МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
(повне найменування назва факультету)

Кафедра прикладної математики
(повна назва кафедри)

Магістерська робота

Узагальнення задачі про максимальний потік та його застосування

Виконав: студент групи ПМПМ-22с
спеціальності:
113 «Прикладна математика»
(шифр і назва спеціальності)

Жук О.-А. С.
(підпис) (прізвище та ініціали)

Керівник Білецький В. М.
(підпис) (прізвище та ініціали)

Рецензент
(підпис) (прізвище та ініціали)

Львів – 2022

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики

Кафедра прикладної математики

Спеціальність 113 «Прикладна математика»

(шифр і назва)

«ЗАТВЕРДЖУЮ»

Завідувач кафедри _____

" ___ " _____ 20 __ року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Жуку Олегу-Андрію Степановичу

(прізвище, ім'я, по батькові)

1 .Тема роботи Узагальнення задачі про максимальний потік та його застосування

керівник роботи к. ф.-м. н., доцент Білецький Василь Миколайович ,

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені Вченою радою факультету від "22" вересня 2021 року № 6

2. Строк подання студентом роботи 16 травня 2022 р.

3. Вихідні дані до роботи Дослідження узагальненої задачі про знаходження максимального потоку в мережі. Дослідження алгоритмів знаходження максимального потоку. Дослідження алгоритму скасування циклів, що генерують потік. Програмна реалізація досліджених алгоритмів. Розробка застосунку для демонстрації роботи алгоритму.

4. Зміст магістерської роботи (перелік питань, які потрібно розробити):
Вступ;
Потокова мережа;
Алгоритми розв’язку задачі про максимальний потік;
Програмна реалізація;
Висновки;

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Графічний матеріал відсутній.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 02.09.2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Дослідження потокових мереж та задачі знаходження максимального потоку в мережі.	02.09.2021-11.10.2021	
2	Дослідження узагальнення задачі про максимальний потік в мережі.	11.10.2021-01.11.2021	
3	Дослідження алгоритмів знаходження максимального потоку в мережі.	01.11.2021-17.12.2021	
4	Програмна реалізація та прикладне застосування досліджуваних алгоритмів.	01.02.2022-18.03.2022	
5	Розробка веб-застосунку для демонстрації роботи алгоритму.	18.03.2022-15.04.2022	
6	Оформлення звіту до дипломної роботи.	04.04.2022-13.05.2022	

Студент Жук О.-А. С.
(підпис) (прізвище та ініціали)

Керівник роботи Білецький В. М.
(підпис) (прізвище та ініціали)

ОЦІНОЧНИЙ ЛИСТ

магістерської роботи студента Жука Олега-Андрія Степановича
прізвище, ім'я, по-батькові
групи ПМПМ-22с факультету прикладної математики та інформатики

Відгук наукового керівника

Тематика

	Комп'ютерне моделювання
	Чисельні методи
	Оптимізація процесів
	Системне програмування
	Бази даних
	Навчальні програми
	Веб-проектування
	Інше

Зміст

Максимальна кількість балів		
4		Складність, повнота розкриття дослідження
4		Наукова новизна, елементи творчості
4		Самостійність виконання, систематичність роботи
4		Якість та складність програм

Оформлення

4		Стиль, грамотність Ілюстративний матеріал Відповідність вимогам до роботи
Сума балів		

Коментарі

Науковий керівник

Оцінка рецензента _____б. **Оцінка за захист _____б.** **СУМА БАЛІВ _____**

Голова ЕК

РЕЦЕНЗІЯ

на магістерську роботу

студента Жука Олега-Андрія Степановича
прізвище, ім'я, по-батькові

групи ПМПМ-22с факультету прикладної математики та інформатики

	Тематика
	Комп'ютерне моделювання
	Чисельні методи
	Оптимізація процесів
	Системне програмування
	Бази даних
	Навчальні програми
	Веб-проектування
	Інше

Максимальна кількість балів		Зміст
6		Складність
6		Наукова новизна
6		Повнота розкриття дослідження
6		Якість та складність програм
		Оформлення
2		Стиль, грамотність
2		Ілюстративний матеріал
2		Відповідність вимогам до роботи
Сума балів		

Коментарі

Рецензент

Львівський національний університет імені Івана Франка

Подання
голові екзаменаційної комісії
щодо захисту магістерської роботи

Направляється студент Жук Олег-Андрій Степанович
(прізвище та ініціали)
до захисту магістерської роботи
за спеціальністю 112 «Прикладна математика»
(шифр і назва спеціальності)
на тему Узагальнення задачі про максимальний потік та його застосування
(назва теми)
_____
_____
_____

