УДК 519.8 https://doi.org/10/15407/pp2022.03-04.073

ПОШУК МАКСИМАЛЬНИХ НЕЗАЛЕЖНИХ МНОЖИН ВЕРШИН ГРАФА ДЛЯ ВДОСКОНАЛЕННЯ ПРОГРАМНИХ ПРОЕКТІВ

Ольга Слабоспицька, Петро Стецюк, Ольга Хом'як

Зафіксовано потребу вдосконалення програмного проекту за рахунок непротирічної інтеграції технологічно-описового та нормативного підходів проектного менеджменту шляхом налаштування класичних задач дискретної оптимізації на графах для задач керування програмним проектом, кращі практики розв'язання яких недостатньо регламентовані в руслі технологічного підходу. Запропоновано клас задач керування програмним проектом для демонстрації перспективності зазначеної інтеграції. Досліджено дві задачі булевого лінійного програмування для знаходження деякої незалежної множини максимального розміру (розділ 1) та алгоритм знаходження всіх можливих незалежних множин максимального розміру (розділ 2). У розділі 3 надано постановку задачі знаходження заданої кількості незалежних множин, які не перетинаються й мають найбільшу суму кількостей вершин у незалежних множинах. На її основі описано алгоритм Візинга-Плесневича для розфарбування вершин графа мінімальною кількістю кольорів. Для розв'язання булевих задач використано спеціалізовану мову математичного програмування AMPL і відповідну їй програму-солвер *gurobi.* Для основних розроблених алгоритмів наведено рамкові версії коду мовою AMPL та результати їх запускання.
У розділі 4 розглянуто показові приклади вдосконалення програмного проекту за допомогою розроблених алгоритмів: формування з 25 фахівців, між якими в попередніх проектах мали місце конфлікти, згуртованих безконфліктних під-команд для портфеля програмних проектів; оптимізації розкладу автономного тестування компонентів повторного використання в складі критичної програмної системи; формування ядер незалежних команд у критичному програмному проекті.

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

The need is fixed to software project enhancing with seamless integration of technological-descriptive and normative project management approaches by means of classical Graph Discrete Optimization Problems tailoring for software project management tasks, poorly equipped with best practices within technological approach. Class of software project management tasks is proposed to demonstrate the benefits of such integration. Two Boolean linear programming problems are investigated for searching some maximum size independent set (Section 1) and an algorithm for searching all possible maximum size independent sets (Section 2). Section 3 presents Problem Statement for searching a given number of non-intersecting independent sets with maximum sum of vertices' numbers within independent sets. Based on it, Vizing-Plesnevich algorithm is described for coloring the graph vertices with the minimum number of colors.
To solve Boolean problems, both specialized mathematical programming language AMPL and corresponding solver program named *gurobi* are used. For basic algorithms developed, reference AMPL code versions are given as well as their running results.
Illustrative examples of software project enhancing with the algorithms elaborated are considered in Section 4, namely: 25 specialists being conflicted during their previous projects partitioning into coherent conflict-free sub-teams for software projects portfolio; schedule optimization for autonomous testing of reusable components within a critical software system; cores composing for independent teams in a critical software project.

Keywords: software project, resource allocation, Boolean linear programming problem, independent set of graph vertices, minimal graph coloring, Vizing-Plesnevich algorithm.

Вступ

Однією з актуальних тенденцій стрімкого розвитку індустрії програмного забезпечення в світі та Україні є наразі повно-аспектне впровадження проектних форм створення програмних продуктів у найрізноманітніших предметних областях – від медицини й освіти до медіа-індустрії та міжнародного співробітництва. Однак практика програмних проектів, узагальнена, зокрема, в регулярних звітах авторитетної аналітичної організації Standish Group (https://www.standishgroup.com/) висвітлює дедалі нагальнішу потребу непротирічної інтеграції в керуванні програмним проектом двох базових підходів проектного менеджменту:

- технологічно-описового, засади якого зафіксовано в Зводі знань з управління проектами (7 ред.) [1];
- нормативного, спрямованого на розроблення взаємоузгоджених приписних правил і процедур взаємодії учасників проекту з гарантованими бажаними властивостями за допомогою (імовірнісного) календарно-мережного планування й управління з потужним сучасним інструментарієм (дискретної) оптимізації на графах, який і гарантує оптимальність отримуваних рішень [2].

Важливим аспектом цієї інтеграції є виокремлення класичних задач алгоритмів дискретної оптимізації та їх налаштування для ресурсно прийнятного отримання оптимальних розв'язків тих задач керування програмним проектом, для яких технологічний підхід не надає однозначних рекомендацій щодо вибору кращих і гірших практик, у ролі прототипу для доцільного подальшого вдосконалення. Внаслідок різнорідності й складності сучасних програмних продуктів і методологій їх розроблення показовими прикладами таких задач є формування з наявних фахівців найбільш згуртованої команди, а також виявлення технологічно незалежних робіт проекту та робіт, що доцільно виконувати одночасно.

Саме ці задачі розглянуті авторами в [3], де їм зіставлені спеціальні постановки класичних задач про максимальну незалежну множину та хроматичне число неорієнтованого графа. Стаття, презентована наразі, містить подальші результати досліджень [3]. Вона має подвійну мету:

– засвідчити перспективність залучення алгоритмів дискретної оптимізації на графах для вдосконалення програмного проекту та окреслити першочергові задачі, для яких воно доцільне;

– уточнити постановки задачі про максимальну незалежну множину, подані в [3], і надати більш ефективні коди мовою AMPL для знаходження усіх максимальних незалежних множин і максимальної суми незалежних множин вершин неорієнтованого графа.

