УДК 519.17 https://doi.org/10.31713/vt1202222

Турбал Ю. В., д.т.н., професор (Національний університет водного господарства та природокористування, м. Рівне, y.v.turbal@nuwm.edu.ua), **Бабич С. В.** (Рівненський фаховий коледж національного університету біоресурсів і природокористування України, м. Рівне, asha.maneteren@gmail.com)

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ЗАДАЧ КАЛЕНДАРНОГО ПЛАНУВАННЯ НА ОСНОВІ КОНФІГУРАЦІЙНИХ ПІДХОДІВ

У статті розглянуто можливість застосування до розв'язку задач календарного планування алгоритмів, що ґрунтуються на процедурі декомпозиції спеціальним чином модифікованих перманент матриць інцидентності. Відповідні матриці інцидентності будуються на основі даних про розклад зустрічей, зокрема інформації про те, скільки та яких зустрічей мають проєктні менеджери з кожною проєктною командою протягом тижня. Для реалізації відповідних алгоритмів пропонується можливий вибір відповідних структур даних та функцій, зокрема рекурсивних, що дозволяють реалізувати відповідні алгоритми на основі структур даних, що пропонуються. Реалізація здійснюється за допомогою мови програмування С++. Результатом роботи відповідних функцій є генерація повної системи різних представників матриць розкладів. В роботі наведено приклад роботи відповідних функцій.

Ключові слова: **календарне планування; розклад; матриця розкладу; перманент; декомпозиція перманента; рекурсія; система різних представників.**

Задачі календарного планування останніми роками не втрачають свою актуальність, незважаючи на наявність великої кількості теоретичних результатів та практичних підходів до їх розв'язання. Це пов'язано з тим, що алгоритми побудови розкладів мають значну складність. Зокрема, на складність алгоритмів побудови розкладів (NP-повні задачі) вказував С. В. Давидов [1], запропонувавши вирішення методом наближеним до точного методу для певного критерію оптимальності, оскільки з «виконання задач побудови розкладів методом перебору, як і перебору з поверненням, неефективне» [2]. Тому в сучасних наукових працях віддається перевага евристичним

підходам, адже реальні розміри отриманих задач поки не дозволяють практично вирішити їх існуючими точними методами для задач квадратичного програмування. Результати, до котрих призводить те чи інше впорядкування, суттєво відрізняються. В ряді практичних випадків ці відмінності визначаються іншою величиною, або приймають вартісний характер.

В основі конфігураційного підходу, що пропонується як теоретична основа для розробки відповідного програмного забезпечення для задач календарного планування, лежать підходи, що ґрунтуються на поняттях конфігурацій. Зокрема для задачі складання розкладу навчальних занять конфігурації являють собою деякі множини пар, що читаються одним викладачем. При цьому виникають деякі поняття, таких як матриця розкладу, її допустимість, системи різних представників стовпчиків, матриця інцидентності конфігурацій, перманента з різними модифікаціями тощо. Ці поняття становлять основу алгоритмів оптимізації матриць розкладів.

Нехай маємо n проєктних команд $C_1, C_2, ..., C_n$ та m проєктних менеджерів $P_1, P_2, ..., P_m$, яким необхідно провести визначену кількість зустрічей протягом робочого тижня з окремими проєктними командами чи групами команд одночасно. Будемо розглядати певні часові щоденні індикатори зустрічей, які визначають можливий час початку зустрічі для будь-якого проєктного менеджера $t_1, t_2, ..., t_k$ з рахуванням особливостей робочого часу менеджерів. Тут вважатимемо час кожної зустрічі таким, що не перевищує певну константу. Необхідно сформувати розклад зустрічей менеджерів з робочими командами впродовж тижня.