Магістерська робота і рецензія додаються.
Декан факультету _____
(підпис)

Довідка про успішність
Жук О.-А. С.
(прізвище та ініціали студента)
за період навчання на факультеті прикладної математики та інформатики
з 2020 року до 2022 року повністю виконав навчальний план
з таким розподілом оцінок за:
національною шкалою:
відмінно ____%, добре ____%, задовільно____%;
шкалою ECTS:
A ____%; B ____%; C ____%; D ____%; E ____%

Секретар факультету_____
(підпис) (прізвище та ініціали)

Висновок керівника магістерської роботи

Студент_____

Керівник роботи _____

(підпис)

"__ "_____20 ___року

Висновок кафедри про магістерську роботу

Магістерська робота розглянута.

Студент _____Жук О.-А. С._____

(прізвище та ініціали)

допускається до захисту даної роботи в Екзаменаційній комісії.

Завідувач кафедри _____

(підпис) (прізвище та ініціали)

"_____"_____ 2022 року

Зміст

Вступ

В сучасному світі логістика і планування є основою ведення будь-якого бізнесу, що перейшов рамки локального. Так, наприклад, вправне вирішення логістичних питань дозволяє оптимізувати роботу служби доставки, побудувати ефективну інфраструктуру та мінімізувати витрати на транспортування.

Часто такі та схожі задачі можна інтерпретувати як задачі на знаходження максимального потоку в мережі, знаходження мінімального розрізу, чи задачі на знаходження максимального потоку мінімальної вартості. У формулюванні задач такого плану гарантується, що потік зберігає свою потужність впродовж транспортування. Однак, практика показує, що це не завжди так — старі труби часто течуть, рідини випаровуються, а лінії електропередач втрачають частку енергії при транспортуванні на відстані.

Таким чином, постає необхідність в узагальненні задачі про максимальний потік для мереж, де ребра втрачають частку потоку.

Отже, мета даної роботи полягає в дослідженні алгоритмів розв’язку узагальненої задачі максимального потоку та застосування таких алгоритмів.

Виходячи з мети, було поставлено наступні завдання роботи:

1. Сформулювати узагальнення задачі про максимальний потік;
2. Дослідити існуючі алгоритми розв’язку задачі та оцінити їхню ефективність;
3. Реалізувати алгоритм розв’язку задачі;
4. Створити застосунок для демонстрації роботи алгоритму на заданому графі;

Перший розділ «Потокова мережа» описує задачу знаходження максимального потоку та її узагальнення, формулює умову валідності та оптимальності знайденого розв’язку.

Другий розділ «Алгоритми розв’язку задачі про максимальний потік» розглядає існуючі алгоритми розв’язку задачі та порівнює їхню ефективність.

Третій розділ «Програмна реалізація» резюмує авторську реалізацію алгоритму та веб-застосунку для його візуалізації.

Розділ 1. Потокова мережа

1.1. Потокова мережа. Означення потокової мережі

Потокова мережа – це граф, в якому для кожного ребра визначена пропускна ємність – обмеження на кількість потоку, яку воно може пропускати вздовж ребра. Вважається, що потік починається з особливої вершини-джерела і прямує до вершини стоку. В задачах на знаходження потоку в мережі дозволяється контролювати, скільки потоку можна пропустити вздовж того чи іншого ребра, за умови, що ця кількість не перевищує задану пропускну ємність ребра.

Всі задачі, які ми будемо розглядати в цій роботі, стосуються орієнтованого графу (V, E), де V – множина вершин, а E – множина ребер графу. Зазначимо, що кількість вершин в графі рівна n, а кількість ребер – m.

З поміж усіх вершин розрізняємо дві вершини: вершина джерела s та стоку t. Вершина джерела – це умовна вершина, яка генерує нескінченну кількість потоку, який прямує до вершини стоку. Задача максимального потоку полягає в знаходженні максимально можливої кількості потоку, що досягає t.

Вважатимемо, що пропускна ємність ребра – це певна величина, яка обмежує максимальну кількість потоку, що може проходити по ребру. Позначимо функцію $u: E \to \mathbb{R}$ такою, що характеризує пропускну ємність ребра. Нехай пара (v, w) – це ребро з вершини v в w, тоді позначимо пропускну ємність цього ребра рівною $u(v, w)$.