1. Булеві задачі для знаходження числа незалежності та їх властивості

Множину вершин графа G називають незалежною, якщо ніякі дві вершини цієї множини не є суміжними (не з'єднані ребром). Індукований цією множиною підграф складається з ізольованих вершин, їх кількість визначає розмір незалежної множини вершин графа. Оптимізаційна задача про максимальну незалежну множину формулюється таким чином: у заданому неорієнтованому 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 [4, 5].

Задача про максимальну незалежну множину 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 [6], що задачу знаходження хроматичного числа можна звести до ітеративної послідовності задач знаходження числа незалежності графів, отриманих як декартів добуток повних графів та копій графа G.

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 – в протилежному випадку. Тут $V(G)$ – множина вершин графа G, $E(G)$ – множина його ребер. Щоб знайти число незалежності $\alpha(G)$, достатньо знайти один із розв'язків задачі булевого лінійного програмуваннях [7]:

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

за обмежень

$$x_i + x_j \le 1, \quad (i,j) \in E(G). \tag{2}$$

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.

Задача (1), (2) може бути переформульована у вигляді задачі булевого лінійного програмування, яка використовується для опису максимального k-плексу [8]. Для цього достатньо обмеження (2) замінити на такі:

$$\sum_{j \in \Gamma(i)} x_j \le d_i \left(1 - x_i\right), \ \forall i \in V\left(G\right). \tag{3}$$

Задача (1), (3) містить $|V(G)|$ булевих змінних та $|V(G)|$ обмежень у формі лінійних нерівностей. Тут граф G представлено у формі $G = (V(G), \Gamma(G))$, де $\Gamma(G) = \{\Gamma(i),\ i = 1,\ldots,n\}$, а $\Gamma(i)$ – кінцеві вершини тих дуг, у яких початковою є вершина i, яка має ступінь $d_i = |\Gamma(i)|$. Обмеження (3) означають, що коли вершина належить незалежній множині вершин графа G, то жодна суміжна з нею вершина не може належати цій незалежній множині. В задачах (1), (3) кількість лінійних обмежень буде значно меншою, ніж у задачі (1), (2), для графів з великою кількістю ребер.

Число незалежності $\alpha(G)$ визначається однозначно, але йому може відповідати багато максимальних незалежних множин графа G. Так, наприклад, граф W_6 (має форму колеса на основі C_5, циклу з п'яти вершин), множину ребер та матрицю суміжності якого наведено на рис. 1, містить п'ять незалежних множин розміру $\alpha(W_6) = 2$. На рис. 1 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, позначено чорним кольором.

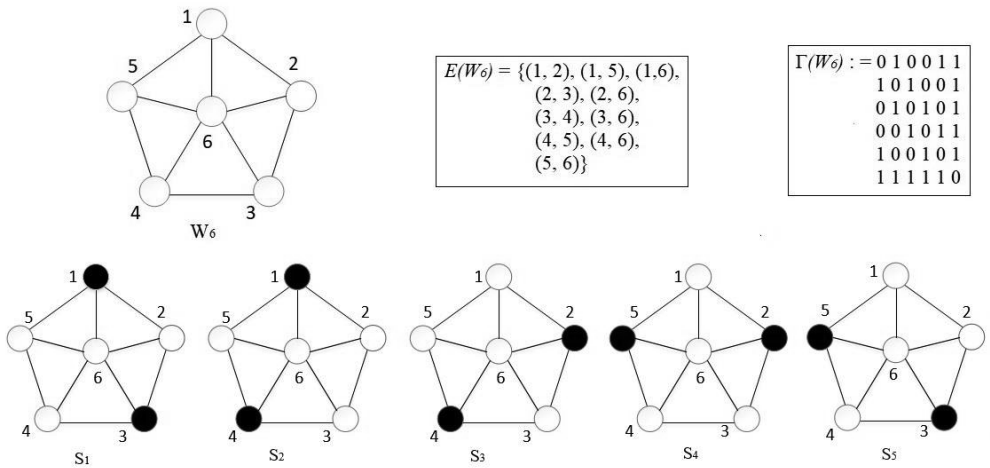

Рисунок 1. Граф W_6 та його п'ять незалежних множин максимального розміру

$$\alpha_\varepsilon(G) = \max_{x_i \in \{0,1\}} \sum_{i \in V(G)} \left(x_i + \varepsilon_i x_i\right), \tag{4}$$

де випадкові величини $|\varepsilon_i| << 1$ для всіх $i \in V(G)$. Залежно від вибраних значень для збурень ε_i, $i = 1,\ldots,|V(G)|$, як розв'язок задачі (4), (2) або задачі (4), (3) отримаємо ту чи іншу незалежну множину розміру $\alpha(G)$. Цей спосіб вимагає значної кількості розв'язань задачі (4), (2) або задачі (4), (3), що підтверджують результати обчислювального експерименту для знаходження незалежних множин $S_1,\ldots,S_5$ у графі W_6 за допомогою солвера **gurobi** [9]. 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 отримані за допомогою розв'язання задачі (4), (2), колонки 8–12 отримані за допомогою розв'язання задачі (4), (3)) для різних кількостей запусків солвера **gurobi**: $nCalls = 10$, $nCalls = 100$ і $nCalls = 1000$ (колонка 2).

