УДК 519.1

Гришанович Тетяна Олександрівна

Київський національний університет ім. Т.Шевченка, факультет кібернетики

СКЛАДНІСТЬ АЛГОРИТМІВ РОЗКЛАДАННЯ ГРАФІВ

Робота присвячена дослідженню часової складності алгоритмів декомпозиції графів за їх вершинами в залежності від способу подання графа. Розглянуто точні та наближені алгоритми.

Работа посвящена исследованию временной сложности алгоритмов декомпозиции графов по их вершинами. Рассмотрено точные и приближенные алгоритмы.

This work is dedicated to the analysis of the time complexity of the graphs decomposition algorithms. The exact and drawn near algorithms are considered.

Ключові слова: алгоритми розкладання графів, часова складність, оцінка складності алгоритму.

Вступ. Розвиток теорії дискретних та комбінаторних задач та практика їх розв'язання показали, що загальність методу розв'язання та його ефективність знаходяться у досить значному протиріччі. Тому виникає наступне питання: розбивати задачі на вузькі класи та, користуючись їх специфікою, розробляти для них ефективні алгоритми чи орієнтуватись на загальні, але у свою чергу менш ефективні алгоритми? [2]

У найширшому значенні поняття ефективності алгоритму розв'язання задачі пов'язане із обчислювальними ресурсами. Але найчастіше під найефективнішим алгоритмом розуміють найшвидший, оскільки обмеження в часі досить часто є домінуючим фактором, що визначає придатність його для використання на практиці. Визначення часової оцінки роботи алгоритму називають апріорним аналізом алгоритму. Цей аналіз ігнорує усі фактори, що залежать від конкретної обчислювальної платформи (розрахунки здійснюються на РАМ – машині, що не передбачає паралельного виконання операцій [8]), мови програмування, та концентрується на визначенні типу функції, яка характеризує час виконання алгоритму. [3]

Час роботи алгоритму виражається у вигляді функції від однієї змінної, яка характеризує розмір індивідуальної задачі, тобто об'єм вхідних даних, що необхідні для опису цієї задачі. Вхідна довжина індивідуальної задачі визначається як число символів у ланцюжку, що отримується застосуванням до конкретної задачі схеми кодування для масової задачі. Саме це число, вхідна довжина, і використовується в якості формальної характеристики розміру індивідуальної задачі. Спосіб вимірювання розміру входу залежить від конкретної задачі – це може бути число елементів на вході, число біт, необхідне для подання усіх вхідних даних. В окремих випадках, розмір входу визначається кількома числами. Наприклад, число вершин та число ребер графа [8]

Якщо при заданому розмірі входу в якості міри складності береться найбільша із складностей (за всіма входами даного розміру), то її називають складністю у найгіршому випадку. Якщо ж вибирається середня складність алгоритму за даним розміром входу, то її називають середньою або усередненою. Цей тип часової складності алгоритму знайти важче, ніж складність у найгіршому випадку. [3]

Постановка задачі та метод її розв'язування. У даній роботі досліджуються часові складності алгоритмів розкладання графів за їх вершинами в залежності їх від способу подання графа. Зокрема, розглядаються точні алгоритми, тобто такі, які гарантують відшукання оптимального розфарбування вершин та хроматичного числа довільного графа та наближені, які не завжди знаходять точний розв'язок, але є більш ефективними.

Часом роботи алгоритму називатимемо число елементарних кроків, які він виконує. Вважатимемо, що елементарним кроком є один пункт алгоритму, але у тому випадку, коли він не містить вказівок такого роду, як «відсортувати масив за зростанням».[7] Насамперед нас буде цікавити час роботи T таких алгоритмів в найгіршому випадку з огляду на наступне:

- Знаючи час роботи у найгіршому випадку, можна гарантувати, що виконання алгоритму закінчиться за деякий час, не знаючи, вхід якого розміру буде використовуватись.
- На практиці «погані» входи (для яких час роботи близький до максимального) можуть траплятись досить часто.
- Середній час роботи може бути дуже близьким до часу роботи у найгіршому випадку. [4]