Таким чином, потокова мережа – це орієнтований граф G з визначеними пропускними ємностями для кожного ребра, іншими словами $G = (V, E, u)$.

Позначимо функцію $f: E \to \mathbb{R}$ такою, що характеризує потік графу та задовольняє обмеження пропускної ємності (1.1), скісної симетрії (1.2) та збереження потоку (1.3):

$$\forall (v, w) \in E: f(v, w) \leq u(v, w) \tag{1.1}$$

$$\forall (v, w) \in E: f(v, w) = -f(w, v) \tag{1.2}$$

$$\forall v \in V \backslash \{s, t\}: \sum_{w \in V} f(v, w) = 0 \tag{1.3}$$

Таким чином, для функції потоку f можна визначити значення залишкової пропускної ємності мережі: $u_f: E \to \mathbb{R}$. Гарантується, що u_f задовольняє рівність (1.4):

$$u_f(v,w) = u(v,w) - f(v,w) \tag{1.4}$$

Залишкова пропускна ємність мережі генерує так звану залишкову мережу $G_f = (V, E, u_f)$.

Наприклад $u(v,w) = 7, u(w,v) = 0, f(v,w) = -f(w,v) = 4$, тоді в залишковій мережі G_f пропускна здатність ребра $u_f(v,w) = 7 - 4 = 3$, тоді як $u_f(w,v) = 0 - (-4) = 4$.

Для залишкової мережі можна визначити збільшуючий шлях, як шлях з вершини s до вершини t, що проходить тільки по ребрах з позитивним значенням пропускної ємності: $(v,w) \in E$, таких що:

$$u_f(v,w) > 0$$

1.2. Задача знаходження максимального потоку

Задача знаходження максимального потоку полягає в доставці максимально можливої кількості потоку з джерела до стоку потокової мережі $G = (V, E, u)$. Оскільки функція потоку мережі f повинна задовольняти умову збереження потоку (1.3) для всіх вершин, окрім s та t, то значення функції потоку мережі можна визначити наступним чином:

$$|f| = \sum_{v \in V \ (v,t) \in E} f(v,t) \tag{1.5}$$

Отже, ціль задачі знаходження максимального потоку полягає в знаходженні функції потоку мережі з максимально можливим значенням $|f|$.

Зрозуміло, що якщо в мережі G_f існує збільшуючий шлях, то ми можемо збільшити значення потоку, пропустивши потік вздовж цього шляху. Обернене твердження також має місце, що було показано в роботах Форда та Фалкерсона [7]. Цей факт дозволяє сформулювати умову оптимальності потоку f: потік f –

максимальний тоді і тільки тоді, коли в потоковій мережі G_f не існує збільшуючого шляху.

1.3. Узагальнення задачі про максимальний потік.

Задачу про знаходження максимального потоку можна інтерпретувати в реальності для мереж постачання. Такою мережею, зокрема, може бути мережа нафтопроводу в масштабах країни. Однак, дане формулювання задачі не враховує можливі недоліки такої мережі – старі труби можуть текти і втрачати певну частку потоку в процесі транспортування. Задачу про максимальний потік в недосконалій потоковій мережі розв'язує узагальнена задача про максимальний потік.

Розглянемо стандартну задачу знаходження максимального потоку. Разом з цим для кожного ребра визначимо додатній коефіцієнт, який коригує значення кожної одиниці потоку, що проходить цим ребром. Функцію, що описує такий коефіцієнт посилення потоку, назвемо $\gamma: E \to \mathbb{R}$. Так, наприклад: нехай у вершину v прибуває 100 одиниць потоку, $\gamma(v,w) = \frac{1}{4}$, тоді після його транспортування ребром (v,w) в вершину w прибуде $100 \cdot \gamma(v,w) = 25$ одиниць потоку.

Без обмеження загальності будемо вважати, що функція γ – симетрична, тобто така, що $\gamma(v,w) = \frac{1}{\gamma(w,v)}$.

