НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий фізико-технічний інститут
кафедра математичного моделювання та аналізу даних

«На правах рукопису»
УДК _____

До захисту допущено:
Завідувач кафедри
Наталія КУССУЛЬ
«___»_____20__ р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-науковою програмою «Математичні методи моделювання, розпізнавання образів та комп'ютерного зору»

зі спеціальності 113 «Прикладна математика»

на тему: «Задача розмітки на деревах для випадку поступового надходження даних»

Виконав:
студент VI курсу, групи ФІ-01мн
Кириленко Іван Андрійович _____

Науковий керівник:
доцент кафедри ММАД, к.ф.-м.н
Орєхов О. А _____
_____

Рецензент:
м.ф-м.н.,
Павло Михайлович Курочка _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2022 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий фізико-технічний інститут
кафедра математичного моделювання та аналізу даних

Рівень вищої освіти – другий (магістерський)

Спеціальність – 113 «Прикладна математика»

Освітньо-наукова програма: «Математичні методи моделювання, розпізнавання образів та комп'ютерного зору»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Наталія КУССУЛЬ
«___»_____20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Кириленко Іван Андрійович

1. Тема дисертації «Задача розмітки на деревах для випадку поступового надходження даних», науковий керівник дисертації Орєхов Олександр Арсенійович, доцент, к.ф.-м.н., затверджені наказом по університету від 08.06.2022 р. № 976-с

2. Термін подання студентом дисертації _____

3. Об'єкт дослідження

4. Предмет дослідження

5. Перелік завдань, які потрібно розробити

6. Орієнтовний перелік графічного (ілюстративного) матеріалу

7. Орієнтовний перелік публікацій

8. Консультанти розділів дисертації*

* Якщо визначені консультанти. Консультантом не може бути зазначено наукового керівника магістерської дисертації.

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

9. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка

Студент Іван КИРИЛЕНКО

Науковий керівник Олександр ОРЄХОВ

ABSTRACT

This thesis contains 52 pages, 50 figures and 9 references.

The object of this study is Markov's hidden model.

The subject of this study is effective estimation of Markov's model hidden parameters.

This thesis examines tree labeling problem for slowly incoming data. Effective algorithms for solving this problem are developed and implemented in C++. Run time of algorithms are compared for different data transmission cases. The analysis of acquired results is conducted.

COMPUTER VISION, TREE LABELING, ONLINE ALGORITHMS, PATTERN RECOGNITION.

РЕФЕРАТ

Обсяг роботи 52 сторінки, 50 ілюстрацій, 3 додатки, 9 джерел.

Об'єкт дослідження — прихована модель Маркова.

Предмет дослідження — ефективна оцінка прихованих параметрів моделі Маркова.

У роботі розглянуто задачу розмітки на деревах для випадку повільного надходження даних. Розроблено алгоритми для ефективного розв'язання задачі розмітки для різних випадків повільного надходження даних. Розроблена програмна імплементація запропонованих алгоритмів на мові C++. Порівняно час роботи алгоритмів після надходження всіх даних для різних випадків та характерів надходження цих даних. Проведено аналіз отриманих результатів.

КОМП'ЮТЕРНИЙ ЗІР, ЗАДАЧІ РОЗМІТКИ, ОНЛАЙН АЛГОРИТМИ, РОЗПІЗНАВАННЯ ОБРАЗІВ.

ЗМІСТ

ВСТУП

Задачі розмітки є одною з основних проблем у сфері розпізнавання образів. Від доповненої реальності до автономної навігації — задачі розмітки мають багато застосувань в сучасному світі.

Задачі розмітки вирішують багато задач. Велике поширення мають у сфері комп'ютерного зору.

Актуальність роботи. Offline алгоритми розв'язання задачі розмітки можуть почати свою роботу тільки після надходження всіх даних. Проте, з розвитком мережевих технологій все частіше трапляється, що необхідна інформація завантажується безпосередньо з інтернету по мірі необхідності, а не зберігається локально. Також можлива ситуація, коли дані для обрахунку відправляються до обчислювального центру. Сучасні дата-центри мають дуже швидкі та надійні канали доступу до мережі, цього не можна сказати про їх клієнтів. Тож до часу роботи самого алгоритму буде додаватися ще й час надходження всіх даних. Цю проблему можна вирішити, використовуючи алгоритми, які можуть починати обрахунки вже після надходження першої частини даних, зменшуючи тим самим сумарний час обробки даних.

Об'єкт дослідження — прихована модель Маркова.

Предмет дослідження — ефективна оцінка прихованих параметрів моделі Маркова.

Мета дослідження. Розробка онлайн алгоритму для задачі розмітки на деревах.