Структури даних, які використано для подання графів, – це матриці інцидентності та суміжності, списки суміжності та двійковий код графа. [3]

Аналіз отриманих результатів. Спочатку розглянемо наближені алгоритми розкладання, а саме: наближений та покращений алгоритми послідовного розфарбування вершин графа**.**

Алгоритм послідовного розфарбування. Довільній вершині v_1 графа G припишемо колір 1. Якщо вершини $v_1, v_2, \ldots v_i$ розфарбовані k кольорами,

k≤i, то новій довільно вибраній вершині v_{i+1} припишемо мінімальний колір, що не використовувався при побудові розфарбування суміжних вершин. [10]

for v∈ V **do**

C[v]:=0 {усі вершини не пофарбовані}

end for

for v∈ V **do**

A:={1, …p} {усі кольори}

for u∈ Γ^+ **do**

A:=A\{C[u]} {зайняті усі кольори}

end for

C[v]:=min A {мінімальний вільний колір}

end for

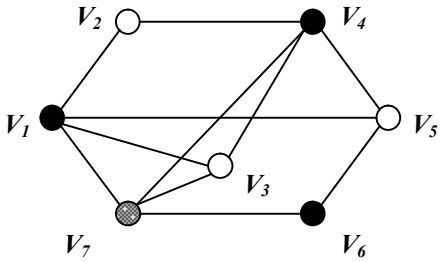


Рис. 1. «Розкладання графа за допомогою алгоритму послідовного розфарбування»

Знайдемо часову складність даного алгоритму, якщо граф G задається матрицею суміжності. Нагадаємо, що у такому випадку вершини графа G(V, E) нумерують числами 1, 2, … |V| та розглядається матрицю A=(a_{ij}) розміру |V| × |V|, для якої

$$a_{ij} = \begin{cases} 1, ÿêùî \ \ (i,j) \in E \\ 0, ÿêùî \ \ (i,j) \notin E \end{cases}$$

1. Задати вхідну послідовність із n^2 елементів, де n – кількість вершин графа.
2. Вибрати довільним чином вершину. Наприклад, v_1.

Зауваження: Для зберігання розфарбування графа використаємо масив C, де номер елемента вказуватиме на номер вершини, а його вміст – на колір цієї вершини.

3. Для вибраної вершини v_1 приписуємо колір 1: C[1]:=1.
4. Перейти до наступної вершини, у даному випадку – v_2.
5. Переглядаємо усі суміжні v_2 вершини. Їх буде щонайбільше (n – 1).
6. У масиві C серед елементів з відповідними номерами знаходимо мінімальний елемент. Оскільки ми оцінюємо час роботи алгоритму у найгіршому випадку, то припустимо, що v_2 суміжна з усіма іншими вершинами графа. При пошуку мінімального кольору min переглянемо (n – 1) елемент.
7. Вершині v_2 приписуємо колір (min+1): C[2]:=min+1.
8. Перейти до кроку 2.

$$T_1(n)= n^2+(n-1)((n-1)+n) = 3n^2 - 3n + 1.$$

Теорема: Якщо $A(k) = a_m k^m + ... + a_1 k^1 + a_0$ є поліномом степеня m, тоді (*)

$$A(k)=O(k^m).$$

Отже, часова складність вище описаного алгоритму становить $O(n^3)$.

Якщо граф G, який має n вершин та e ребер, задається матрицею інцидентності, то розмірність матриці становить n×e. Матриця B=(b_{ij}) формується із нулів та одиниць за наступним правилом [2]:

$$b_{ij} = \begin{cases} 1, ÿêùî \ \ v_i \ ³íöèäåíòíà \ \ ðåáðó \ e_j \\ 0, ó \ ïðîòèëåæ \ \ íîìó \ âèïàä \ \ êó \end{cases}$$

В цьому випадку кількість елементарних операцій при виконанні послідовного алгоритму розфарбування графа становитиме $T_2(n)=2mn - m + 1$, а отже $T_2(n)$ – лінійна функція складності.