Для узагальненої потокової мережі $G = (V, E, u, \gamma)$ схожим чином визначимо узагальнену функції потоку $g: E \to \mathbb{R}$ таку, що задовольняє обмеження пропускної ємності (1.1), збереження потоку (1.3), та узагальнену скісну симетрію (1.6):

$$\forall (v,w) \in E: g(v,w) = -\gamma(w,v) g(w,v) \qquad (1.6)$$

Аналогічним чином можна визначити значення узагальненого потоку $|g|$:

$$|g| = \sum_{v \in V\ (v,t) \in E} \gamma(v,t) g(v,t) \qquad (1.7)$$

Залишкову пропускну ємність мережі по відношенню до узагальненого потоку визначимо наступним чином:

$$u_g(v,w) = u(v,w) - g(v,w) \qquad (1.8)$$

Тоді $G_g = (V, E, u_g, \gamma)$ – залишкова мережа.

Звідси і надалі, говорячи про задачу знаходження максимального потоку та її похідні означення, будемо мати на увазі їх узагальнену версію.

Для потокової мережі $G = (V, E, u, \gamma)$ визначимо функцію розфарбування мережі $\mu: V \to \mathbb{R}^+ \cup \{\infty\}$ таку, що $\mu(t) = 1$. Вперше ідею розфарбування мережі показали в своїх роботах Ф. Гловер та Д. Клінгман [5].

Згідно з такою функцією розфарбування, слід визначити розфарбовані ємності(1.9) та коефіцієнт посилення потоку (1.10):

$$u_\mu(v, w) = u(v, w)/\mu(v) \qquad (1.9)$$

$$\gamma_\mu(v, w) = \gamma_\mu(v, w)\mu(v)/\mu(w) \qquad (1.10)$$

Згідно з такими перетвореннями, утворимо розфарбовану мережу $G_\mu = (V, E, u_\mu, \gamma_\mu)$. Тоді має місце наступне твердження:

Для довільної функції розфарбування μ, g – це функція узагальненого потоку в мережі G тоді і тільки тоді, якщо $g_\mu(v, w) = g(v, w)/\mu(v)$ – узагальнений потік в мережі G_μ а також $|g| = |g_\mu|$.

Іншими словами, розфарбування мережі не впливає на розв'язок задачі про максимальний потік.

Разом з цим назвемо розфарбування μ канонічним, якщо для кожної вершини v воно рівне оберненому максимальному значенню узагальненого потоку, що досягає вершини стоку t, за умови, що з вершини v витікає одна одиниця потоку. Наприклад, нехай $\gamma(v, w) = \frac{1}{4}, \gamma(w, q) = \frac{6}{5}$ та $\gamma(q, t) = 3$, а також значення ємності потоку цих ребер більше 1. Тоді у вершину стоку прибуде рівно $\gamma(v, w)\gamma(w, q)\gamma(q, t) = \frac{1}{4} \cdot \frac{6}{5} \cdot 3 = \frac{18}{20}$ одиниць потоку, за умови, що з вершини v витікає одна така одиниця. Таким чином, канонічне розфарбування $\mu(v) = \frac{20}{18}$.

Знайти канонічне розфарбування можна за допомогою алгоритму пошуку мінімального шляху в графі з ребрами довжиною $l(v, w) = -\log \gamma(v, w)$.

Розділ 2. Алгоритми розв’язку задачі про максимальний потік

2.1. Алгоритм збільшуючого шляху

В цьому підрозділі розглянемо алгоритм Трумпера, який є певного роду узагальнення алгоритму Форда-Фалкерсона знаходження максимального потоку. Основна ідея алгоритму полягає в знаходженні найвигіднішого шляху в мережі, вздовж якого одиниця потоку досягає стоку з максимально можливим значенням, та проштовхування максимально можливого потоку вздовж такого шляху.

Використовуючи канонічне розфарбування мережі, опишемо алгоритм Трумпера:

1. Нехай оптимальний потік $g \equiv 0$.
2. Нехай початкове розфарбування мережі $\mu \equiv 1$
3. Поки існує збільшуючий шлях в G_g повторювати:
    a. Нехай μ – канонічне розфарбування графу G_g.
    b. Нехай g_μ – максимальний потік в графі $G_{g,\mu}$, що проходить тільки по ребрах (v, w) таких, що $\gamma_\mu(v, w) = 1$.
    c. Обновлюємо оптимальний потік:
    $\forall (w, v) \in E: g(w, v) \coloneqq g(w, v) + \mu(v) g_\mu(w, v)$.
4. Кінець алгоритму.

Складність знаходження канонічного розфарбування графу рівна складності знаходження найкоротшого шляху в графі. На всіх ітераціях циклу канонічне розфарбування можна знаходити, відштовхуючись від розфарбування, знайденого на попередньому кроці. Таким чином можна гарантувати, що в графі G_g для всіх ребер значення функції $\gamma \leq 1$, що в свою чергу дозволяє використовувати алгоритм Дейкстри з потенціалами Джонсона [2].