Таблиця 1. Частота знаходження максимальних незалежних множин $S_1,\ldots,S_5$ для графа W_6 за рівномірного та нормального розподілів випадкових збурень ε_i, $i = 1,...,6$

		Для булевої задачі (4), (2)					Для булевої задачі (4), (3)				
$\varepsilon_i, i = 1,...,6$	$nCalls$	$n(S_1)$	$n(S_2)$	$n(S_3)$	$n(S_4)$	$n(S_5)$	$n(S_1)$	$n(S_2)$	$n(S_3)$	$n(S_4)$	$n(S_5)$
0.001 Uniform(-1,1)	10	3	2	1	1	3	3	2	1	1	3
	100	19	22	24	15	20	19	22	24	15	20
	1000	199	204	218	180	199	199	204	218	180	199
0.001 Normal(-1,1)	10	1	2	0	3	4	1	2	0	3	4
	100	20	17	20	19	24	20	17	20	19	24
	1000	197	193	206	200	204	197	193	206	200	204

Розрахунки для табл. 1 проводилися програмою-солвером **gurobi 2.0.1** студентської версії AMPL [10, 11]. Частоти знаходження множин $S_1,\ldots,S_5$ для булевої задачі (4), (2) обчислювались за допомогою AMPL-коду:

```
     # Подання графу G для булевої задачі (4),(2)
     set V_G := {1,2,3,4,5,6}; # вершини графа
     set E_G:={(1,2),(1,5),(1,6),(2,3),(2,6),(3,4),(3,6),(4,5),(4,6),(5,6)}; #
ребра графа
     display V_G,E_G;

     # Опис та задання параметрів
     param nCalls:=100; #кількість запусків солвера
     param ns{i in 1..5} default 0; #кількості множин S_i
     param nv := card(V_G); param eps{i in 1..nv} default 0;

     # Опис булевої задачі (4), (2)
     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; # вибір солвера

     # nCalls разів розв'язуємо задачу (4), (2)
     for{k in 1..nCalls} {
        for{i in 1..nv} {
          let eps[i]:= 0.001*Uniform(-1,1); #0.001*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;
```

а для булевої задачі (4), (3) обчислювались за допомогою такого AMPL-коду:

```
     # Опис графу G для булевої задачі (4), (3)
     set V_G := {1,2,3,4,5,6}; # вершини графа
     param nv := card(V_G); # кількість вершин графа
     param I_G{i in 1..nv, j in 1..nv} >=0 <=1;
     param d{i in 1..nv} = sum{j in 1..nv} I_G[i,j];

     # Опис та задання параметрів
     param nCalls:=100; #кількість запусків солвера
     param ns{i in 1..5} default 0; #кількості множин S_i
     param eps{i in 1..nv} default 0;

     # Опис булевої задачі (4), (3)
     var xs{i in 1..nv} binary;
     maximize alpha_G: sum{i in 1..nv} (xs[i]+eps[i]*xs[i]);
     subject to constr3 {i in 1..nv}: sum{j in 1..nv} I_G[i,j]*xs[j] <= d[i]*(1-
xs[i]);

     option solver gurobi; # вибір солвера

     data;
     param I_G: 1 2 3 4 5 6:=
               1 0 1 0 0 1 1
               2 1 0 1 0 0 1
               3 0 1 0 1 0 1
               4 0 0 1 0 1 1
               5 1 0 0 1 0 1
               6 1 1 1 1 1 0;
     display V_G,I_G;

     # nCalls разів роз'язуємо задачу (4), (3)
     for{k in 1..nCalls} {
        for{i in 1..nv} {
          let eps[i]:= 0.001*Uniform(-1,1); #0.001*Normal(-1,1)
        }
        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 частоти знаходження кожної з п'яти незалежних множин, які отримані за допомогою розв'язання задачі (4), (2), співпадають з частотами, отриманими за допомогою розв'язання задачі (4), (3). Зауважимо, що за десяти запусків солвера **gurobi** незалежна множина S_3 жодного разу не була знайдена при нормальному розподілі. Із зростанням кількості запусків (100 та 1000) солвера **gurobi** імовірність знаходження кожної множини наближається до 1/5 (20%).

2. Алгоритм знаходження усіх максимальних незалежних множин

У даному розділі розглянемо суттєво економніший спосіб для знаходження усіх максимальних незалежних множин вершин графа G, який за кожний послідовний запуск солвера **gurobi** буде або знаходити нову незалежну множину розміру $\alpha(G)$, або закінчувати роботу, якщо незалежні множини розміру $\alpha(G)$ уже вибрані. Він полягає у наступному.

Нехай знайдено m максимальних незалежних множин $S_j, j = 1,\ldots,m,$ і серед них немає співпадаючих. Щоб знайти нову максимальну незалежну множину, додамо до задачі (1), (2) або до задачі (1), (3) обмеження:

$$\sum_{i \in V(S_j)} x_i \le \alpha_K(G) - 1/2, \quad j = 1,...,m. \tag{5}$$

Лінійні нерівності (5) відсікають уже знайдені множини $S_j, j=1,\ldots,m$, тому розв'язком задачі (1), (2), (5) або задачі (1), (3), (5) 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 $S_j, j=1,\ldots,m$. Якщо розмір множини S_{m+1} менший ніж $\alpha(G)$, то це значить, що список множин $S_j, j=1,\ldots,m$ містить усі максимальні незалежні множини для графа G.

Вказаний алгоритм знаходження *nCalls* розв'язків задачі (1), (2), (5) або задачі (1), (3), (5) програмно реалізований мовою AMPL. Стартовий список максимальних незалежних множин містить одну із незалежних множин розміру $\alpha(G)$, яка знайдена як розв'язок задачі (1), (2). Для графа W_6 та *nCalls* = 100 послідовних запусків солвера **gurobi** алгоритм на основі задачі (1), (2), (5) реалізує AMPL-код:

```
      # Опис графу G для булевої задачі (1), (2), (5)
      set V_G := {1,2,3,4,5,6};      # вершини графа
      set E_G:={(1,2),(1,5),(1,6),(2,3),(2,6),(3,4),(3,6),(4,5),(4,6),(5,6)}; #
ребра графа
      display V_G,E_G;

      # Опис та задання параметрів
      param nCalls:=100; #кількість запусків солвера
      param alphaG1;
      set nS default {}; set S{nS} default {};

