
                        if (HasPendingPhase(node))
                                continue;

                        node.NextStatus = NodeStatus.Competitor;
                }

                Node.ApplyNextStatuses(GraphNodes);

                // Phase start
                foreach (Node node in GraphNodes)
                {
                        // Wait for other competitions to end.
                        if (thereArePendingCompetitions)
                                break;

                        if (IsNodeFinalized(node))
                                continue;

                        // Skip nodes that are 'quite'
                        if (node.Status == NodeStatus.Ruled)
                                continue;

                        node.R.Clear();
                        node.R[0] = node.ID;

                        if (node.Status == NodeStatus.Ruler)
                                node.NextStatus = NodeStatus.Competitor;

                        node.J = 0;
                }

                Node.ApplyNextStatuses(GraphNodes);

                // Competition iteration
                foreach (Node node in GraphNodes)
                {
                        if (IsNodeFinalized(node))
                                continue;

                        node.J++;
                }
                // All nodes that are for Competition have the same iteration now.

                // Competition - Update Rj
                foreach (Node node in GraphNodes)
                {
                        if (IsNodeFinalized(node))
                                continue;

                        if (node.Status == NodeStatus.Competitor)
                        {
                                if (node.Neighbors.All(neighborNode =>
                                IsNodeFinalized(neighborNode)))
                                {
                                        node.NextStatus = NodeStatus.Dominator;
                                        node.CurrentIterationValue = 0;
                                        continue;
                                }

                                var competeNeighbors = node.CompeteNeighbors;
        if (node.CompeteNeighbors.Count > 0)
        {
            int minRjPrevious = competeNeighbors.Min(neighborNode =>
                                neighborNode.PreviousIterationValue);
            Node nodeToCompete = competeNeighbors.FirstOrDefault(neighborNode =>
                                neighborNode.PreviousIterationValue == minRjPrevious);

            CompeteWith(node, nodeToCompete);
        }

                        }
```

```
                    }

                    // Update Status based on Rj
                    foreach (Node node in GraphNodes)
                    {
                            if (node.Status != NodeStatus.Competitor)
                                    continue;

                            List<Node> nodeCompeteNeighbors = node.CompeteNeighbors;
                            if (nodeCompeteNeighbors.Count > 0)
                            {
                                    if (nodeCompeteNeighbors.All(neighborNode =>
                                        neighborNode.
                                      CurrentIterationValue > node.CurrentIterationValue))
                                      node.NextStatus = NodeStatus.Dominator;
                                    else if (nodeCompeteNeighbors.All(neighborNode =>
                                      neighborNode.
                                      CurrentIterationValue >= node.CurrentIterationValue))
                                      node.NextStatus = NodeStatus.Ruler;
                            }
        else
            node.NextStatus = NodeStatus.Dominator;
    }

                    Node.ApplyNextStatuses(GraphNodes);

                    // Update Status based on neighbors' Statuses
                    foreach (Node node in GraphNodes)
                    {
                            if (IsNodeFinalized(node))
                                    continue;

                            if (node.Neighbors.Any(neighborNode =>
                                    neighborNode.Status == NodeStatus.Dominator))
                                    node.NextStatus = NodeStatus.Dominated;
                            else if (node.Status != NodeStatus.Ruler && node.Neighbors.
                              Any(neighborNode => neighborNode.Status == NodeStatus.Ruler))
                                    node.NextStatus = NodeStatus.Ruled;
                    }

                    Node.ApplyNextStatuses(GraphNodes);

                    OnGlobalParallelStepEnded?.Invoke(this, EventArgs.Empty);

            }
            while (GraphNodes.Any(node => node.Status != NodeStatus.Dominated &&
                      node.Status != NodeStatus.Dominator));
    }

    #region Synchronization Methods

    private bool AreTherePendingCompetitions()
    {
            bool areTherePendingCompetitions = GraphNodes.Any(anyGraphNode =>
                    anyGraphNode.Status == NodeStatus.Competitor && anyGraphNode.J > 0);
            return areTherePendingCompetitions;
    }

    private static bool HasPendingPhase(Node node)
    {
            return node.Neighbors.Union(new Node[] { node }).Any(anyNode =>
                                                    anyNode.Status == NodeStatus.Ruler);
    }

    private static bool IsNodeWaitingNextPhase(Node node)
    {
            return node.Status == NodeStatus.Ruler || node.Status == NodeStatus.Ruled;
    }

    /// <summary>
    /// Node that has these two statuses is finished and excluded from alghorithm.
    /// </summary>
    /// <param name="node"></param>
    /// <returns></returns>
    private static bool IsNodeFinalized(Node node)
    {
            return node.Status == NodeStatus.Dominated || node.Status == NodeStatus.Dominator;
    }