Очевидно, що розклад зустрічей можна представити у вигляді матриці, в якій стовпчикам ставляться у відповідність проєктні команди, рядочкам–часові індикатори, а елементами матриці будуть ідентифікатори (номери) проєктних менеджерів. Нехай, наприклад, маємо довільну матрицю денного розкладу, розмірністю 3xn, що являє собою сукупність номерів проєктних менеджерів, які проводять відповідні зустрічі. Модифікована матриця інцидентності будується з врахуванням «потоків» (коли один проєктний менеджер зустрічається з декількома командами одночасно) наступним чином. Рядки матриці будуть помічатись ідентифікаторами, що відповідають проєктним командам (ідентифікаторами команд), стовпчики-ідентифікаторами, що відповідають проєктним менеджерам. Якщо

проєктний менеджер має можливість чи необхідність зустрітись одночасно з кількома командами, то утворюється окремий стовпчик, що описує таку ситуацію. На перетині рядка та стовпчика ставиться 1, якщо відповідний проєктний менеджер має зустріч з відповідною проєктною командою і 0, якщо ні.

Наприклад, для матриці розкладів виду

$$\begin{matrix} 1 & 1 & 1 & 2 & 3 & 5 \\ 2 & 3 & 1 & 4 & 4 & 4 \\ 3 & 1 & 4 & 5 & 2 & 4 \end{matrix} \qquad (1)$$

можемо побудувати наступну матрицю інцидентності конфігурацій стовпчиків:

$$\begin{array}{c} \\ R_1 \\ R_2 \\ R_3 \\ R_4 \\ R_5 \\ R_6 \end{array} \begin{pmatrix} 1 & 1^n & 2 & 3 & 4 & 4^n & 5 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}. \qquad (2)$$

Модифікації вимагає і поняття перманента матриці конфігурацій.

Означення. Модифікованим перманентом матриці інцидентності будемо називати суму всіх можливих добутків елементів матриці, в кожен з яких входить по одному елементу з кожного рядка та з різних стовпчиків, причому елемент потокового стовпчика (стовпчика, що відповідає потоковому елементу) не може бути в добутку разом з елементами інших рядків, що відповідають цьому ж потоку.

Очевидно, що у випадку відсутності потокових елементів модифікований перманент є звичайним перманентом.

Для підрахунку модифікованого перманента матриці інцидентності можемо використовувати алгоритм розкладу за рядком. Причому, виходячи з означення, процедура розкладу наступна. Ненульовий елемент рядка множиться на модифікований перманент матриці, утвореної за наступними правилами: якщо елемент рядка належить потоковому стовпчику, то матриця утворюється з вихідної викреслюванням стовпчика, де цей елемент стоїть, та всіх рядків, що відповідають всім елементам цього потоку. Якщо елемент рядка не належить потоковому стовпчику, то матриця утворюється шляхом викреслювання стовпчика, де стоїть цей елемент, та рядка, а також всіх потокових стовпчиків, на перетині яких з цим рядком стоять не-

нульові елементи. Відмітимо, що якщо викреслюються всі елементи, то результат вважається одиницею.

Запропонувавши процедуру розкладу модифікованого перманента за рядком, що враховує індексні елементи стовпчиків, можна побудувати коректну процедуру побудови системи різних представників стовпчиків (СРПС).

Приклад: нехай маємо матрицю розкладу

$$R = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 3 & 1 \\ 3 & 1 & 4 \end{pmatrix} \tag{3}$$

Тоді матриця інцидентності має вигляд

$$\begin{matrix} \\ 1 \\ 2 \\ 3 \end{matrix}\begin{pmatrix} 1 & 1^n & 2 & 3 & 4 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix}. \tag{4}$$

Побудуємо процес розкладу модифікованого перманента з запам'ятовуванням за першим рядком

$$per\mathrm{mod}\begin{matrix} \\ 1 \\ 2 \\ 3 \end{matrix}\begin{pmatrix} 1 & 1^n & 2 & 3 & 4 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix} = 1_1^{1^n} * 1 + 1_1^2 * per\mathrm{mod}\begin{matrix} \\ 2 \\ 3 \end{matrix}\begin{pmatrix} 1 & 3 & 4 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} +$$

$$+1_1^3 * per\mathrm{mod}\begin{matrix} \\ 2 \\ 3 \end{matrix}\begin{pmatrix} 1 & 2 & 4 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix} = 1_1^{1^n} * 1 +$$