      # Опис булевої задачі (1), (2), (5)
      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 con5 {i in nS}: sum{j in S[i]} xs[j] <= alphaG1;

      option solver gurobi; # вибір солвера

      #
      # nCalls разів розв'зуємо задачу (1), (2), (5)
      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..nCalls} {
         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 nCalls,S; # друкуємо кількість запусків солвера та отримані макси-
мальні незалежні множини
```

За п'ять запусків солвера **gurobi** цей AMPL-код знаходить усі максимальні незалежні множині графа W_6 у такому порядку: S[1]={1,3}, S[2]={2,4}, S[3]={1,4}, S[4]={3,5}, S[5]={2,5}. Для його налаштування на пошук максимальних незалежних множин відповідного графа достатньо в AMPL-коді перших два оператори замінити на оператори з описом множини вершин та множини ребер цього графа.

Алгоритм знаходження *nCalls* розв'язків задачі (1), (3), (5) для графа W_6 та *nCalls* = 100 послідовних запусків солвера **gurobi** реалізує такий AMPL-код:

```
      # Опис графу G для булевої задачі (1), (3), (5)
      set V_G := {1,2,3,4,5,6}; # вершини графа
      param nv := card(V_G); # кількість вершин графа
      param I_G{i in 1..nv,j in 1..nv} >=0 <=1;
      param d{i in 1..nv} = sum{j in 1..nv} I_G[i,j];

      # Опис та задання параметрів
      param nCalls:=100; #кількість запусків солвера
      param alphaG1;
      set nS default {}; set S{nS} default {};

      # Опис булевої задачі (1), (3), (5)
      var xs{i in 1..nv} binary;
      maximize alpha_G: sum{i in 1..nv} xs[i];
      subject to constr3 {i in 1..nv}: sum{j in 1..nv} I_G[i,j]*xs[j] <= d[i]*(1-
xs[i]);
```

```
      subject to constr5 {i in nS}: sum{j in S[i]} xs[j] <= alphaG1;

      option solver gurobi; # вибір солвера

      # Задання матриці суміжності
      data;
      param I_G: 1 2 3 4 5 6:=
               1 0 1 0 0 1 1
               2 1 0 1 0 0 1
               3 0 1 0 1 0 1
               4 0 0 1 0 1 1
               5 1 0 0 1 0 1
               6 1 1 1 1 1 0;
      display V_G,I_G;

      # nCalls разів розв'зуємо задачу (1), (3), (5)
      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..nCalls} {
         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 nCalls,S; # друкуємо кількість запусків солвера та отримані макси-
мальні незалежні множини
```

За п'ять запусків солвера **gurobi** цей AMPL-код знаходить усі максимальні незалежні множині графа W_6 у такому порядку: S[1]={1,3}, S[2]={2,4}, S[3]={1,4}, S[4]={2,5}, S[5]={3,5}. Для його налаштування на пошук максимальних незалежних множин відповідного графа достатньо в AMPL-коді перший оператор замінити на оператор з описом множини вершин цього графа, а в операторі «data» вказати відповідну графу матрицю суміжності I_G.

AMPL-код для знаходження *nCalls* розв'язків задачі (1), (3), (5) апробовано для формування згуртованих під-команд окремих програмних проектів у складі їх портфеля, тобто розфарбування 25 фахівців – виконавців програмних проектів – різними кольорами так, щоб фахівці, між якими в попередніх проектах мали місце конфлікти, не могли бути зафарбовані одним і тим же кольором (див. задачу 1 в розд.4).

3. Булеві задачі для $\alpha_K(G)$ та алгоритм Візинга-Плесневича

Розглянемо таку задачу: знайти K попарно неперетинних незалежних множин графа G, сумарний розмір (сума вершин, які входять в незалежні множини) яких є максимальним. Цей розмір позначимо $\alpha_K(G)$.

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 ($k = 1,...,K$), та нулю – в протилежному випадку. Щоб знайти $\alpha_K(G)$, достатньо знайти один із розв'язків задачі булевого лінійного програмування:

$$\alpha_K(G) = \max_{x_{ki} \in \{0,1\}} \sum_{k=1}^{K} \sum_{i \in V(G)} x_{ki} \tag{6}$$

за обмежень

$$x_{ki} + x_{kj} \le 1, \quad k = 1,...,K, \ (i,j) \in E(G), \tag{7}$$

$$\sum_{k=1}^{K} x_{ki} \le 1, \quad i \in V(G). \tag{8}$$

Задача (6)–(8) містить $K \times |V(G)|$ булевих змінних та $\left(K \times |E(G)| + |V(G)|\right)$ обмежень – лінійних нерівностей. Обмеження (7) означають, що коли в графі G вершини i та j з'єднані ребром, то лише одна з них може належати k-тій незалежній множині вершин графа G. Обмеження (8) означають, що кожна з вершин графа G може належати одній з незалежних множин і гарантують попарну неперетинність незалежних множин у графі G.

Число $\alpha_K(G)$ та відповідні йому незалежні множини $S_1, \ldots, S_K$ можна знайти за допомогою модифікації перших трьох операторів в AMPL-коді:

```
      param K:=2; # кількість незалежних множин
      set V_G := {1,2,3,4,5,6}; # вершини графа
      set E_G:={(1,2),(1,5),(1,6),(2,3),(2,6),(3,4),(3,6),(4,5),(4,6),(5,6)}; #
ребра графа
      display V_G,E_G;
```

```
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 constr7 {k in 1..K, (i,j) in E_G}: xs[k,i]+xs[k,j] <= 1;
subject to constr8 {i in V_G}: sum{k in 1..K} xs[k,i] <= 1;


option solver gurobi; # вибір солвера

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

Цей AMPL-код налаштований на пошук двох незалежних множин у графі W_6 за допомогою розв'язання задачі (6) – (8). Він знаходить незалежні множини: S[1]={1,3}, S[2]={2,5}; яким відповідає $\alpha_2(C_5)=2$. Якщо в AMPL-коді перший 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 другому операторі описати множину вершин $V(G)$, а у третьому – множину ребер $E(G)$, то AMPL-код може бути використаний для пошуку $\alpha_K(G)$ для того значення параметра «*K*», який задається у першому операторі.