Наступний спосіб подання графа – двійковий код. Використовуючи матрицю суміжності графа, можна подати його у вигляді двійкового коду. Очевидно, якщо запам’ятати послідовність нулів та одиниць матриці, то за нею можна відновити весь граф. Так як матриця симетрична, то достатньо запам’ятати лише ті елементи, які розміщені над головною діагоналлю,

тобто послідовність a_{12}, a_{13}, a_{14}, …$a_{n-1\ n}$. Довжина цієї послідовності рівна $C_n^2 = \frac{n(n-1)}{2}$.

Число, яке отримаємо після виконання наступної операції:

$$a_{1,2} \cdot 2^0 + a_{1,3} \cdot 2^1 + a_{2,3} \cdot 2^2 + a_{1,4} \cdot 2^3 + ... + a_{n-1,n} \cdot 2^{c_n^2 - 1},$$

назвемо двійковим кодом матриці. Оскільки різним нумерація вершин графа відповідають різні матриці суміжності, то існує n! таких кодів. Найменший із цих кодів називають міні-кодом μ (G), а найбільший – максі-кодом μ (G).[2]

Таким чином, за умови подання графа за допомогою двійкового коду після переведення максі-коду μ (G) у двійкове число ми отримаємо матрицю суміжності графа. Оскільки виконання алгоритму за умови подання у вигляді матриці суміжності уже описано, то спробуємо визначити кількість операцій при перетворенні максі-коду μ (G) у двійкове число. Їх кількість становитиме $\frac{n(n-1)}{2}$.

Таким чином часова складність алгоритму послідовного розкладання становитиме $O(n^3)$.

Одним із поширених способів подання графів є списки суміжності. Цей спосіб доцільно використовувати у випадку розріджених графів (|E| значно менше від $|V^2|$). Подання графів у вигляді списків суміжності використовує масив *Adj* із |V| списків – по одному на вершину. Для кожної вершини v ∈ V список суміжних вершин *Adj[v]* містить у довільному порядку (вказівники на) усі суміжні з нею вершини. Підрахунок кількості елементарних операцій, які здійснюються в ході виконання алгоритму за умови такого способу подання приводить до наступної оцінки складності:

$$T_4(n)=n^3+2n^2+8n+3lg\ n+24m+1.$$

Застосувавши теорему (*), отримали $O(n^4+3\lg n)$.

Покращений алгоритм послідовного розфарбування. Наступний алгоритм також будує допустиме розфарбування. Але на відміну від попереднього, розфарбування розпочинається із вершини з найбільшим степенем. Якщо ж їх розфарбовувати в останню чергу, то може виявитись, що для них немає вільного кольору. [10]

```
Sort(v) {впорядкувати вершини за не зростанням степенів}
c:=1
for v ∈ V do
        C[v]:=0 {усі вершини не зафарбовані}
end for
while V≠ ø do
      for v ∈ V do
        for u ∈ Γ+(v) do
              if C[u]=c then
                    next for v {вершину не можна пофарбувати у колір c}
              end if
        end for
        c[v]:=c {зафарбовуємо вершину у колір c}
        V:=V\{v} {видаляємо вершину із переліку тих, які будуть розглядатись}
  end for
c:=c+1
end while
```

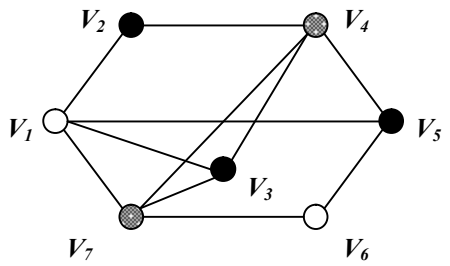


Рис. 2 «Розкладання графа за допомогою покращеного алгоритму послідовного розфарбування»

Оцінка часової складності проводиться методом, аналогічним до попереднього. Особливість даного алгоритму полягає в тому, що у ньому використовується сортування степенів вершин не по зростанню. Для його реалізації використано метод бульбашки, який використовує n^2 операцій, де n – кількість елементів, які сортують. [7]

Оцінки часової складності покращеного алгоритму розфарбування вершин графа наступні:

Матриця суміжності - $O(n^4)$.

