

П.І. Стецюк^{1,3}, О.О. Слабоспіцька^{2,3}, О.О. Ушакова³

¹Інститут кібернетики імені В.М. Глушкова НАН України, Київ

²Інститут програмних систем НАН України, Київ

³Київський національний університет імені Тараса Шевченка

МАКСИМАЛЬНІ НЕЗАЛЕЖНІ МНОЖИНІ ВЕРШИН ГРАФА ТА ЇХ ЗАСТОСУВАННЯ В КЕРУВАННІ ПРОЕКТАМИ

Досліджено задачі булевого лінійного програмування для знаходження максимальної незалежної множини та максимальної суми незалежних множин вершин графа. Описано алгоритм Візинга-Плесневича для мінімального розфарбування вершин графа та його застосування для двох задач керування програмними проектами. Наведено AMPL-коди задач булевого лінійного програмування та результати обчислювальних експериментів для програм CPLEX і gurobi з оптимізаційного веб-сервісу NEOS-solver.

Исследованы задачи булевого линейного программирования для нахождения максимального независимого множества и максимальной суммы независимых множеств вершин графа. Описаны алгоритм Визинга-Плесневича для минимальной раскраски вершин графа и его применение для двух задач управления программными проектами. Приведены AMPL-коды задач булевого линейного программирования и результаты вычислительных экспериментов для программ CPLEX и gurobi из оптимизационного веб-сервиса NEOS-solver.

Boolean linear programming problems for finding maximum independent set and maximum sum of independent sets of graph vertices are considered. The Vizing-Plesnevich algorithm for minimal coloring of the vertices of a graph and its application for two problems of software project management are described. AMPL-codes of Boolean linear programming problems and results of experiments for CPLEX and gurobi programs from NEOS optimization Web-service are presented.

Ключові слова: незалежна множина вершин графа, мінімальне розфарбування графа, алгоритм Візинга-Плесневича, задача лінійного програмування, булева змінна, програмний проект, розподіл ресурсів.

Вступ. Множину вершин графа називають незалежною, якщо ніякі дві вершини цієї множини не з'єднані ребром (не є суміжні). Індукований цією множиною підграф складається з ізольованих вершин, їх кількість визначає розмір незалежної множини вершин графа. Оптимізаційну задачу

про незалежну множину формулюють таким чином: у заданому неорієнтовному графі G без петель потрібно знайти незалежну множину максимального розміру. Цей розмір називають числом незалежності або числом внутрішньої стійкості і позначають $\alpha(G)$.

Задача знаходження числа незалежності $\alpha(G)$ належить до класу NP-повних задач [1; 2]. Вона тісно пов'язана з задачею знаходження хроматичного числа $\chi(G)$ – найменшої кількості різних кольорів, якими можна розфарбувати вершини графа G таким чином, щоб ніякі дві суміжні вершини не були розфарбовані однаково. В.Г. Візинг та Г.С. Плесневич встановили [3], що задачу знаходження хроматичного числа можно звести до ітеративної послідовності задач знаходження числа незалежності графів, отриманих як декартів добуток повних графів та графа G .

У статті розглянуто задачі булевого лінійного програмування (булеві задачі) для знаходження незалежної множини максимального розміру (розділ 1) та заданої кількості незалежних множин, які не перетинаються і мають максимальний розмір суми незалежних множин (розділ 2). На їх основі у розділі 2 описано алгоритм Візинга-Плесневича для розфарбування вершин графа мінімальною кількістю кольорів. У розділі 3 розглянуто задачі оптимізації розкладу автономного тестування компонентів повторного використання у складі критичної програмної системи та формування ядер незалежних команд у критичному програмному проекті. Для розв'язання булевих задач застосовано програми **CPLEX** і **gurobi** з NEOS-сервера [4] та мову моделювання AMPL [5].