Оскільки складність знаходження найкоротшого шляху асимптотично швидша, ніж складність знаходження максимального потоку в мережі, можна вважати, що складність ітерацій циклу (a)-(c) напряму залежить від складності знаходження максимального потоку в графі.

Можна показати, що кількість ітерацій циклу (a)-(c) – скінченна, однак вона може бути експонентного порядку.

2.2. Алгоритм видалення циклів, що генерують потік.

Розглянемо алгоритм процедури, запропонований А. Голдбергом, С. Плоткіном та Є. Тардос [3].

Визначимо функцію $\pi: V \to \mathbb{R}$, що описує значення потенціалу кожної вершини. Нехай функція $c: E \to \mathbb{R}$ описує вартість ребер в початковому графі: $c(v,w) = -\log_b(\gamma(v,w))$, для певної константи b. Тоді можна визначити потенціальну вагу ребра $c_\pi(v,w)$ наступним чином:

$$c_\pi(v,w) = c(v,w) - \pi(v) + \pi(w) \qquad (2.1)$$

Ідея алгоритму опрацювання циклів полягає в знаходженні таких значень потенціалів, що виконується наступна умова (2.2) для певної змінної ϵ:

$$c_\pi(v,w) \geq -\epsilon \qquad (2.2)$$

Після знаходження таких потенціалів, для яких виконується (2.2), з графу G_g слід видалити всі такі цикли, що потенціальні ваги всіх його ребер – від'ємні. Після чого оновлюється функція потоку та зменшується значення змінної. Такий ітеративний метод дозволяє знайти і опрацювати всі цикли, що генерують потік в початковому графі за поліноміальний час.

Сформулюємо алгоритм для початкового графу G:

1. Нехай $g \equiv 0, \pi \equiv 0, \mu \equiv 0, \epsilon \coloneqq -\min\limits_{(v,w)\in E} c(v,w)$.
2. Поки існує цикл, що генерує потік в G_g, повторювати:
    a. Обчислити значення потенціалів π, що задовольняють (2.2)
    b. Видалити всі цикли по ребрах з негативними вагами c_π та оновити потік g після видалення циклів.
    c. $\epsilon \coloneqq \epsilon \cdot \frac{n-1}{n}$
3. Визначаємо розфарбування $\mu(v) \coloneqq b^{\pi(v)-\pi(t)}$
4. Кінець алгоритму.

Важливим є спостереження, що, коли $\epsilon = 0$, то потенціали вершин, що задовольняють (2.2), гарантують наявність такого розфарбування графу, в якому $\gamma_\mu \leq 1$.

Оцінимо складність такого алгоритму. Видалення всіх циклів (крок 2.b.) можна зробити за допомогою алгоритму пошуку в глибину за $O(m^2)$. Втім, якщо в процесі відмічати вершини, які вже не є частиною циклів, складність алгоритму можна оцінити як $O(nm)$. Це можна покращити до $O(mlog(n))$ з використанням динамічних дерев Слетора і Тар'яна [4]. Програмну реалізацію цього кроку можна переглянути в додатках.

Разом з цим, на проміжних етапах крок 2.a. можна обчислювати за $O(m)$ за допомогою алгоритму топологічного сортування. В додатках наведений приклад реалізації цього алгоритму.

Перевірку існування циклів, що генерують потік (крок 2), можна робити за допомогою алгоритму Беллмана-Форда. Його складність – $O(nm)$. Цю перевірку можна робити з меншою частотою – кожні n ітерацій циклу, що оптимізує загальну складність алгоритму. Його реалізацію можна переглянути в додатках.

Таким чином, складність однієї ітерації циклу повною мірою залежить від складності знаходження негативних циклів в графі.

Стверджується, що кількість ітерацій циклу 2 залежить від значень ваг ребер в графі G. Якщо їх можна представити як частку цілих чисел на проміжку від 1 до C, то кількість ітерацій циклу 2 рівна $O\left(n^2 log(C)\right)$. Звідси випливає, що загальна складність алгоритму – $O(mn^3 \log(C))$, а з використанням динамічних дерев Слетора і Тар'яна – $O(mn^2 log(n) \log(C))$. Реалізацію алгоритму без використання динамічних дерев можна переглянути в додатках.