Матриця інцидентності - $O(n^4)$.

Двійковий код - $O(n^4)$.

Списки суміжності - $O(n^4+3\lg n)$.

Щодо точних алгоритмів розфарбування, то вони, на відміну від наближених, гарантують відшукання оптимального розфарбування вершин та істинного значення хроматичного числа довільного графа. [10]

Точний алгоритм розфарбування. Даний алгоритм базується на використанні незалежних множин для розкладання графа за його вершинами. Задача відшукання найбільшої незалежної множини вершин графа належить до класу трудомістких задач.

Основна ідея алгоритму полягає у використанні рекурсивного виклику процедури (у даному випадку вона носить назву P), що знаходить максимальну незалежну множину вершин графа та зафарбовує їх у допустимий колір. [10] Множину вершин графа називають незалежною, якщо ніякі дві вершини у ній не є суміжними. [1]

if V=ø **then**

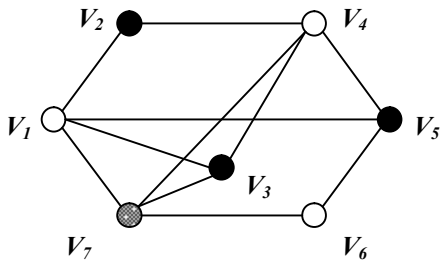
return {розфарбування закінчено}

end if

s:=selectmax (G) {S – максимальна незалежна множина}

C[S]:=i {зафарбовуємо вершини множини у колір i}

P(G-S, i+1) {рекурсивний виклик}



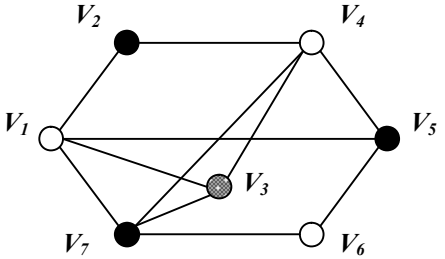


Рис. 3 «Розкладання графа за допомогою покращеного алгоритму послідовного розфарбування»

Для відшукання незалежних множин графа використаємо уже відомий алгоритм [10]. Побудова максимальної незалежної множини розпочинається із порожньої множини і в ході виконання алгоритму поповнюється вершинами із зберіганням незалежності.

k:=0 {кількість елементів у поточній незалежній множині}

S[k]:=ø {незалежна множина із k вершин}

$Q^-[k]$:=ø {множина вершин, що уже використані для розширення S[k]}

$Q^+[k]$:=V { множина вершин, які можна використати для розширення S[k]}

M1: {крок вперед}

select $v \in Q^+[k]$ {розширяюча вершина}

$S[k+1]:=S[k] \cup v$ {розширена множина}

$Q^-[k+1]:= Q^-[k] \setminus \Gamma[v]$ {вершина v використана для розширення}

$Q^+[k+1]:= Q^+[k] \setminus (\Gamma[v] \cup \{v\})$ {усі вершини, суміжні з v не можуть бути використані для розширення}

k:=k+1

M2:

for u ∈ $Q^-[k]$ **do**

if Γ[u] ∩ $Q^+[k]$= ø **then**

goto M3 {можна повертатись}

end if

end for

if $Q^+[k]$:= ø **then**

if $Q^-[k]$:= ø **then**

yield S[k] {множина S[k] максимальна}

end if

goto M3 {можна повертатись}

else goto M1 {можна іти вперед}

end if

M3: {крок назад}

v:=last(S[k]) {останній добавлений елемент}

k:=k-1

S[k]:=S[k+1] – {*v*}

$Q^-[k]$:= $Q^-[k]$ ∪ {*v*}

$Q^+[k]$:= $Q^+[k]$ \ {*v*} {вершина *v* уже додавалась}

if k=0 **and** $Q^+[k]$= ø **then**

stop {перебір завершено}

else goto M2 {перехід на перевірку}

enf if

Часову складність даного алгоритму оцінюватимемо разом із усіма операціями у алгоритмі.