1. Булева задача для $\alpha(G)$ та її властивості. Нехай $x_i \in \{0,1\}$ – булева змінна, яка дорівнює одиниці, якщо вершина $i \in V(G)$ включається в незалежну множину вершин графа $G = (V(G), E(G))$, і нулю – у протилежному випадку. Щоб знайти число незалежності $\alpha(G)$, достатньо знайти один із розв'язків задачі булевого лінійного програмування [6]:

$$\alpha(G) = \max_{x_i \in \{0,1\}} \sum_{i \in V(G)} x_i \quad (1)$$

за обмежень

$$x_i + x_j \leq 1, \quad (i, j) \in E(G). \quad (2)$$

Задача (1), (2) містить $|V(G)|$ булевих змінних та $|E(G)|$ обмежень у формі лінійних нерівностей. Обмеження (2) означають, що коли у графі G вершини i та j з'єднані ребром, то тільки одна з них буде належати незалежній множині вершин графа G .

Число незалежності $\alpha(G)$ визначається однозначно, але йому може відповідати багато максимальних незалежних множин у графі G . Так, наприклад, граф C_5 (цикл із п'яти вершин, рис. 1, a) містить п'ять незалежних множин розміру $\alpha(C_5)=2$. На рис. 1 (б, в, г, д, е) наведено ці множини: $S_1 = \{1, 3\}$, $S_2 = \{1, 4\}$, $S_3 = \{2, 4\}$, $S_4 = \{2, 5\}$, $S_5 = \{3, 5\}$. Вершини, які входять в них, позначені чорним кольором.

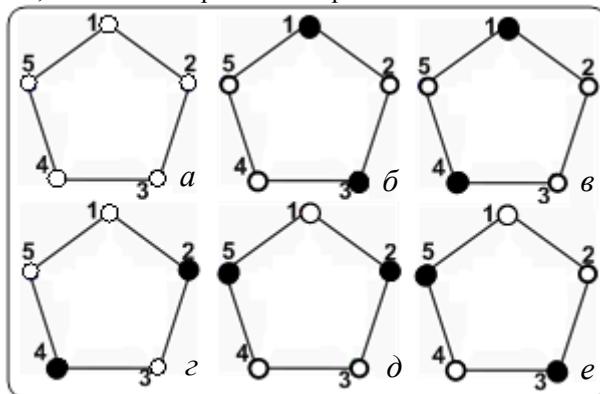


Рис. 1. П'ять незалежних множин максимального розміру у графі C_5

Щоб за допомогою задачі (1), (2) знайти якомога більшу кількість незалежних множин розміру $\alpha(G)$, можна скористатися випадковими збуреннями для булевих змінних x_i . Для цього цільову функцію (1) слід замінити на цільову функцію

$$\alpha_{\varepsilon}(G) = \max_{x_i \in \{0,1\}} \sum_{i \in V(G)} (x_i + \varepsilon_i x_i), \quad (1a)$$

де випадкові величини $|\varepsilon_i| \ll 1$ для всіх $i \in V(G)$. Залежно від вибраних значень для збурень ε_i , $i = 1, \dots, |V(G)|$, як результат розв'язання задачі (1a), (2) отримаємо ту чи іншу незалежну множину розміру $\alpha(G)$. Цей спосіб потребує значної кількості розв'язань задачі (1a), (2), що підтверджують результати обчислювального експерименту для знаходження незалежних множин S_1, \dots, S_5 у графі C_5 за допомогою програми **gurobi**. Результати експерименту наведені у табл. 1: правила генерації випадкових збурень (колонка 1) та частота знаходження кожної з п'яти незалежних множин з рис. 1 (колонки 3–7) для різних кількостей запусків програми **gurobi**: $nCall = 10$, $nCall = 100$ і $nCall = 1000$ (колонка 2). Для генерації

випадкових збурень використовувалися датчики **Uniform** та **Normal**, що забезпечують рівномірний та нормальній розподіли випадкових величин.