    #endregion Synchronization Methods

    #region Calculation Methods

    internal static void CompeteWith(Node currentNode, Node nodeToCompete)
    {
            int currentNodeValue = currentNode.PreviousIterationValue;
            int competeNodeValue = nodeToCompete.PreviousIterationValue;

            bool hasGreaterValue = currentNodeValue > competeNodeValue;

            int currentNodeResultValue = !hasGreaterValue ? 0 :
                            GetMaxDifferBitPosition(currentNodeValue, competeNodeValue);
            currentNode.CurrentIterationValue = currentNodeResultValue;
    }

    internal static int GetMaxDifferBitPosition(int currentNodeValue, int competeNodeValue)
    {
            // Int32, position of 1 just before the last which is for sign.
            int compareBitValue = 1 << 30;

            while (compareBitValue > 0)
            {
                    int currentNodeBit = compareBitValue & currentNodeValue;
```

```
                                int competeNodeBit = compareBitValue & competeNodeValue;
                                if (currentNodeBit > competeNodeBit)
                                        break;

                                compareBitValue = compareBitValue >> 1;
                        }

                        int resultBitValueComparePosition = 0;
                        while (compareBitValue > 0)
                        {
                                resultBitValueComparePosition++;
                                compareBitValue = compareBitValue >> 1;
                        }

                        return resultBitValueComparePosition;
                }

                #endregion Calculation Methods

                #region Diagnostic Methods

                public string NodesSummary()
                {
                        StringBuilder sb = new StringBuilder();

                        foreach (Node node in GraphNodes)
                        {
                                sb.AppendLine(node.ToString());
                        }

                        return sb.ToString();
                }

                #endregion Diagnostic Methods

                #endregion Methods
        }
}

using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Windows.Forms;
using MisAlgorithm.BusinessLayer;
using System.IO;

namespace MisAlgorithm
{
        public partial class Form1 : Form
        {
        static readonly string textFile = @"C:\Temp\Data\Random\graf1";
        public Form1()
                {
                        InitializeComponent();
                        Initialize();
                }

                private void Initialize()
                {

                }

                private void btnCompetition_Click(object sender, EventArgs args)
                {
            MisAlgorithmCore algorithm = new MisAlgorithmCore();

            algorithm.GraphNodes = IntializeNodes();

                        algorithm.Execute();
            List<Node> s = algorithm.GraphNodes.Where(x => x.Status == NodeStatus.Dominator).ToList();
            richTextBox1.AppendText("Maksimalni nezavisni skup cine cvorovi:");
            foreach (Node n in s)
                richTextBox1.AppendText(n.Name + "  ");
            richTextBox1.AppendText("Kardinalnost dobijenog resenja je: " +algorithm.GraphNodes.
                                    Where(x=> x.Status== NodeStatus.Dominator).Count().ToString());
            richTextBox1.AppendText("***************\r\n");
        }

        private void btnLuby_Click(object sender, EventArgs args)
        {
            LubyAlgorithm algorithm = new LubyAlgorithm();
            List<NodeForLuby> nodes = algorithm.ExecuteLuby(textFile);

            richTextBox1.AppendText("Maksimalni nezavisni skup cine cvorovi:");
            foreach (NodeForLuby n in nodes)
                richTextBox1.AppendText(n.Name + "  ");
            richTextBox1.AppendText("Kardinalnost dobijenog resenja je: " + nodes.Count().ToString());
            richTextBox1.AppendText("***************\r\n");
        }
        private void btnSimplified_Click(object sender, EventArgs args)
        {
            SimplifiedAlgorithm algorithm = new SimplifiedAlgorithm();
            List<Node> nodes = algorithm.ExecuteSimplified(textFile);

            richTextBox1.AppendText("Maksimalni nezavisni skup cine cvorovi:");
            foreach (Node n in nodes)
                richTextBox1.AppendText(n.Name + "  ");
            richTextBox1.AppendText("Kardinalnost dobijenog resenja je: " + nodes.Count().ToString());
            richTextBox1.AppendText("***************\r\n");
        }

        private List<Node> IntializeNodes()
```

```
        {
            List<Node> list = new List<Node>();
            if (File.Exists(textFile))
            {
                string[] lines = File.ReadAllLines(textFile);
                int n = lines.Length;
                var rand = new Random();
                List<int> listOfIds = Enumerable.Range(1, n).OrderBy(i=> rand.Next()).ToList();
                for(int index= 1; index<=n; index++)
                    list.Add(new Node(string.Format("node{0}", index), listOfIds[index - 1]));

                int nodeNumber = 0;
                foreach (string line in lines)
                {
                    string[] indicators = line.Split(',');
                    for (int j = 0; j < indicators.Length-1; j++)
                        if (indicators[j] == "1")
                            list[nodeNumber].Neighbors.Add(list[j]);
                    nodeNumber++;
                }
            }
            return list;
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }
    }
}
```