$$1_1^2 * (1_2^1\, per\mathrm{mod}\begin{matrix} \\ 3 \end{matrix}\begin{pmatrix} 3 & 4 \\ 0 & 1 \end{pmatrix} + 1_2^3\, per\mathrm{mod}\begin{matrix} \\ 3 \end{matrix}\begin{pmatrix} 1 & 4 \\ 1 & 1 \end{pmatrix}) + 1_1^3 1_2^1\, per\mathrm{mod}\begin{matrix} \\ 3 \end{matrix}\begin{pmatrix} 2 & 4 \\ 0 & 1 \end{pmatrix} = 1_1^{1^n} +$$

$$+1_1^2 1_2^1 1_3^4 + 1_1^2 1_2^3 1_3^1 + 1_1^2 1_2^3 1_3^4 + 1_1^3 1_2^1 1_3^4 . \tag{5}$$

При недостатній кількості СРПС відносно кількості зустрічей, очевидно, побудова розкладу неможлива.

Бачимо, що в останньому прикладі модифікований перманент рівний 5 (і це є кількість всіх можливих СРПС). Крім того, відомі й самі СРПС. Вони записуються як верхні індекси одиничок. Причому, якщо певні нижні індекси відсутні, то відповідну кількість разів повторю-

ється верхній елемент (ситуація потокового елементу): 111, 214, 231, 234, 314.

При наявності всіх можливих СПРС, є допустимим вирішення задач різних класів. Наприклад, якщо нас цікавить допустима форма матриці розкладу, то вона в існує лише за умови, коли у множині всіх СПРС є такі три СПРС, що не містять в собі на однакових місцях однакових елементів, окрім елементів потоків.

Для даного прикладу існує тільки один набір СПРС: 111, 314, 231.

Таким чином, використання поняття модифікованого перманента дозволяє розробити ефективний алгоритм формування системи різних представників, яка, очевидно, є важливою при формуванні матриць розкладів.

Запропоновані алгоритм та структури даних.

Розв'язок поставленої задачі можна розбити на два етапи:

- Побудова множини всіх можливих СРПС.
- Вибір підмножини, що відповідає відповідному набору умов (критеріїв).

Для вирішення першого етапу розв'язку задачі можемо запропонувати наступний підхід. Варіанти СРПС будемо зберігати у списку, утвореному екземплярами класу SRP:

```
class SRP
{
        public:
        int nlast;
        int nmax;
        char *s;
        SRP *next;
        SRP()
        {
        s=new char [20];
                nlast=0;
        next=NULL;
        }
}
```

Для збереження розширеної матриці інцидентності (що містить номери рядків та елементи, які відповідають стовпчикам) будемо використовувати клас incydent, який і буде поступати на вхід рекурсивної функції генерації СРПС разом з елементом, який зумовив утво-

рення відповідної матриці інцидентності в розкладі (адресою відповідного елемента списку, утвореного екземплярами класу SRP)

```
class incydent
{
        public:
        int nmaxrow;
        int nmaxcol;
        int **r;
        incydent(int nrow, int ncol)
        {
        nmaxrow=nrow;
                nmaxcol=ncol;
                r=new int *[nmaxrow];
                for(int i=0;i<nmaxrow;i++)
        r[i]=new int [nmaxcol];
        }
        incydent(void){}
        ~incydent()
        {
        for(int i=0;i<nmaxrow;i++)
                delete [] r[i];
                delete [] r;
        }
};
```