Таблиця 1

Частота знаходження незалежних множин S_1, \dots, S_5 для графа C_5
за рівномірного (Uniform) та нормального (Normal)
розділів випадкових збурень ε_i , $i = 1, \dots, 5$

$\varepsilon_i, i = 1, \dots, 5$	$nCall$	$n(S_1)$	$n(S_2)$	$n(S_3)$	$n(S_4)$	$n(S_5)$
0.001 \times Uniform(-1,1)	10	3	4	0	1	2
	100	16	24	28	9	23
	1000	205	214	177	206	198
0.001 \times Normal(0,0.1)	10	0	2	0	5	3
	100	18	21	15	21	25
	1000	211	187	208	206	188

Розрахунки для табл. 1 проведено програмою **gurobi 2.0.1** студентської версії AMPL за допомогою такого AMPL-коду:

```

set V_G := {1,2,3,4,5};
set E_G:={(1,2),(1,5),(2,3),(3,4),(4,5)};
display V_G,E_G;

param nCall:=100; param ns{i in 1..5} default 0;
param nv := card(V_G); param eps{i in 1..nv} default 0;
var xs{i in 1..nv} binary;

maximize alpha_G: sum{i in 1..nv} (xs[i]+eps[i]*xs[i]);
subject to constr2 {(i,j) in E_G}: xs[i]+xs[j] <= 1;
option solver gurobi;

for{k in 1..nCall} {
    for{i in 1..nv} {
        let eps[i]:= 0.001*Uniform(-1,1); #Normal(0,0.1)
    }
    solve; display alpha_G;
    if (xs[1]=1 and xs[3]=1) then let ns[1]:=ns[1]+1;
    if (xs[1]=1 and xs[4]=1) then let ns[2]:=ns[2]+1;
    if (xs[2]=1 and xs[4]=1) then let ns[3]:=ns[3]+1;
    if (xs[2]=1 and xs[5]=1) then let ns[4]:=ns[4]+1;
    if (xs[3]=1 and xs[5]=1) then let ns[5]:=ns[5]+1;
}
display sum{i in 1..5}ns[i],ns;

```

З табл. 1 видно, що за десяти запусків програми **gurobi** незалежна множина S_3 жодного разу не була знайдена у випадку рівномірного розподілу, а незалежні множини S_1 та S_3 – за нормального розподілу. Із зростанням кількості запусків (100 та 1000) програми **gurobi** імовірність знаходження кожної множини наближається до 1/5 (20%).

Далі розглянемо суттєво економніший спосіб, який за кожний послідовний запуск програми **gurobi** буде або знаходити нову незалежну множину розміру $\alpha(G)$, або закінчувати роботу, якщо незалежні множини розміру $\alpha(G)$ уже вибрані. Він полягає у такому.

Нехай знайдено m максимальних незалежних множин $S_j, j=1, \dots, m$, і серед них немає збіжних. Щоб знайти нову максимальну незалежну множину, додамо до задачі (1)–(2) такі обмеження:

$$\sum_{i \in V(S_j)} x_i \leq \alpha_K(G) - 1/2, \quad j = 1, \dots, m. \quad (3)$$

Лінійні нерівності (3) відсікають уже знайдені множини $S_j, j=1, \dots, m$, тому розв'язком задачі (2) – (3) буде нова незалежна множина S_{m+1} . Якщо її розмір дорівнює $\alpha(G)$, то множина S_{m+1} буде новою максимальною незалежною множиною і її можна додати до списку множин $S_j, j=1, \dots, m$. Якщо розмір множини S_{m+1} менший ніж $\alpha(G)$, то це означає, що список множин $S_j, j=1, \dots, m$ містить усі максимальні незалежні множини для графа G .

Указаний спосіб програмно реалізовано мовою AMPL. Стартовий список максимальних незалежних множин містить одну з незалежних множин розміру $\alpha(G)$, яка знайдена як розв'язок задачі (1), (2). Для графа C_5 та $nCall$ запусків **gurobi** цей спосіб реалізує AMPL-програма:

```

set V_G := {1,2,3,4,5};
set E_G:={(1,2),(1,5),(2,3),(3,4),(4,5)};
display V_G,E_G;

param nCall:=100; param alphaG1;
set nS default {}; set S{nS} default {};
var xs{i in 1..card(V_G)} binary;

maximize alpha_G: sum{i in 1..card(V_G)} xs[i];
subject to con2 {(i,j) in E_G}: xs[i]+xs[j] <= 1;
subject to con3 {i in nS}: sum{j in S[i]} xs[j] <= alphaG1;

option solver gurobi;
solve; display alpha_G; let alphaG1:=alpha_G-0.5;
let nS:={1}; let S[1] := {j in V_G: xs[j]>=0.99};
for{i in 2..nCall} {
    solve; display alpha_G;
    if (alpha_G<alphaG1) then break;
    let nS:=nS union {i}; let S[i] := {j in V_G: xs[j]>=0.99};
}
display S;