Отже, часова оцінка точного алгоритму розкладання графа становить:

Матриця суміжності - $O(n^n)$.

Матриця інцидентності - $O(n^n)$.

Двійковий код - $O(n^n)$.

Списки суміжності - $O(n^4+8m+2mlg\ n)$.

Алгоритм, що базується на використанні максимальних r-підграфів. Породжений підграф $<S_r[G]>$ графа G=(X, Γ), де $S_r[G] \subseteq X$, називають *r*-підграфом, якщо він *r*-хроматичний. Якщо не існує такої множини H, що $H \supset S_r[G]$ і підграф H є *r*-хроматичним, то $<S_r[G]>$ називають максимальним *r*-підграфом графа G.

Хроматичне число графа є найменшим значенням r, при якому $S_r[G]$ – множина вершин деякого максимального *r*-підграфа – співпадає із множиною вершин X графа. Тому процедури послідовної побудови *r*-підграфів можна використовувати для знаходження хроматичного індексу графа, якщо на кожному кроці процедури перевіряти, чи не міститься множина вершин графа у котромусь із знайдених *r*-підграфів. [9]

r:=1

Q:={$S^j_r[G]$ | j=1, … q_r} {знаходимо максимальну множину r-підграфів}

label

k:=Selectmax(X - $S^j_r[G]$); {знаходимо максимальну незалежну множину}

if k≠ø **then**

S:= $S^j_r[G] \cup S_1[G]$

if S=X **then stop** {(r+1) – хроматичне число, розфарбування знайдено}

else

case S **of**

$S \subseteq S'$ **then goto label** { $S' \in Q$}

$S \supset S'$ **then Q:=Q – {S}** { $S' \in Q$}

else Q:=Q ∪ {S} **goto** label

end case

end if

else

if $j<q_r$ **then** j:=l+1 **goto** label

end if

if j=q_r **then** j:=l+1

r:=r+1

q_r:=|Q| **goto** label

end if

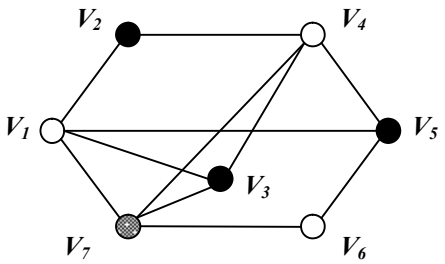


Рис. 4 «Розкладання графа за допомогою покращеного алгоритму послідовного розфарбування»

Для відшукання незалежних множин графа та його максимальних *r*-підграфів використано ту ж процедуру, що і для точного алгоритму. Часова складність алгоритму розкладання графа за допомогою його максимальних *r*-підграфів становить:

Матриця суміжності - $O(n^n+ n^2+c_1n+c_2)$, де c_1, c_2 – сталі.

Матриця інцидентності - $O(n^n+nm+nc_1+c_2)$, де c_1, c_2 – сталі.

Двійковий код - $O(n^n+ n^2+c_1n+c_2)$, де c_1, c_2 – сталі.

Списки суміжності - $O(n^n+ c_1n+8m +2m \lg n)$, де c_1– стала.

Слід зауважити, що останні два алгоритми використовують для побудови максимальних незалежних множин вершин алгоритм, який має переборний характер. Таким чином кожен із цих алгоритмів в тій чи іншій мірі все ж містить перебір варіантів.[10]

Далі розглянемо ***алгоритм розкладання графа за допомогою його кістяків*** [5] та спробуємо, як і вище, оцінити його часову складність в залежності від способу подання.