Задача (6) – (8) побудована на основі використання задачі (1), (2) для *K* копій графа G ($K>1$). Якщо для цих же копій графа використовувати задачу (1), (3), то задачу для знаходження $\alpha_K(G)$ можна записати за допомогою задачі булевого лінійного програмування:

$$\alpha_K(G)=\max_{x_{ki}\in\{0,1\}}\sum_{k=1}^{K}\sum_{i\in V(G)}x_{ki} \tag{9}$$

за обмежень

$$\sum_{j\in\Gamma(i)}x_{kj}+\sum_{\substack{t=1,\ldots,K,\\ t\neq k}}\sum_{j\in V(G)}x_{tj}\le\left(d_i+(K-1)\left|V(G)\right|\right)\left(1-x_{ki}\right),\ \forall k\in\{1...K\},\forall i\in V\left(G\right). \tag{10}$$

Задача (9), (10) містить $K\times\left|V(G)\right|$ булевих змінних та $K\times\left|V(G)\right|$ обмежень – лінійних нерівностей. Обмеження (10) означають, що якщо вершина буде належати незалежній множині вершин в *k*-тій копії графа *G*, то ні одна з суміжних з нею вершин не може належати цій незалежній множині.

Зауважимо, що для пошуку $\alpha_K(G)$ та відповідних йому незалежних множин може бути використана задача (1), (2) та задача (1), (3) для числа незалежності графа $G_K=G\times I_K$, отриманого як декартів добуток графа *G* та повного *K*-вершинного графа $I_K=(V(I_K),E(I_K))$. Має місце

Лема 1. $\alpha_K\left(G\right)=\alpha\left(G\times I_K\right)$.

Доведення леми 1 за допомогою формулювання задачі (1), (2) наведено в [3]. Доведення за допомогою формулювання задачі (1), (3) можна зробити за аналогічною схемою.

Задачі (6) – (8) та (9), (10) 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 наступна властивість чисел $\alpha_K(G)$ та $\alpha(G_K)$.

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

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

Для пошуку $\chi(G)$ у графі, для якого $1\le\left|E(G)\right|<\left|V(G)\right|\times\left(\left|V(G)\right|-1\right)/2$, алгоритм Візинга-Плесневича на основі леми 2 та методу дихотомії передбачає такі кроки.

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

Ітеративний процес. Нехай на *k*-й ітерації знайдені $\chi_{\min}$ та $\chi_{\max}$. Для переходу до (*k* + 1)-ї ітерації виконуємо наступні дії.

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

2. Обчислимо $K=\left\lceil\left(\chi_{\min}+\chi_{\max}\right)/2\right\rceil$, де $\lceil\bullet\rceil$ – найближче ціле число. Знайдемо $\alpha=\alpha_K(G)$, вирішуючи задачу (6) – (8) або задачу (9) – (10), або $\alpha=\alpha(G_K)$, вирішуючи задачу (1) – (2) або задачу (1) – (3).

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

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

Наведений алгоритм знаходить хроматичне число графа *G* не більше ніж за $(\log_2\left(\left|V(G)\right|\right)+1)$ ітерацій.

4. Задачі управління програмними проектами

Задача 1. Нехай 25 фахівців – виконавців портфеля програмних проектів впорядковані згідно з алфавітгним порядком їх прізвищ і занумеровані числами від 1 до 25. У табл. 2 наведено дані про конфлікти між ними в попередніх проектах: номер фахівця (перша колонка), номери фахівців, з якими мав місце конфлікт (друга колонка).

Таблиця 2. Дані щодо конфліктів фахівців для графа G_{25}

Номери фахівців, які конфліктували в попередніх проектах					
1	21	10	2,6,16,23,25	19	9,13,17,22,24
2	6,10,11,15,22,23,24	11	2,4,14,15,16,23	20	4,5,12,16,18
3	13,17	12	5,20	21	1,4,8,14
4	5,8,11,14,16,20,21	13	3,7,9,17,19	22	2,6,17,19,24
5	4,8,12,20	14	4,11,15,21	23	2,10,11,16
6	2,10,17,22	15	2,11,14	24	2,9,19,22
7	9,13	16	4,10,11,18,20,23	25	10,16,18
8	4,5,21	17	3,6,13,19,22		
9	7,13,19,24	18	16,20,25		

Для AMPL-коду вершини графу G_{25} задаються як:

```
set V_G := {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,2
4,25};
```