```

За п'ять запусків програми **gurobi** AMPL-код знаходить усі максимальні незалежні множини графа C_5 у такому порядку: $S[1]=\{2,5\}$, $S[2]=\{1,3\}$,

$S[3]=\{3,5\}$, $S[4]=\{1,4\}$, $S[5]=\{2,4\}$. Для налаштування програми на пошук максимальних незалежних множин відповідного графа достатньо в AMPL-коді перші два оператори замінити на оператори з описом множини вершин та множини ребер цього графа.

Програми **CPLEX** та **gurobi** з NEOS-сервера дозволили знайти усі максимальні незалежні множини у графах **Queen8_8**, **Queen7_7**, **Queen6_6**, **Queen5_5** із сайта за адресою <http://mat.gsia.cmu.edu/COLOR04/INSTANCES/>. Так, 64-вершинний граф Queen8_8 містить 92 незалежні множини розміру 8, 49-вершинний граф Queen7_7 – сорок множин розміру 7, 36-вершинний Queen6_6 – чотири множини розміру 6, а 25-вершинний граф Queen5_5 – десять множин розміру 5. У графі Queen6_6 незалежні множини є такі: $S[1]=\{4,7,17,20,30,33\}$, $S[2]=\{2,10,18,19,27,35\}$, $S[3]=\{5,9,13,24,28,32\}$; $S[4]=\{3,12,14,23,25,34\}$.

2. Булева задача для $\alpha_k(G)$ і алгоритм Візинга-Плесневича. Розглянемо наступну задачу: у заданому графі G потрібно знайти K попарно непересічних незалежних множин, сумарний розмір (сума вершин, які входять в незалежні множини) яких є максимальний. Цей розмір будемо позначати $\alpha_k(G)$.

Нехай $x_{ki} \in \{0,1\}$ – булева змінна, яка дорівнює одиниці, якщо вершина $i \in V(G)$ включається в k -ту незалежну множину вершин графа G ($k = 1, \dots, K$), та нульо – у протилежному випадку. Щоб знайти $\alpha_k(G)$, достатньо знайти один із розв'язків задачі булевого лінійного програмування:

$$\alpha_k(G) = \max_{x_{ki} \in \{0,1\}} \sum_{k=1}^K \sum_{i \in V(G)} x_{ki} \quad (4)$$

за обмежень

$$x_{ki} + x_{kj} \leq 1, \quad k = 1, \dots, K, \quad (i, j) \in E(G), \quad (5)$$

$$\sum_{k=1}^K x_{ki} \leq 1, \quad i \in V(G). \quad (6)$$

Задача (4)–(6) містить $K \times |V(G)|$ булевих змінних та $(K \times |E(G)| + |V(G)|)$ обмежень – лінійних нерівностей. Обмеження (5) означають, що коли у графі G вершини i та j з'єднані ребром, то тільки одна з них може належати k -тій незалежній множині вершин графа G . Обмеження (6) означають, що кожна з вершин графа G може включатися в одну з незалежних множин, та гарантують попарну неперетинність незалежних множин у графі G .