Ідея алгоритму полягає в наступному: для запропонованого графа будується кістяк. Він перетворюється в ході розв'язання задачі до тих пір, поки усі його вершини не задовольнятимуть відношення часткового порядку «<», що визначений на множині вершин. (Відношення часткового порядку «<» визначається наступним чином: вершина y < вершини z тоді і тільки тоді, коли найкоротший шлях від кореневої вершини x до вершини z проходить через вершину y.) Після того, коли такий кістяк знайдено, будується розфарбування самого графа. Обґрунтування даного методу міститься у [12].

Наближений алгоритм даного способу розкладання має такий вигляд.

Tr:=kraskala(G) {будуємо кістяк графа G за алгоритмом Краскала (можна вибрати довільний інший алгоритм)}

V_0:=SelectRoot {вибираємо кореневу вершину}

kistjak:=false;

repeat

BEGIN

{ побудова допоміжного масиву norm для визначення того, чи кістяк нормальний}

dejkstr(nn, ks, x, i, l1, k1, path); // процедура, що реалізує алгоритм Дейкстри відшукання мінімальної відстані між вершинами; x, i – номери вершин, відстань між якими обчислюється; l1 – допоміжна змінна; k1 – відстань між вершинами x та i; path – масив, що містить шлях від x до i.

dejkstr(nn, ks, x, j, l2, k2, path1);

if path1[p]=i **then**

norm[h].b11:=true **else** norm[h].b11:=false;

{ітерація виправлення кістяка на нормальний; даний блок команд повторюється доки масив norm не міститиме жодного елемента b11=false }

if norm[i].b11 = false **then**

dejkstr(nn, ks, x, norm[i].a11, l1, k1, path);

dejkstr(nn, ks, x, norm[i].a12, l2, k2, path1);

if k1<=K2 **then**

j:=norm[i].a12;

dejkstr(nn, ks, x, p, l1, k3, path);

```
            if (matrix[p, j]<>0) and (matrix[p, j]<>1000) and (ks[p, j]<>0)
and(k3<=k1-1) then
    // видаляємо ребро (p, j)
                    ks[norm[i].a11, norm[i].a12]:=1;
                    ks[norm[i].a12, norm[i].a11]:=1;
                    norm[i].b11:=true;
                  else
                j:=norm[i].a12;
                dejkstr(nn, ks, x, p, l1, k3, path);
                if (matrix[p, j]<>0) and (matrix[p, j]<>1000) and (ks[p, j]<>0)
                and(k3<=k1-1) then
    // видаляємо ребро (p, j)
                    ks[norm[i].a11, norm[i].a12]:=1; // додаємо нове ребро
                    ks[norm[i].a12, norm[i].a11]:=1;
                    norm[i].b11:=true;
               {перевіримо, чи масив norm містить значення false}
               if norm[i].b11=false then kistjak:=false;
      until kistjak=false;
        {розфарбування отриманого нормального кореневого кістяка усіма
можливими кольорами}
      hi[i]:= (k1)mod a;
    end.
```

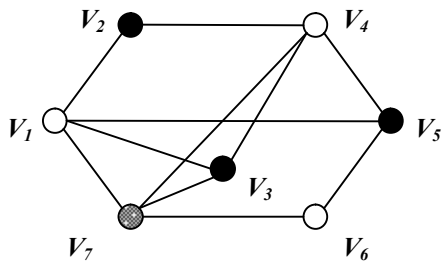


Рис. 5 «Розфарбування графа за допомогою алгоритму розкладання графа за допомогою його кістяків»

Оцінки часової складності описаного алгоритму наступні [6, 11]:

Матриця суміжності - $O(n^4)$.

Матриця інцидентності - $O(n^4+m \log m)$.

Двійковий код - $O(n^4)$.

Списки суміжності - $O((2n + 8m + 2m[\log_{10} n])^3)$.

Далі в таблиці подані оцінки часових складностей алгоритму розкладання графів.