2.3. Жадібний алгоритм знаходження максимального потоку

Розглянемо жадібну версію алгоритму описаного в 2.1. де на кожному кроці ми знаходимо шлях з найбільшим значенням потоку і проштовхуємо потік вздовж такого шляху. Слід зазначити, що таким чином розв'язується схожа задача – алгоритм знаходження максимального потоку мінімального значення.

На відміну від попереднього алгоритму, ми не можемо гарантувати, що на кожному кроці не утворяться цикли, що генерують потік, тому на кожній ітерації алгоритму слід знаходити і опрацьовувати окремо такі цикли.

Опишемо цей жадібний алгоритм:

1. Нехай оптимальний потік $g \equiv 0$
2. Поки існує збільшуючий шлях в G_g повторювати:
    a. $g' \coloneqq \text{ОпрацюванняЦиклів}(G_g)$
    b. $g \coloneqq g + g'$
    c. $P \coloneqq$ Шлях з найбільшим потоком в G_g
    d. Обновлюємо опимальний потік g шляхом P
3. Кінець алгоритму.

Знаходити шлях з найбільшим потоком можна схожим принципом, як в попередньому алгоритмі – за допомогою ідеї алгоритму Дейкстри з потенціалами Джонсона[2] або з використанням алгоритму Беллмана-Форда. Реалізацію алгоритму Беллмана-Форда можна переглянути в додатках.

Згідно з 2.2. функція, що опрацьовує та видаляє цикли в графі, працює зі складністю $O(mn^2 log(n)\log(C))$.

Стверджується, що цикл 2 потребує $O\big(mlog(|f|)\big)$ ітерацій. Таким чином, загальну складність такого алгоритму можна оцінити рівною $O\big(m^2n^2log(n)log(\mathrm{C}n)\big)$.

В додатках наведено приклад реалізації цього алгоритму без використання динамічних дерев Слетора і Тар'яна з загальною складністю $O\big(m^2n^3log(\mathrm{C}n)\big)$.

Розділ 3. Програмна реалізація

Програмна реалізація складається з двох основних частин:

1. Реалізація алгоритму пошуку максимального потоку, описаного в 2.3.
2. Реалізація веб-застосунку для візуалізації роботи алгоритму.

В додатках можна знайти основні компоненти авторської реалізації алгоритму мовою програмування Rust. Реалізовані алгоритми використовують стандартний набір бібліотек та структур даних.

Веб застосунок розроблено з використанням наступних технологій та бібліотек: typescript, React, mui, webpack, react-force-graph (унаочнення графу), wasm-tool (інтеграція скомпільованої Rust-бібліотеки).

Вихідний код програмної реалізації можна переглянути та завантажити з онлайн-репозиторію за посиланням: https://github.com/andrii-zhuk/generalizedFP.

Веб застосунок для візуалізації роботи алгоритму можна переглянути за посиланням: https://graph.algotester.com.

Висновки

В цій роботі було описано теоретичні засади узагальнення задачі про максимальний потік та описано та оцінено складність алгоритмів, які дозволяють знайти оптимальний розв’язок до задачі.

Так, існують алгоритми з потенційно експонентним часом виконання, описаний в розділі 2.1. Крім того, ми розглянули алгоритм з поліноміальним часом виконання, який знаходить оптимальний розв’язок зі складністю $O\left(m^2n^2log(n)log(C)log(Cn)\right)$ – розділ 2.3. В роботі запропоновано авторську програмну реалізацію цього алгоритму мовою програмування Rust. Окрім того, було створено веб-застосунок для демонстрації його роботи.