а матриця суміжності I_G має вигляд:

```
param I_G: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 :=
          1  0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0
          2  0 0 0 0 0 1 0 0 0  1  1  0  0  0  0  0  0  0  0  0  0  1  1  1  0
          3  0 0 0 0 0 0 0 0 0  0  0  0  1  0  0  0  1  0  0  0  0  0  0  0  0
          4  0 0 0 0 1 0 0 1 0  0  1  0  0  1  0  1  0  0  0  1  1  0  0  0  0
          5  0 0 0 1 0 0 0 1 0  0  0  1  0  0  0  0  0  0  0  1  0  0  0  0  0
          6  0 1 0 0 0 0 0 0 0  1  0  0  0  0  0  0  1  0  0  0  0  1  0  0  0
          7  0 0 0 0 0 0 0 0 0  1  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0
          8  0 0 0 1 1 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0
          9  0 0 0 0 0 0 0 1 0  0  0  0  0  1  0  0  0  0  0  1  0  0  0  0  1  0
         10  0 1 0 0 0 1 0 0 0  0  0  0  0  0  0  1  0  0  0  0  0  0  1  0  1
         11  0 1 0 1 0 0 0 0 0  0  0  0  0  1  1  1  0  0  0  0  0  0  1  0  0
         12  0 0 0 0 1 0 0 0 0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0
         13  0 0 1 0 0 0 1 0 1  0  0  0  0  0  0  0  1  0  1  0  0  0  0  0  0
         14  0 0 0 1 0 0 0 0 0  0  1  0  0  0  1  0  0  0  0  0  1  0  0  0  0
         15  0 1 0 0 0 0 0 0 0  0  1  0  0  1  0  0  0  0  0  0  0  0  0  0  0
         16  0 0 0 1 0 0 0 0 0  1  1  0  0  0  0  0  0  1  0  1  0  0  1  0  0
         17  0 0 1 0 0 1 0 0 0  0  0  0  1  0  0  0  0  0  1  0  0  1  0  0  0
         18  0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  1  0  0  0  1  0  0  0  0  1
         19  0 0 0 0 0 0 0 0 1  0  0  0  1  0  0  0  1  0  0  0  0  1  0  1  0
         20  0 0 0 1 1 0 0 0 0  0  0  1  0  0  0  1  0  1  0  0  0  0  0  0  0
         21  1 0 0 1 0 0 0 1 0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0
         22  0 1 0 0 0 1 0 0 0  0  0  0  0  0  0  0  1  0  1  0  0  0  0  1  0
         23  0 1 0 0 0 0 0 0 0  1  1  0  0  0  0  1  0  0  0  0  0  0  0  0  0
         24  0 1 0 0 0 0 0 0 1  0  0  0  0  0  0  0  0  0  1  0  0  1  0  0  0
         25  0 0 0 0 0 0 0 0 0  1  0  0  0  0  0  1  0  1  0  0  0  0  0  0  0;
```

Для графа G_{25} алгоритм знайшов $\alpha(G_{25}) = 10$, якому відповідають 16 різних максимальних незалежних множин:

```
set S[1] := 1 3 6 7 8 12 14 18 19 23;      set S[2] := 1 3 6 7 8 12 15 23 24 25;
set S[3] := 1 3 4 6 7 12 15 23 24 25;      set S[4] := 1 3 6 7 8 12 15 19 23 25;
set S[5] := 1 3 6 7 8 12 14 19 23 25;      set S[6] := 1 3 6 7 8 12 14 18 23 24;
set S[7] := 1 3 6 7 8 12 14 23 24 25;      set S[8] := 1 3 6 7 8 14 19 20 23 25;
set S[9] := 1 3 6 7 8 15 19 20 23 25;     set S[10] := 1 3 4 6 7 12 15 18 23 24;
set S[11] := 1 3 6 7 8 12 15 18 19 23;    set S[12] := 1 3 6 7 8 12 15 18 23 24;
set S[13] := 1 3 6 7 8 14 20 23 24 25;    set S[14] := 1 3 4 6 7 12 15 18 19 23;
set S[15] := 1 3 4 6 7 12 15 19 23 25;    set S[16] := 1 3 6 7 8 15 20 23 24 25;
```

і підтвердив, що сімнадцятої максимальної незалежної множини не існує.

Крім підтримки формування прототипних складів згуртованих команд, запропоноване налаштування алгоритму Візинга-Плесневича дозволяє також опрацювати обмеження традиційних методів календарно-ресурсного планування для спеціальних задач управління програмним проектом.

Задача 2. 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 $c_1,...,c_n$. Кожний з компонент має тестуватися на $k \geq 1$ спеціалізованих стендах або каркасах автономного тестування $s_1,...,s_k$ впродовж деякої одиниці часу проекту (дня, тижня тощо) $m \geq 1$ тестерами $t_1,...,t_m$. При цьому одночасне тестування кількох компонентів на одному стенді є неможливим, і для кожного тестера t_j визначено множину

$$O_j = \{o(c_i, s_u), i = 1,...,n, u = 1,...,k\} \neq \varnothing, \tag{11}$$

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 $V = \{o(c_i, s_u), i = 1,...,n, u = 1,...,k\}$ та ребер $E \subseteq V \otimes V$, де ребра поєднують вершини, відповідні операціям, сумісне виконання яких в одну одиницю часу неможливе згідно з (11):

$$\begin{gathered}((o(c_i, s_u), o(c_j, s_v)) \in E) \leftrightarrow (i = j)!(u = v)!(\exists j, 1 \leq j \leq m \mid \\ (o(c_i, s_u) \in O_j, o(c_j, s_v) \neq O_j)!(o(c_i, s_u) \neq O_j, o(c_j, s_v) \in O_j).\end{gathered} \tag{12}$$

Після мінімального правильного розфарбування графу за допомогою алгоритму Візинга-Плесневича з урахуванням співвідношення (12) отримане хроматичне число h визначатиме тривалість тестування. Підмножини вершин, розфарбованих одним кольором $l = 1,...,h$ 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 графу з вершинами–операціями $o(c_i, s_u)$ і застосування до нього алгоритму Візинга-Плесневича дозволяє отримати розфарбований граф з хроматичним числом 4, наведений на рис. 2 (операції, що мають виконуватися впродовж *l*-ї одиниці часу проекту, позначено відповідно білим і чорним кольорами, сірими крапками та сірим штрихуванням).

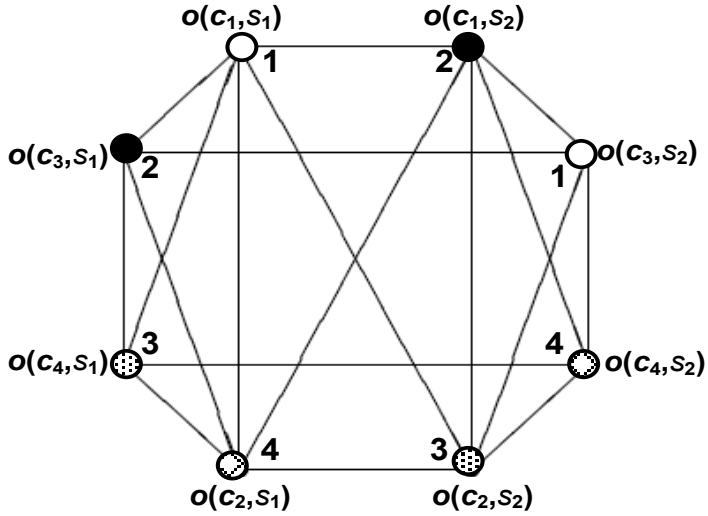

Рисунок 2. Граф для побудови розкладу тестування, розфарбований фарбами 1-4

Отриманий розклад тестування подано в табл. 3.