Практичне значення результатів. Розроблений алгоритм можна використовувати для ефективного вирішення задачі розмітки на деревах в умовах повільного надходження даних.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ

Перший розділ присвячено опису задач розмітки та їх постановці на деревах. Описаний спосіб розв'язання таких задач за допомогою представлення їх як задачі пошуку оптимального шляху на графі спеціального вигляду.

1.1 Постановка задачі розмітки

Багато прикладних задач можуть бути зведені до задач розмітки. І дуже часто задачі розмітки природно зводяться до задач оптимізації на графах. В цій роботі будемо розглядати задачі розмітки на деревах.

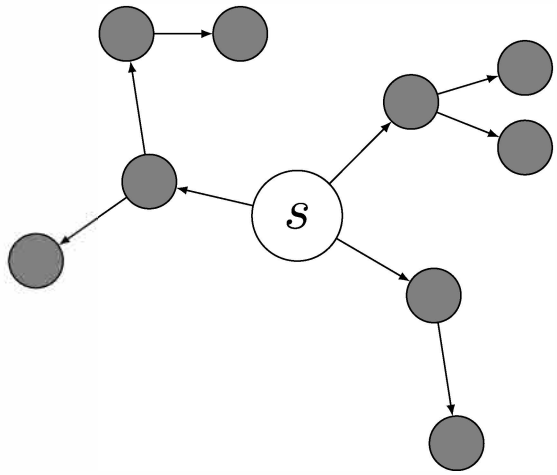


Рисунок 1.1 — Дерево G.

Нехай є деякий граф $G =< \mathcal{V}, \mathcal{E} >$, де $\mathcal{V}$ – множина вершин графу, а $\mathcal{E}$ – множина ребер графу. Нехай кожна вершина $p \in \mathcal{V}$ може мати мітку d_p із множини міток D. Розміткою графу G будемо називати послідовність $\overline{d} \in D^{\mathcal{V}}$.

Для того, щоб визначити якість розмітки $\overline{d}$ введемо функції унарної якості h та бінарної якості g:

$$h : \mathcal{V} \times D \to \mathcal{R},$$

$$g : D \times D \to \mathcal{R}.$$

Функція h буде відповідати за якість самої вершини, а функція g буде відповідати за якість сусідства та умовну "гладкість"всієї розмітки загалом. Таким чином загальну якість E всього графу G будемо визначати як суму якостей вершин та зв'язків між ними:

$$E = \sum_{p \in \mathcal{V}} h(p, d_p) + \sum_{(p,q) \in \mathcal{E}} g(d_p, d_q), \tag{1.1}$$

Вираз 1.1 зазвичай називають "Енергією графу".

Задача розмітки графу G полягає у знаходженні такої оптимальної розмітки $d^* \in D^{\mathcal{V}}$, яка б мінімізувала (або максимувала) енергію графу G.

$$d^* \in \arg\min_{d \in D^{\mathcal{V}}} \sum_{p \in \mathcal{V}} h(p, d_p) + \sum_{(p,q) \in \mathcal{E}} g(d_p, d_q), \tag{1.2}$$

1.2 Представлення задачі як пошук оптимального шляху на графі спеціального вигляду

Представимо задачу пошуку послідовності d^* , що мінімізує енергію графу G (1.1), як задачу пошуку найкоротшого шляху через орієнтований зважений граф спеціального вигляду.

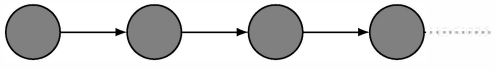


Рисунок 1.2 — Граф-ланцюг.

Нехай початкове дерево буде ланцюгом (1.2). Хоч це і доволі простий випадок, проте він доволі часто зустрічається у реальних задачах, наприклад у задачі построкового стереозору (англ. *scanlines stereo*). До того ж, на такій

ситуації легше продемонструвати побудову відповідного графу спеціального вигляду, для розв'язку відповідної задачі розмітки.