Число $\alpha_k(G)$ та відповідній йому незалежні множини S_1, \dots, S_K можна знайти за допомогою заміни першого і другого операторів та модифікації четвертого оператора у наступній AMPL-програмі:

```

set V_G := {1,2,3,4,5};
set E_G:={(1,2),(1,5),(2,3),(3,4),(4,5)};
display V_G,E_G;

param K:=2; param nv:=card(V_G);
set kk:={1..K}; set S{kk} default {};
var xs{k in 1..K, i in 1..card(V_G)} binary;

maximize alpha_Gk: sum{k in 1..K, i in 1..nv}xs[k,i];
subject to con4 {k in 1..K, (i,j) in E_G}: xs[k,i]+xs[k,j] <= 1;
subject to con5 {i in V_G}: sum{k in 1..K} xs[k,i] <= 1;

option solver gurobi;
solve; display alpha_Gk;
for{k in 1..K} {
    let S[k] := {j in V_G: xs[k,j]>=0.9};
}
display S;

```

Програма налаштована на пошук двох незалежних множин у графі C_5 за допомогою розв'язання задачі (4) – (6). Вона знаходить незалежні множини: $S[1]=\{1,3\}$, $S[2]=\{2,5\}$, яким відповідає $\alpha_2(C_5)=2$. Якщо в AMPL-коді четвертий оператор модифікувати на оператор "param K:=3", то отримаємо $\alpha_3(C_5)=5$, якому відповідають незалежні множини: $S[1]=\{1,3\}$, $S[2]=\{2,4\}$, $S[3]=\{5\}$.

Якщо для графа $G=(V(G),E(G))$ у першому операторі описати множину вершин $V(G)$, а у другому – множину ребер $E(G)$, то програмою можна скористатися для пошуку $\alpha_k(G)$ для того значення параметра "K", який задається у четвертому операторі. Так, наприклад, $\alpha_5(G)=40$ для 64-вершинного графа Queen8_8 програма **CPLEX** знаходить за 1.42 секунди, а програма **gurobi** – за 4.37 секунди.

Для пошуку $\alpha_k(G)$ та відповідних йому незалежних множин можна скористатися задачею (1), (2) для числа незалежності графа $G_K = G \times I_K$, отриманого як декартів добуток графа G та повного K -вершинного графа $I_K = (V(I_K), E(I_K))$. Справедливим є такий результат.

Лема 1. $\alpha_k(G) = \alpha(G \times I_K)$.

Доведення. Граф $G_K = (V(G_K), E(G_K))$ містить множину вершин $V(G_K) = \{(k, i) : k \in V(I_K), i \in V(G)\}$ (всього $K \times |V(G)|$) і множину ребер

$E(G_K) = \{E_1(G_K), E_2(G_K)\}$, де $E_1(G_K) = \{(k, i), (k, j) : k \in V(I_K), (i, j) \in E(G)\}$ та $E_2(G_K) = \{(k, i), (k', i) : (k, k') \in E(I_K), i \in V(G)\}$. Для графа G_K оптимізаційна задача (1), (2) має такий вигляд:

$$\alpha(G_K) = \max_{x_{ki} \in \{0,1\}} \sum_{k=1}^K \sum_{i \in V(G)} x_{ki} \quad (7)$$

за обмежень

$$x_{ki} + x_{kj} \leq 1, \quad k = 1, \dots, K, \quad (i, j) \in E(G), \quad (8)$$

$$x_{ki} + x_{k'i} \leq 1, \quad 1 \leq k < k' \leq K, \quad i \in V(G). \quad (9)$$

Легко бачити, що цільова функція (7) та обмеження (8) задачі (7) – (9) збігаються з цільовою функцією (4) та обмеженням (5) задачі (4) – (6). Обмеження (9) задачі (7) – (9) та обмеження (6) задачі (4) – (6) характеризують два різні способи опису незалежних множин для повних підграфів $I_K(i)$, які у графі G_K відповідають вершинам $i \in V(G)$. Обмеження (8) задають лінійними нерівностями для кожного з ребер підграфа $I_K(i)$, а обмеження (6) – лінійними нерівностями для клік розміру K , якими будуть підграфи $I_K(i)$, $i \in V(G)$. Враховуючи, що змінні в обох задачах є булеві, обмеження (8) та (6) описують одні й ті ж незалежні множини. Лема доведена.