Рекурсивна функція generic здійснює розклад за рядком з «запам'ятовуванням».

```
int generic(SRP *a, incydent *pI) //умова завершення рекурсії
{
                numb++; //підрахунок ітерацій рекурсії
                SRP *_current=head;
                // друк SRP
                if (pI->nmaxrow==0) return 1;
                if (pI->nmaxrow==1) return 1;
                SRP *current=new SRP;
                current=a;
        for(int j=0;j<pI->nmaxcol;j++)
                {
                if(pI->r[1][j]==1)
                {
```

```
                incydent *_pl=new incydent(pl->nmaxrow,pl->nmaxcol);
                        _pl->nmaxrow=pl->nmaxrow;
                        _pl->nmaxcol=pl->nmaxcol;
                        for(int x=0;x<_pl->nmaxrow;x++)
                                for(int z=0;z<_pl->nmaxcol;z++)
                                _pl->r[x][z]=pl->r[x][z];
                        if (_pl->r[0][j]<64)
                        {int j1=j;
                        for(int l=0;l<_pl->nmaxcol;l++)
                        {
                        if((_pl->r[1][l])&&(_pl->r[0][l]>=64))
                { for(int m=l;m<_pl->nmaxcol-1;m++)
                    for(int tmp=0;tmp<_pl->nmaxrow; tmp++)
                        _pl->r[tmp][m]=_pl->r[tmp][m+1];
                                _pl->nmaxcol--;
                                if(l<j1)
                                j1--;
                        }}
                                for(int m=j1;m<_pl->nmaxcol-1;m++)
                        for(int tmp=0;tmp<_pl->nmaxrow;tmp++)
                                _pl->r[tmp][m]=_pl->r[tmp][m+1];
                        _pl->nmaxcol--;
                        for(int d=1;d<_pl->nmaxrow-1;d++)
                        for(int tmp=0;tmp<_pl->nmaxcol;tmp++)
                                _pl->r[d][tmp]=_pl->r[d+1][tmp];
                        _pl->nmaxrow--;
                        }

if(_pl->r[0][j]>=64)
//блок коду перевірки поточних елементів, аналогічний наведеному
//вище
                        SRP *temp=new SRP;
                        for(int r=0;r<a->nlast;r++)
                                temp->s[r]=a->s[r];
                        temp->nlast=a->nlast;
                        if(pl->r[0][j]>=64)
                                for(int q=1; q<pl->nmaxrow; q++)
                                if(pl->r[q][j]==1)
                                        temp->s[temp->nlast++]=pl->r[0][j];
```

```
                        if(pl->r[0][j]<64)
                        temp->s[temp->nlast++]=pl->r[0][j];
                         temp->next=current->next;
                        current->next=temp;
//друк матриці після змін
                        generic (temp, _pl);
                }
                }
        }
}
```

Приклад роботи такої функції наведено на рисунку.

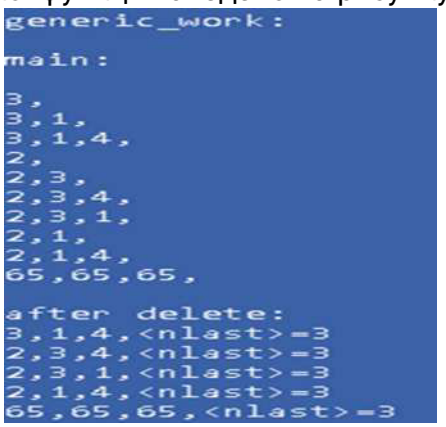

Рисунок. Приклад роботи функції генерації СРПС для матриці (3)

Таким чином, в роботі розглянуто можливу програмну реалізацію на С++ перманентного алгоритму генерації систем різних представників стовпчиків матриці інцидентності, що будується на основі матриці розкладів (системи вимог до розкладу). Відповідний підхід може бути покладений в основі загальної системи складання розкладу зустрічей, занять в ЗВО тощо, що має важливе практичне значення. Перманентний підхід загалом може бути покладений в основу алгоритмів генерації комбінаторних об'єктів в широкому розумінні, а тому має гарні перспективи для подальшого його використання. Відповідна методика має низку суттєвих переваг у порівнянні з існуючими підходами, зокрема певну універсальність. Адже за допомогою перманентних алгоритмів можна генерувати як прості комбінаторні об'єкти, наприклад перестановки, так і складні розклади, що мають низку додаткових вимог до їх формування.