Поставимо задачу розмітки для такої ситуації. Граф $G =< \mathcal{V}, \mathcal{E} >$ має вигляд ланцюжка. Кожна вершина $p \in \mathcal{V}$ може мати мітку d_p із множини міток D. Маємо унарну функцію якості $h : D \to \mathcal{R}$, та бінарну функцію якості $g : D \times D \to \mathcal{R}$. Хочемо знайти оптимальну послідовність $\overline{d} \in D^{\mathcal{V}}$, яка мінімізує енергію графу G.

$$d^* \in \arg\min_{\overline{d} \in D^{\mathcal{V}}} \sum_{p \in \mathcal{V}} h(p, d_p) + \sum_{(p,q) \in \mathcal{E}} g(d_p, d_q), \tag{1.3}$$

Побудуємо орієнтований зважений граф $G' =< \mathcal{V}', \mathcal{E}' >$ відповідний до задачі (1.3). Кожній вершині графу G буде відповідати стовпчик із $|D|$ вершин у графі G', а кожному ребру у графі G буде відповідати $|D|^2$ ребер у графі G' таким чином, що:

$$p \in \mathcal{V} \implies p_d \in \mathcal{V}', \forall d \in D,$$
$$(p,q) \in \mathcal{E} \implies (p_{d1}, q_{d2}) \in \mathcal{E}', \forall d1 \in D, \forall d2 \in D.$$

Кожна вершина (ланка) (1.3) графу G "розкривається" у повнозв'язний стовпчик вершин (1.4) у графі G'.

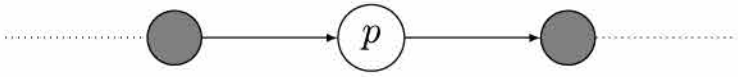


Рисунок 1.3 — Ланка у графі G

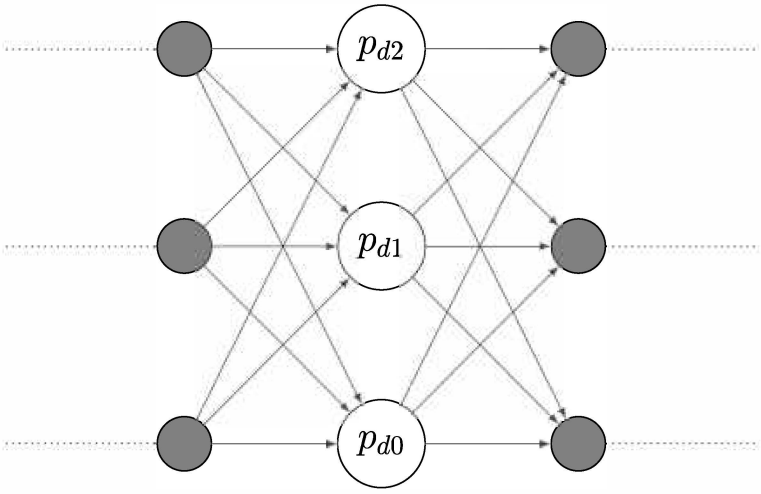


Рисунок 1.4 — Відповідна ланка у графі G'

Введемо функцію ваги вершини $w : \mathcal{V}' \to \mathcal{R}$. Кожній вершині $p_d \in \mathcal{V}'$ призначимо вагу

$$w(p_d) = h(p, d).$$

Введемо також функцію ваги ребер $w : \mathcal{E}' \to \mathcal{R}$. Кожному ребру $(p_{d1}, q_{d2}) \in \mathcal{E}'$ призначимо вагу

$$w(p_{d1}, q_{d2}) = g(p_{d1}, q_{d2}).$$

Тоді зважена ланка (1.4) буде виглядати як на рисунку (1.5).

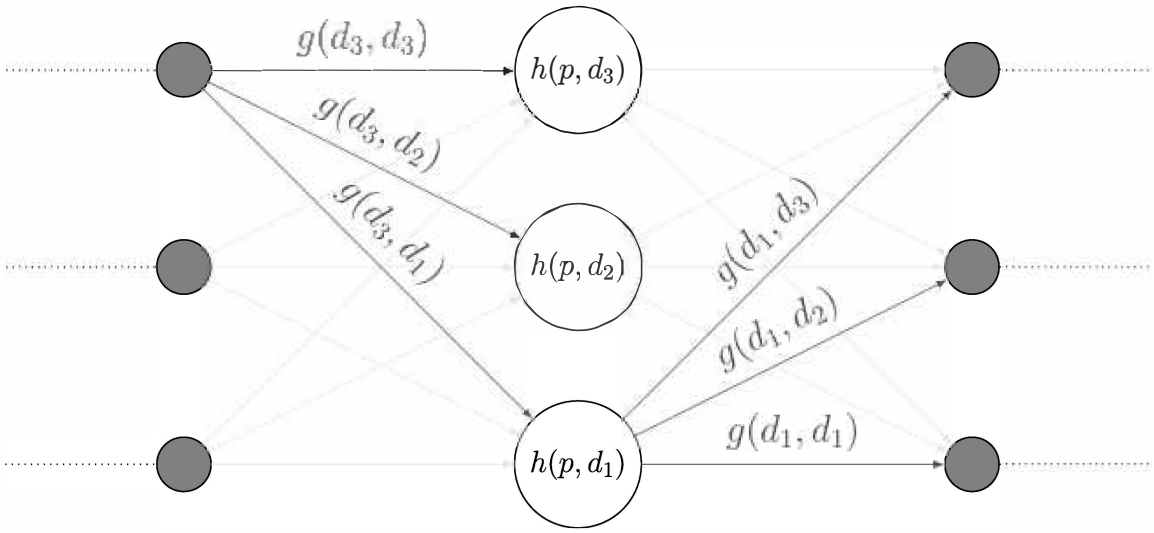


Рисунок 1.5 — Ваги ланки у графі G'