Обидві задачі можуть бути використані в алгоритмі Візинга-Плесневича для пошуку $\chi(G)$ – мінімальної кількості кольорів для розфарбування вершин графа G , щоб суміжні вершини були розфарбовані різними кольорами. В основі алгоритму лежить така властивість чисел $\alpha_K(G)$ та $\alpha(G_K)$.

Лема 2. $\chi(G) = \min \{k : \alpha(G \times I_k) = |V(G)|\} = \min \{k : \alpha_k(G) = |V(G)|\}$.

Перша рівність у лемі 2 є наслідком теореми 1 [3], а друга рівність випливає з леми 1.

Згідно з лемою 2 та методом дихотомії алгоритм Візинга-Плесневича для пошуку $\chi(G)$ у графі, для якого $1 \leq |E(G)| < |V(G)| \times (|V(G)| - 1)/2$, має такий вигляд.

Ініціалізація. На ітерації $k = 0$ маємо $\chi_{\min} = 1$ та $\chi_{\max} = |V(G)|$. Оскільки $1 \leq |E(G)| < |V(G)| \times (|V(G)| - 1)/2$, то хроматичне число $\chi(G)$ знаходиться всередині інтервалу $[\chi_{\min}; \chi_{\max}]$. Перейдемо до чергової ітерації зі значеннями χ_{\min} та χ_{\max} .

Ітераційний процес. Нехай на k -ї ітерації знайдені χ_{\min} та χ_{\max} . Для переходу до $(k+1)$ -ї ітерації виконуємо такі дії.

1. Якщо $\chi_{\max} - \chi_{\min} < 1.5$, то $\chi(G) = \chi_{\max}$, і $itn = k$ і зупиняємося.

2. Обчислюємо $K = \lceil (\chi_{\min} + \chi_{\max})/2 \rceil$, де $\lceil \cdot \rceil$ – найближче ціле число.

Знаходимо $\alpha = \alpha_K(G)$, вирішуючи задачу (4) – (6), або $\alpha = \alpha(G_K)$, вирішуючи задачу (7) – (9).

3. Якщо $\alpha = |V(G)|$, то $\chi_{\max} = \alpha$, інакше $\chi_{\min} = \alpha$.

4. Переходимо до $(k+1)$ -ї ітерації з новими χ_{\min} та χ_{\max} .

Для графа G наведений алгоритм знаходить хроматичне число $\chi(G)$ не більше ніж за $(\log_2(|V(G)|)+1)$ ітерацій.

3. Задачі керування програмними проектами. Алгоритм Візинга-Плесневича дозволяє опрацювати обмеження традиційних методів календарно-ресурсного планування для спеціальних задач керування програмним проектом.

Задача 1. Оптимізація розкладу автономного тестування компонентів повторного використання в складі критичної програмної системи.

Нехай програмна система є результатом збірки $n \geq 2$ незалежних компонентів повторного використання c_1, \dots, c_n . Кожний із компонентів має тестиуватися на $k \geq 1$ спеціалізованих стендах або каркасах автономного тестування s_1, \dots, s_k упродовж деякої одиниці часу проекту (дня, тижня тощо) $m \geq 1$ тестерами t_1, \dots, t_m . При цьому одночасне тестування кількох компонентів на одному стенді є неможливе, і для кожного тестера t_j визнано множину

$$O_j = \{o(c_i, s_u), i=1, \dots, n, u=1, \dots, k\} \neq \emptyset, \quad (10)$$

операцій тестування компонента c_i на стенді s_u , які він може виконувати згідно зі своїми рольовими повноваженнями.

Необхідно скласти розклад тестування з мінімальним терміном.