Таблиця 3. Розклад автономного тестування компонентів програмної системи

Одиниця 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 3. 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 $d_1,...,d_n$ $D=\| d_{ij} \|_{i,j=1,...n}, n \geq 3$: $d_{ij}=1$, якщо досвід наявний; $d_{ij}=0$, якщо він відсутній.

Для розв'язання задачі побудуємо граф з множинами вершин $V=\{d_i, i=1,...,n\}$ 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 графу за розглянутим алгоритмом Візинга-Плесневича отримане хроматичне число визначатиме кількість формованих ядер підгруп (термін "ядро" застосовується тут тому, що серед фахівців-кандидатів $d_1,...,d_n$ можуть бути відсутні носії необхідних функціональних компетенцій, яких у цьому випадку необхідно добирати додатково). Самі ж ці ядра являтимуть собою підмножини вершин, розфарбованих одним кольором $l=1,...,h$.

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 графу з вершинами-фахівцями d_i і застосування до нього алгоритму Візинга дозволяє отримати розфарбований граф з хроматичним числом 3, наведений на рис. 3 (вершини, приналежні до складу трьох формованих ядер, позначені відповідно чорним і білим кольорами та штрихуванням).

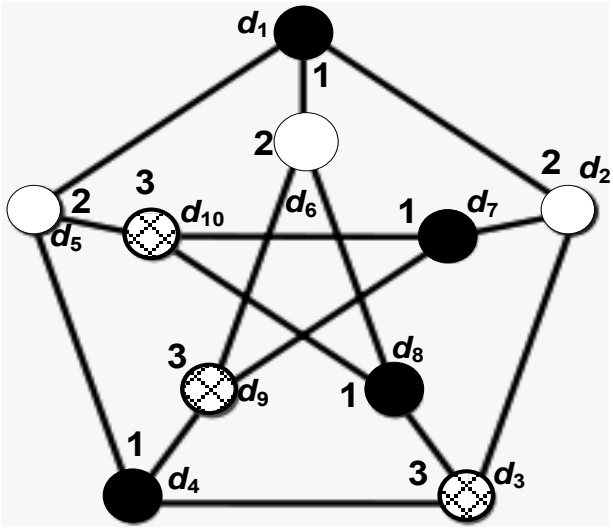

Рисунок 3 – Граф для побудови ядер незалежних груп, розфарбований фарбами 1-3

Слід зазначити, що це – відомий граф Петерсена [12], широко застосовний в 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

Запропоновані постановки задач булевого лінійного програмування для знаходження незалежної множини максимального розміру та заданої кількості попарно неперетинних незалежних множин, що мають максимальний сумарний розмір, і відповідні їм AMPL-коди можуть бути використані для автоматизованого отримання розв'язків тих задач керування програмним проектом, розв'язання яких недостатньо регламентовано в руслі технологічно-описового підходу до керування ним. Це насамперед задачі формування найбільш згуртованої команди програмного проекту; визначення множини програмних проектів у портфелі, які (не) можна виконувати одночасно, формування та аналізу під-команд для складного розподіленого програмного проекту. Через слабку формалізованість та істотну невизначеність, властиві програмному проекту, отримані розв'язки стають для них (суб)-оптимальними прототип ними рішеннями, ситуативно налаштовними в руслі технологічного підходу, а ретроспектива процесу налаштування надає підстави для «зворотного» вдосконалення висхідних постановок оптимізаційних задач.

Демонструючи перспективність вдосконалення програмного проекту за допомогою інструментарію дискретної оптимізації, наведені алгоритми висвітлюють доцільність узагальнення незалежної множини/кліки до так званих co-k-плексу/k-плексу, яке складає напрям подальших досліджень авторів.

Однак навіть наведені AMPL-коди та програма-солвер **gurobi** можуть бути використані в навчальних курсах з дисциплін «Управління програмними проектами» та «Оптимізаційні методи управління проектами» у вищих навчальних закладах України різних рівнів акредитації.