АЛГОРИТМ	**СПОСІБ ПОДАННЯ**			
	Матриця суміжності	**Матриця інцидентності**	**Двійковий код**	**Списки суміжності**
1	**2**	**3**	**4**	**5**
Алгоритм послідовного розфарбування	$O(n^3)$	лінійна	$O(n^3)$	$O(n^2+n \lg n)$
1	**2**	**3**	**4**	**5**
Покращений алгоритм послідовного розфарбування	$O(n^4)$	$O(n^4)$	$O(n^4)$	$O(n^4+3\lg n)$
Точний алгоритм розфарбування	$O(n^n)$	$O(n^n)$	$O(n^n)$	$O(n^4+8m+$ $+2m\lg n)$

Алгоритм розфарбування з використанням максимальних *r*-підграфів	$O(n^n+n^2+ +nc_1+c_2)$	$O(n^n+nm+nc_1+ + c_2)$	$O(n^n+n^2+nc_1+c_2)$	$O(n^n+ +c_1n+8m+ +2m \lg n)$
Алгоритм розфарбування за допомогою кістяків	$O(n^4)$	$O(n^4)$	$O(n^4)$	$O((2n + 8m+ +2m[\log_{10} n])^3)$

Таблиця

«Складність алгоритмів розкладання графів»

Висновки: Оцінки часової складності алгоритмів розкладання графів в залежності від способів подання графів свідчать про те, що запропонований алгоритм забезпечує не гіршу, а для точних алгоритмів навіть і кращу, часову складність. Крім того, час роботи кожного із алгоритмів повністю залежить від кількості вершин і лише в кількох випадках (алгоритм розкладання за допомогою кістяків та максимальних *r*-підграфів) від кількості ребер графа. Отже, за умови подання графа за допомогою списків суміжності такий алгоритм доцільно використовувати для розріджених графів – у яких кількість вершин значно перевищує кількість ребер. Хоча для випадку алгоритму розкладання, що базується на *r*-підграфах, величини 2m, nm та 8m (n – кількість вершин графа, m – кількість його ребер) не можуть суттєво впливати на ситуацію, оскільки яким би великим не було число ребер графа, його значно перевищуватиме число n^n.

Бібліографічні посилання:

1. **Асанов М.О., Баранский В.А., Расин В.В.** Дискретная математика: графы, матроиды, алгоритмы. – М.: РХД. 2001.
2. **Асельдеров З.М.** Представление и восстановление графов. – К.: Наукова думка, 1991.
3. **Ахо А., Хопкрофт Дж., Ульман Дж.** Построение и анализ вычислительных алгоритмов.- М.:Мир, 1979.
4. **Ахо А., Хопкрофт Дж., Ульман Дж**. Структуры данных и алгоритмы. – М.: издательский дом «Вильямс», 2000.
5. **Гришанович Т.О.** «Алгоритм розкладання графів». ІІІ-я Міжнародна науково практична конференція «Моделювання та комп'ютерна графіка 2009», Донецьк.
6. **Дегтярьов В.О.** Розробка методології прокладки комунікацій в САПР для складних технічних об'єктів. //Вестник ХНТУ. – 2005. – №1(21). – С. 190-195.
7. **Катленд Н.** Вычислимость. Введение в теорию рекурсивных функций. – М.: Мир, 1990.
8. **Кормен Т.** Алгоритмы: построение и анализ./Т.Кормен, Ч.Лайзерсон, Р.Риверст.М.:МЦНМО: БИНОМ.Лаборатория знаний, 2004
9. **Кристофидес Н.** Теория графов. Алгоритмический поход. – М.: Мир, 1978.
10. **Новиков Ф.А.** Дискретная математика для программистов. – С.Пб.: Питер, 2000.
11. **Погорілий С.Д., Бойко Ю.В., Білоус Р.В.** Формування та аналіз паралельних схем алгоритму Дейкстри. // Математичні машини і системи. – 2008. – №4. – С.59-65.
12. **Протасов І.В., Протасова К.Д.** Розкладність графів: Навчальний посібник. – Видавничо – поліграфічний центр «Київський університет», 2003.