Для розв'язання задачі побудуємо граф з множинами вершин $V = \{o(c_i, s_u), i=1, \dots, n, u=1, \dots, k\}$ та ребер $E \subseteq V \otimes V$, де ребра поєднують вершини, відповідні операціям, сумісне виконання яких за одну одиницю часу неможливе згідно з (10):

$$\begin{aligned} ((o(c_i, s_u), o(c_j, s_v)) \in E) \leftrightarrow & (i=j)!(u=v)!(\exists j, 1 \leq j \leq m \mid \\ & (o(c_i, s_u) \in O_j, o(c_j, s_v) \neq O_j) \wedge (o(c_i, s_u) \neq O_j, o(c_j, s_v) \in O_j)). \end{aligned} \quad (11)$$

Після мінімального правильного розфарбування графа за допомогою алгоритму Візинга-Плесневича з урахуванням співвідношення (11) отримане хроматичне число χ визначатиме тривалість тестування. Підмножини вершин, розфарбованих одним кольором $l=1,\dots,\chi$, відповідають операціям тестування, виконуваним у програмному проекті.

Поставлена задача зберігає значущість і для автономного тестування компонентів на різних каркасах (JUnit, jBehave, Qt, Google Test тощо).

Приклад 1. Нехай $n=4; k=2; m=3$ і тестер t_1 тестує компоненти c_1, c_4 , а тестери t_2, t_3 – компоненти c_2, c_3 . Побудова графа з вершинами–операціями $o(c_i, s_u)$ і застосування до нього алгоритму Візинга-Плесневича дозволяє отримати розфарбований граф із хроматичним числом 4 (рис. 2: операції, що виконуються впродовж l -ї одиниці часу, позначено відповідно білим і чорним кольорами, сірими крапками та сірим штрихуванням). Отриманий розклад тестування подано в табл. 2

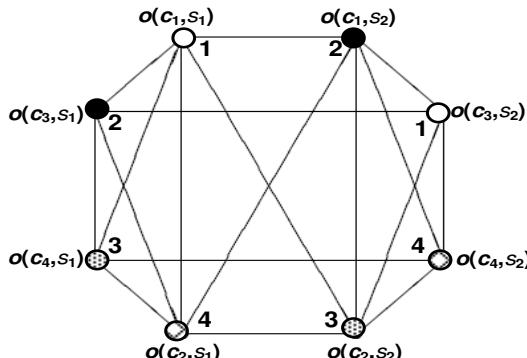


Рис. 2. Граф для побудови розкладу тестування, розфарбований фарбами 1-4

Таблиця 2

Розклад автономного тестування компонентів програмної системи

Одинаця часу	Стенд (або каркас) 1	Стенд (або каркас) 2
1	Перший компонент	Третій компонент
2	Третій компонент	Перший компонент
3	Четвертий компонент	Другий компонент
4	Другий компонент	Четвертий компонент

Задача 2. Формування ядер незалежних команд у критичному програмному проекті.

Визначити мінімальну кількість та склад ядер згуртованих підкоманд для незалежного розроблення версій модулів критичної програмної системи, члени яких повинні мати досвід успішної спільної роботи, на підставі матриці сумісності фахівців-кандидатів d_1, \dots, d_n $D = \|d_{ij}\|_{i,j=1,\dots,n}, n \geq 3$:
 $d_{ij} = 1$, якщо досвід наявний; $d_{ij} = 0$, якщо він відсутній.

Для розв'язання задачі побудуємо граф з множинами вершин $V = \{d_i, i=1, \dots, n\}$ та ребер $E \subseteq V \otimes V$, де ребра поєднують фахівців без досвіду успішної спільної роботи: $((d_i, d_j) \in E) \leftrightarrow d_{ij} = 0$. Після мінімального правильного розфарбування графа за розглянутим алгоритмом Візинга-Плесневича отримане хроматичне число χ визначатиме кількість формованих ядер підгруп (термін "ядро" застосовуємо тут тому, що серед фахівців-кандидатів d_1, \dots, d_n можуть бути відсутні носії необхідних функціональних компетенцій, які у цьому випадку необхідно добирати додатково). Самі ж ці ядра являтимуть собою підмножини вершин, розфарбовані одним кольором $l = 1, \dots, \chi$.

Сформульована задача потребує розв'язання також і в ситуаціях формування мінімальної кількості згуртованих команд для виконання портфелю проектів, подібних між собою за призначенням.