Додатки

Додаток А. Алгоритм Беллмана-Форда

```
pub fn bellman_ford(
    graph: &DirectedGraph,
    edge_value: fn(edge: &Edge) -> Option<f64>,
    mode: Mode,
    start_id: usize,
    reverse: bool,
) -> (Vec<Option<f64>>, Vec<Option<usize>>, Option<Vec<usize>>) {
    let compare = match &mode {
        Mode::Min => |x: f64, y: f64| -> bool { x - y < -EPSILON },
        Mode::Max => |x: f64, y: f64| -> bool { x - y > EPSILON },
    };
    let current_node = if reverse == true {
        |edge: &Edge| edge.to_id
    } else {
        |edge: &Edge| edge.from_id
    };
    let next_node = if reverse == true {
        |edge: &Edge| edge.from_id
    } else {
        |edge: &Edge| edge.to_id
    };
    let mut parents: Vec<Option<usize>> = vec![None; graph.n()];
    let mut dist: Vec<Option<f64>> = vec![None; graph.n()];
    dist[start_id] = Some(0.0);
    for i in 0..graph.n() {
        for (edge_id, edge) in graph.edges_list.iter().enumerate() {
            if dist[current_node(edge)] == None
                || !graph.reachable_from_source(current_node(edge))
                || !graph.reachable_from_source(next_node(edge)) {
                continue;
            }
            let value = edge_value(edge);
            let value = match &value {
                None => continue,
                Some(value) => value,
            };
            if dist[next_node(edge)] == None
                || compare(
                    dist[current_node(edge)].unwrap() + value,
                    dist[next_node(edge)].unwrap(),
                ) {
```

```
                if i == graph.n() - 1 {
                    let mut cycle: Vec<usize> = vec![];
                    let mut node_id = current_node(edge);
                    cycle.push(node_id);
                    while (cycle.len() == 1 || node_id != cycle[0]) && cycle.len() <= 2 *
graph.n() {
                        let edge_id = parents[node_id]
                            .expect("Negative cycle retrieving error: Undefined parent.");
                        cycle.push(edge_id);
                        node_id = current_node(&graph.edges_list[edge_id]);
                        cycle.push(node_id);
                    }


                    cycle.reverse();
                    while cycle.len() > 1 && cycle.first().unwrap() != cycle.last().unwrap() {
                        cycle.pop();
                        cycle.pop();
                    }
                    if reverse {
                        cycle.reverse();
                    }
                    return (dist, parents, Some(cycle));
                }
                dist[next_node(edge)] = Some(dist[current_node(edge)].unwrap() + value);
                parents[next_node(edge)] = Some(edge_id);
            }
        }
    }


    return (dist, parents, None);
}
```

Додаток Б. Алгоритм видалення всіх циклів з від’ємними ребрами

```
fn remove_negative_cycles(graph: &mut DirectedGraph, potentials: &Vec<f64>) -> Option<f64> {
    let mut status = vec![0; graph.n()];
    let mut cycle: Vec<usize> = Vec::<usize>::new(); cycle.reserve(graph.n());
    fn dfs(node_id: usize, graph: &DirectedGraph, potentials: &Vec<f64>, status: &mut
Vec<i32>, cycle: &mut Vec<usize>) {
        status[node_id] = 1;
        for &edge_id in &graph.adj_lists[node_id] {
            let edge = &graph.edges_list[edge_id];
            if edge_value(edge, potentials) == None || status[edge.to_id] == 2 {
                continue;
            }
            if status[edge.to_id] == 0 {
                dfs(edge.to_id, graph, potentials, status, cycle);
            }
            if status[edge.to_id] == 1 || cycle.len() > 0 {
                if cycle.len() == 0 || cycle.len() % 2 == 0 {
                    cycle.push(edge.to_id); cycle.push(edge_id);
                    if cycle[0] == edge.from_id {
                        cycle.push(edge.from_id); cycle.reverse();
                    }
                }
                status[node_id] = 0;
                return;
            }
        }
        status[node_id] = 2;
    }
    let mut cumulative_excess = 0.0;
    for node_id in 0..(graph.n()) {
        if !graph.nodes[node_id].reachable_from_source || status[node_id] == 2{
            continue;
        }
        dfs(node_id, &graph, potentials, &mut status, &mut cycle);
        if cycle.len() == 0 { continue; }
        let cycle_edges = get_path_edge_ids(Some(cycle.clone()));
        let excess = propagate_cycle(graph, cycle_edges).unwrap_or(0.0);
        cumulative_excess += excess;
        cycle.clear();
    }
    if cumulative_excess < EPSILON {
        return None;
    }
    Some(cumulative_excess)
}
```

Додаток В. Алгоритм проштовхування потоку вздовж циклу

```
pub fn propagate_cycle(
    graph: &mut DirectedGraph,
    cycle: Option<Vec<usize>>,
) -> Option<f64> {
    if cycle == None {
        return None;
    }
    let cycle = cycle.unwrap();
    let mut flow: Option<f64> = None;
    for &edge_id in cycle.iter() {
        let edge = &graph.edges_list[edge_id];
        let available = edge.capacity - edge.flow;
        if available < EPSILON {
            return None;
        }
        flow = match flow {
            None => Some(available * edge.amplification),
            Some(value) => Some(value.min(available) * edge.amplification),
        };
    }
    let mut flow = match flow {
        None => return None,
        Some(value) => value,
    };
    let flow_in_destination = flow;

    let &cycle_start = cycle.first().unwrap();
    let cycle_start = graph.edges_list[cycle_start].from_id;

    graph.nodes[cycle_start].excess += flow_in_destination;

    for &edge_id in cycle.iter().rev() {
        let reverse_edge = graph.reverse_edge_ids[edge_id];
        let reverse_edge = &mut graph.edges_list[reverse_edge];

        reverse_edge.flow -= flow;
        flow *= reverse_edge.amplification;

        let edge = &mut graph.edges_list[edge_id];
        edge.flow += flow;
    }

    return Some(flow_in_destination);
}
```