1. Давыдов С. В. Система автоматического построения расписания учебных зянятий. URL: http://davidovsv narod.ru/schedule/index.html. (дата звернення: 10.06.2022). **2.** Сервах В. В., Щербіна Т. А. Алгоритмы решения задач календарного планирования с различными критериями. *Методы оптимизации и их приложения* : материалы 14 Байкалской международной школы-семинара. Северобайкальск, 2008. Т. 1. С. 506–513. **3.** Попов Г. О. Формализация задачи составления учебного расписания в высшем учебном заведении. *Вестник Астраханского государственного технического университета*. 2006. № 1. С. 120. **4.** Бабич С. В., Турбал Ю. В. Алгоритм побудови допустимої матриці розкладів. *XXV International conference PDMU 2015, Skhidnytsia, Ukraine* : Abstracts. С. 60–61. **5.** A. Mingozzi, V. Maniezzo, S. Ricciardelli, and L. Bianco. An exact algorithm for the resource-constrained project scheduling problem. *Management Science*. 1998. Vol. 44, Pp. 715–729. **6.** P. Brucker, S. Knust, A. Schoo, and O. Thiele. A branch & bound algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research*. 1998. Vol. 107, Pp. 272–288. **7.** R. Lepere, D. Trystram, and G. J. Woeginger. Approximation algorithms for scheduling malleable tasks under precedence constraints. *Int. J. Found. Comput. Sci*. 2002. Vol. 13, no. 4. Pp. 613–627. **8.** Vanhoucke E. Demeulemeester, and W. Herroelen. An exact procedure for the resource-constrained weighted earliness-tardiness project scheduling problem. *Annals of Operations Research*. 2001. Vol. 102, Pp. 179–196.

REFERENCES:

1. Davyidov S. V. Sistema avtomaticheskogo postroeniya raspisaniya uchebnyih zyanyatiy. URL: http://davidovsv narod.ru/schedule/index.html. (data zvernennia: 10.06.2022). **2.** Servah V. V., Scherbina T. A. Algoritmyi resheniya zadach kalendarnogo planirovaniya s razlichnyimi kriteriyami. *Metodyi optimizatsii i ih prilojeniya* : materialyi 14 Baykalskoy mejdunarodnoy shkolyi-seminara. Severobaykalsk, 2008. T. 1. S. 506–513. **3.** Popov G. O. Formalizatsiya zadachi sostavleniya uchebnogo raspisaniya v vyisshem uchebnom zavedenii. *Vestnik Astrahanskogo gosudarstvennogo tehnicheskogo universiteta*. 2006. № 1. S. 120. **4.** Babych S. V., Turbal Yu. V. Alhorytm pobudovy dopustymoi matrytsi rozkladiv. *XXV International conference PDMU 2015, Skhidnytsia, Ukraine* : Abstracts. S. 60–61. **5.** A. Mingozzi, V. Maniezzo, S. Ricciardelli, and L. Bianco. An exact algorithm for the resource-constrained project scheduling problem. *Management Science*. 1998. Vol. 44, Pp. 715–729. **6.** P. Brucker, S. Knust, A. Schoo, and O. Thiele. A branch & bound algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research*. 1998. Vol. 107, Pp. 272–288. **7.** R. Lepere, D. Trystram, and G. J. Woeginger. Approximation algorithms for scheduling malleable tasks un-

der precedence constraints. *Int. J. Found. Comput. Sci*. 2002. Vol. 13, no. 4. Pp. 613–627. **8.** Vanhoucke E. Demeulemeester, and W. Herroelen. An exact procedure for the resource-constrained weighted earliness-tardiness project scheduling problem. *Annals of Operations Research*. 2001. Vol. 102, Pp. 179–196.

Turbal Yu. V., Doctor of Engineering, Professor (National University of Water and Environmental Engineering, Rivne), **Babych S. V.** (Rivne Professional College of National University of Life Environmental Sciences of Ukraine, Rivne)

SOFTWARE FOR CALENDAR PLANNING TASKS BASED ON CONFIGURATION APPROACHES

The article notes the possibility of applying algorithms based on the decomposition procedure of specially modified permanent incidence matrices to the solution of calendar planning problems. The relevant incident matrices are built based on meeting schedule data, including information on how many and which meetings project managers have with the skinned project team during the week. For the implementation of the corresponding algorithms, a possible selection of the appropriate data structures and functions is offered, in particular, recursive ones, which do not allow the implementation of the corresponding algorithms based on the proposed data structures. Implementation of creation using the C++ programming language. The result of the corresponding functions is the generation of a complete system of various representatives of the schedule matrix. An example of the operation of the corresponding functions is given in the work.

***Keywords*: calendar planning; schedule; schedule matrix; permanent; permanent decomposition; recursion; system of different representatives.**