Приклад 2. Нехай $n=10$ і в матриці сумісності D $d_{14}, d_{17}, d_{18}, d_{25}, d_{26}, d_{39}, d_{310}, d_{47}, d_{48}, d_{56}, d_{78}, d_{910} = 1$, а решта елементів нульові. Побудова графа з вершинами—фахівцями d_i і застосування до нього алгоритму Візинга дозволяє отримати розфарбований граф з хроматичним числом 3 (рис. 3: вершини, належні до складу трьох формованих ядер, позначені відповідно чорним і білим кольорами та штрихуванням).

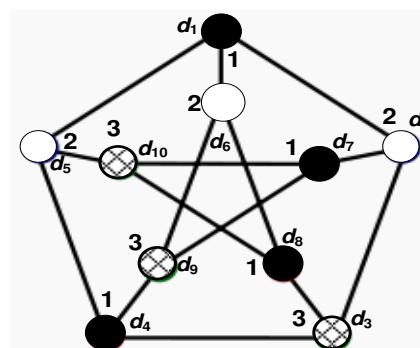


Рис. 3. Граф для побудови ядер незалежних груп, розфарбований фарбами 1-3

Слід зазначити, що це – відомий граф Петерсена [7], широко застосований у теорії графів у ролі демонстраційного прикладу. Відповідний йому склад ядер трьох незалежних груп має вигляд $C_1 = \{d_1, d_4, d_7, d_8\}$, $C_2 = \{d_2, d_5, d_6\}$, $C_3 = \{d_3, d_9, d_{10}\}$.

Висновки. Задачі булевого лінійного програмування для незалежної множини максимального розміру і попарно неперетинних незалежних множин максимального сумарного розміру та відповідні їм AMPL-програми можуть бути використані для розрахунків у різноманітних задачах керування програмними проектами: формування найбільш згуртованої команди програмного проекту; визначення множини програмних проектів у портфелі, які можуть виконуватися одночасно; формування та аналіз підкоманд для складного розподіленого програмного проекту та ін. На їх основі легко реалізувати алгоритм Візинга-Плесневича для задачі розфарбування вершин графа мінімальною кількістю кольорів, який дозволяє знайти оптимальний розклад автономного тестування компонентів у складі критичної програмної системи та оптимальний склад ядер незалежних команд у критичному програмному проекті.

Наведені у статті AMPL-коди за відповідного їх доопрацювання, а також програми **CPLEX** і **gurobi** можуть бути застосовані під час виконання лабораторних робіт із дисципліни «Керування програмними проектами» у вищих навчальних закладах.

Бібліографічні посилання

1. **Кристофидес, Н.** Теория графов. Алгоритмический подход [Текст] / Н. Кристофидес. – М: Мир, 1978. – 432 с.
2. **Гэри, М.** Вычислительные машины и труднорешаемые задачи [Текст] / М. Гэри, Д. Джонсон. – М.: Мир, 1982. – 416 с.
3. **Визинг, В.Г.** К проблеме минимальной раскраски вершин графа [Текст] / В.Г Визинг, Г.С. Плесневич // Сибирский математический журнал. – 1965. – Т. 6. –№ 1. – С. 234–236.
4. **NEOS Solver** [Electronic resource]. – Access mode: <https://neos-server.org/neos/solvers/>. – Title from the screen.
5. **Fourer, R.** AMPL, A Modeling Language for Mathematical Programming [Text] / R. Fourer, D. Gay, B. Kernighan.– Belmont: Duxbury Press, 2003. – 517 p.
6. **Стецюк, П.И.** Об ЛП-ориентированных верхних оценках для взвешенного числа устойчивости графа [Текст] / П.И. Стецюк, А.П. Лиховид // Кибернетика и системный анализ. – 2009. – № 1. – С. 157–170.
7. **Нікольський, Ю.В.** Дискретна математика [Текст] / Ю.В. Ні科尔ський, В.В. Пасічник, Ю.М. Щербина. – К.: Видавничча група ВНВ, 2007. – 368 с.

Надійшла до редколегії 26.05.2016