Література

1. Стандарт з управління проектами та настанова до Зводу знань з управління проектами (Настанова PMBOK). Сьоме видання. Newton Square, Pennsylvanya: Project management Institute, Inc, 2021. – 370 с. [Електронний ресурс]. – Режим доступу: https://pmiukraine.org/wp-content/uploads/2022/08/PMBOK7_Ukr_ForPersonalUseOnly.pdf.
2. Баркалов С.А. Умное управление проектами: учебное пособие / С.А. Баркалов, В.Н. Бурков, Я.Д. Гельруд и др.; под ред. чл.-корр. РАН Д. А. Новикова. Челябинск: Издательский центр ЮУрГУ, 2019. – 189 с.
3. Стецюк П.І. Максимальні незалежні множини вершин графа та їх застосування в керуванні проектами / П.І.Стецюк , О.О.Слабоспицька, О.О.Ушакова // Пит. прикладної математики і математичного моделювання. Д.: РВВ ДНУ, 2016. – Вип. 16. – С. 151–162.
4. Кристофидес Н. Теория графов. Алгоритмический подход / Н.Кристофидес. – М: Мир. 1978. – 432 с.
5. Гэри М. Вычислительные машины и труднорешаемые задачи. / Гэри М., Джонсон Д. – М.: Мир. 1982. – 416 с.
6. Визинг В.Г. К проблеме минимальной раскраски вершин графа / В.Г.Визинг, Г.С.Плесневич // Сибирский математический журнал. –1965. – Т.6. – № 1. – С. 234–236.
7. Стецюк П.И. Об ЛП-ориентированных верхних оценках для взвешенного числа устойчивости графа / П.И.Стецюк, А.П.Лиховид – Кибернетика и системный анализ. – 2009. – № 1. – С. 157–170.
8. Balansundaram B. Clique relaxations in social network analysis: the maximum k-plex problem / B.Balansundaram, S.Butenko, I.V.Hicks // Operations Research. – 2011. – V.59. – N 1. – P. 133–142.
9. Gurobi Optimization Inc.: Gurobi Optimizer Reference Manual.[Electronic resource]. – Mode of access: http://www.gurobi.com/documentation/
10. Fourer R. AMPL, A Modeling Language for Mathematical Programming / R. Fourer, D. Gay, B. Kernighan. – Belmont: Duxburry Press, 2003. – 517 p.
11. AMPL for research [Electronic resource]. – Mode of access: https://ampl.com/.
12. Нікольський Ю.В. Дискретна математика / Ю.В. Нікольський, В.В. Пасічник, Ю.М. Щербина. – К.: Видавнича група BHV, 2007. – 368 с.

References

1. A Guide to the Project Management Body of Knowledge (PMBOK Guide) – and the Standard for Project Management [7 ed.]. Newton Square, Pennsylvanya: Project management Institute, Inc, 2021. – 370 p.
2. Barkalov, S.A., Burkov V.N., Gelrud Ya.D. et al. (2019) Smart Project Management: A Manual. Chelyabinsk:YurGU Publishers. – 189 p.
3. Stetsyuk, P.I., Slabospitska O.O., Ushakova O.O. (2016) Maximal independent sets of graph vertices and their application in project management. Pytannya prykladnoyi matematyky i matematychnoho modelyuvannya. D.: RVV DNU. 16. P. 151-162 (In Ukrainian)
4. Christofides, N (1975) Graph Theory: An Algorithmic Approach. Academic Press Inc, Cambridge, Massachusetts, United States.
5. Garey, M. R.; Johnson, D. S. (1982) Computing machines and hard-to-solve tasks. M.: Mir. 1982. 416 p. (In Russian)
6. Vizing, V.G., Plesnevich, G.S. (1965) On the problem of minimal coloring of graph vertices. Sibirskij matematiceskij zurnal 6 (1) P. 234-236. (In Russian)
7. Stetsyuk, P.I., Lykhovyd, A.P. (2009) LP-oriented upper bounds for the weighted stability number of a graph. Cybern Syst Anal. 45. P.141–152.
8. Balansundaram, B., Butenko, S., Hicks, I.V. Clique relaxations in social network analysis: the maximum k-plex problem. Operations Research. 2011. Vol. 59, N 1. P. 133–142.
9. Gurobi Optimization Inc.: Gurobi Optimizer Reference Manual. http://www.gurobi.com/documentation/
10. Fourer, R., Gay D., Kernighan B. (2003) AMPL, A Modeling Language for Mathematical Programming. Belmont: Duxburry Press. 517 p.
11. AMPL for research [Electronic resource]: https://ampl.com/
12. Nikolskyi, Yu. V., Pasichnyk, V. V., Shcherbina, Yu. M. (2007) Dyskretna matematyka. 368 p. (In Ukrainian)

Одержано 17.08.2022

Про авторів:

Слабоспицька Ольга Олександрівна
старший науковий співробітник,
ст. наук. співроб. наукового відділу,
03143, Київ-143, б. Ак. Вернадського 69[а], кв.18,
кількість публікацій у вітчизняних виданнях – більше 50,
кількість у закордонних виданнях – 5, *h*-index: 5,
http://orcid.org/0000-0001-6556-0947

Стецюк Петро Іванович
старший науковий співробітник,
завідувач відділу,
02090, Київ, вул. Новаторів 22[в], кв.269
Кількість публікацій в українських виданнях – більше 100
Кількість зарубіжних публікацій – 50, *h*-index: 9 (Scopus),
https://orcid.org/0000-0003-4036-2543

Хом'як Ольга Миколаївна
старший науковий співробітник наукового відділу,
08133, м. Вишневе, вул. В. Чорновола, 48Б, кв.137.
Кількість публікацій в українських виданнях – 25
Кількість зарубіжних публікацій – 12.
h-index: 2 (Scopus),
https://orcid.org/0000-0002-5384-9070

Місце роботи авторів:

Слабоспицька Ольга Олександрівна
Інститут програмних систем
НАН України,
03187, Київ-187, Проспект Академіка Глушкова, 40.
Тел.: +38(044) 526 4286.
olsips2017@gmail.com

Стецюк Петро Іванович, Хом'як Ольга Миколаївна
Інститут кібернетики імені В.М. Глушкова НАН України,
м. Київ, 03187, Україна, Проспект Академіка Глушкова, 40
Тел. (+38) (044) 526-20-08 Факс (+38) (044) 526-41-78
stetsyukp@gmail.com, khomiak.olha@gmail.com

Прізвища та ініціали авторів і назва доповіді англійською мовою:
Slabospitska O.O., Stetsyuk P.I., Khomiak O.M.
Maximum independent sets of graph vertices searching
for software projects improvement

Прізвища та ініціали авторів і назва доповіді українською мовою:
Слабоспицька О.О., Стецюк П.І., Хомяк О.М.
Пошук максимальних незалежних множин вершин графа
для вдосконалення програмних проєктів

Контакти для редактора:
Слабоспицька Ольга Олександрівна,
093 219 29 30, olsips2017@gmail.com