Додаток Г. Алгоритм оновлення потенціальних ваг ребер мережі

```
fn recalculate_potentials(graph: &DirectedGraph, potentials: &mut Vec<f64>, barrier: f64) {
    let mut stack: Vec<usize> = vec![];
    let mut status = vec![0; graph.n()];

    fn dfs(
        node_id: usize,
        graph: &DirectedGraph,
        status: &mut Vec<i32>,
        potentials: &Vec<f64>,
        stack: &mut Vec<usize>,
    ) {
        status[node_id] = 1;
        for &edge_id in &graph.adj_lists[node_id] {
            let edge = &graph.edges_list[edge_id];
            if edge_value(edge, &potentials) == None || status[edge.to_id] != 0 {
                continue;
            }
            dfs(edge.to_id, graph, status, potentials, stack);
        }
        stack.push(node_id);
    }

    for node_id in (0..graph.n()).rev() {
        if status[node_id] != 0 {
            continue;
        }
        dfs(node_id, graph, &mut status, &potentials, &mut stack)
    }

    for (topological_order, node_id) in stack.iter().rev().enumerate() {
        potentials[*node_id] += (topological_order as f64) * barrier / (graph.n() as f64);
    }
}
```

Додаток Д. Алгоритм видалення циклів, що генерують потік

```
pub fn cancel_cycles(graph: &mut DirectedGraph) {
    let mut counter = 0;
    let mut potentials = vec![0.0; graph.n()];
    let edge_value = |edge: &Edge| -> Option<f64> {
        if edge.capacity - edge.flow < EPSILON {
            None
        } else {
            Some(-(edge.amplification).log(E))
        }
    };

    let mut barrier: Option<f64> = None;
    for edge in &graph.edges_list {
        let cost = match edge_value(&edge) {
            None => continue,
            Some(value) => value,
        };
        barrier = Some(barrier.unwrap_or(-cost).max(-cost));
    }

    if let Some(mut barrier) = barrier {
        while has_cycles(graph, &mut counter) {
            remove_negative_cycles(graph, &potentials);
            recalculate_potentials(graph, &mut potentials, barrier);
            barrier *= 1.0 - 1.0 / (graph.n() as f64);
        }
    }
}
```

Додаток Е. Алгоритм знаходження максимального потоку в мережі

```
pub fn find_flow(graph: &mut DirectedGraph) -> f64 {
    let mut result = 0.0;
    let mut step = 0;
    while has_augmenting_path(&graph) != None {
        cancel_cycles(graph);
        let augmenting_path = get_augmenting_path(graph);
        let augmenting_path_edges = get_path_edge_ids(augmenting_path);

        let flow = propagate_path(graph, augmenting_path_edges);
        result += flow.unwrap_or(0.0);
    }
    result
}
```

Список використаної літератури

[1] Томас Г. Кормен, Чарлз Е. Лейзерсон, Роналд Л. Рівест, Кліфорд Стайн. Вступ до алгоритмів. — К. : К. І. С., 2019. — 1288 с

[2] Andrew V. Goldberg An Efficient implementation of a scaling minimum-cost flow algorithm - Journal of Algorithms, 1997

[3] A. V. Goldberg, S. A. Plotkin, and E. Tardos. Combinatorial algorithms for the ´ generalized circulation problem. Mathematics of Operations Research, 16:351– 379, 1991

[4] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. Journal of Computer and System Sciences, 26:362–391, 1983

[5]F. Glover and D. Klingman. On the equivalence of some generalized network flow problems to pure network problems. Mathematical Programming, 1973.

[6] K. D. Wayne. Generalized maximum flow algorithms, 1991

[7] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. Canadian Journal of Mathematics, 1956

[8] V. Chvatal. Linear programming. W. H. Freeman, New York, 1983