В кожному стовпчику вершин виберемо по одній вершині p_{d^*}, та назвемо множину таких вершин $\hat{\Delta}$. Множина таких вершин буде відповідати деякій

розмітці $\overline{d}$. А енергія графу G з такою розміткою $\overline{d}$, буде відповідати довжині шляху, проведеного через обрані вершини і буде дорівнювати

$$E(\overline{d}) = \sum_{p_d \in \hat{\Delta}} h(p, d_p) + \sum_{\substack{(p_d, q_{d'}) \in \mathcal{E}' \\ p_d, q_{d'} \in \hat{\Delta}}} g(d_p, d_q), \tag{1.4}$$

Виходить, що для того щоб знайти оптимальну розмітку d^* графу G, необхідно знайти таку послідовність вершин у графі G', яка відповідає кращому шляху через граф G'.

1.3 Пошук кращого шляху на ланцюжку

Для випадку графу-ланцюжка задача знаходження кращого шляху на графі гарно вирішується за допомогою підходу динамічного програмування.

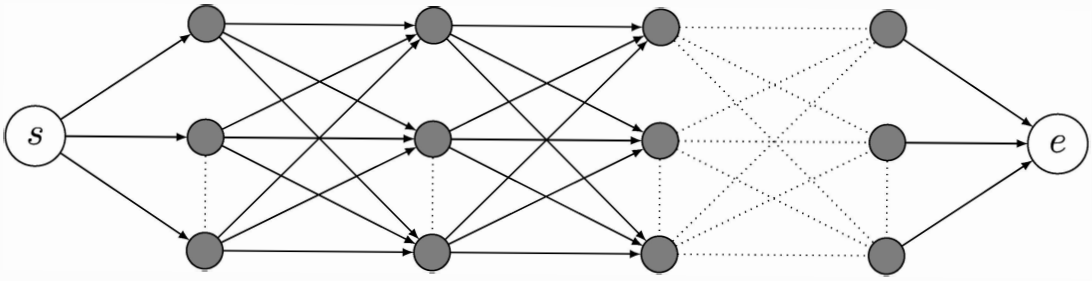


Рисунок 1.6 — Пошук розмітки на ланцюжку.

Позначимо довжину найкоротшого шляху з початкової вершини S в вер-

шину $_d$ як $f_p(d)$. Тоді

$$
\begin{aligned}
f_1(d) &= h(1,d), \forall d \in D \\
f_2(d) &= \min_{d' \in D} \Big(f_1(d) + g(d',d) \Big) + h(2,d), \forall d \in D \\
&\vdots \\
f_i(d) &= \min_{d' \in D} \Big(f_{i-1}(d) + g(d',d) \Big) + h(i,d), \forall d \in D.
\end{aligned}
$$

Тоді елементи послідовності $\overline{d}$ знаходимо за формулами

$$
\begin{aligned}
d_n &= \arg\min_{d' \in D} \big(f_n(d') \big), \\
d_i &= \arg\min_{d' \in D} \big(f_i(d') + g(d', d_{i+1}) \big), \ i = \overline{n-1, 1}.
\end{aligned}
$$

Висновки до першого розділу

У першому розділі було розглянуто основні теоретичні відомості необхідні для подальшого розуміння та постановки задачі розмітки. Була поставлена задача розмітки на деревах та зведення цієї задачі до задачі пошуку найкоротшого шляху на орієнтованому зваженому графі спеціального вигляду. Для задачі пошуку найкоротшого шляху був наведений алгоритм її вирішення, що використовує динамічне програмування. Постановка задачі розмітки для випадку повних даних та її зведення до задачі пошуку найкоротшого шляху на графі, що наведена в цьому розділі, буде узагальнена на випадки повільного надходження даних у наступному розділі.

2 ПОПЕРЕДНІ РОБОТИ

Другий розділ присвячено попереднім роботам на тему задачі розмітки для випадку повільного надходження даних. Розглядається постановка та методи вирішення задачі розмітки для повільного надходження даних на прикладі задачі стереозору. Досліджується алгоритм розв'язання такої задачі.

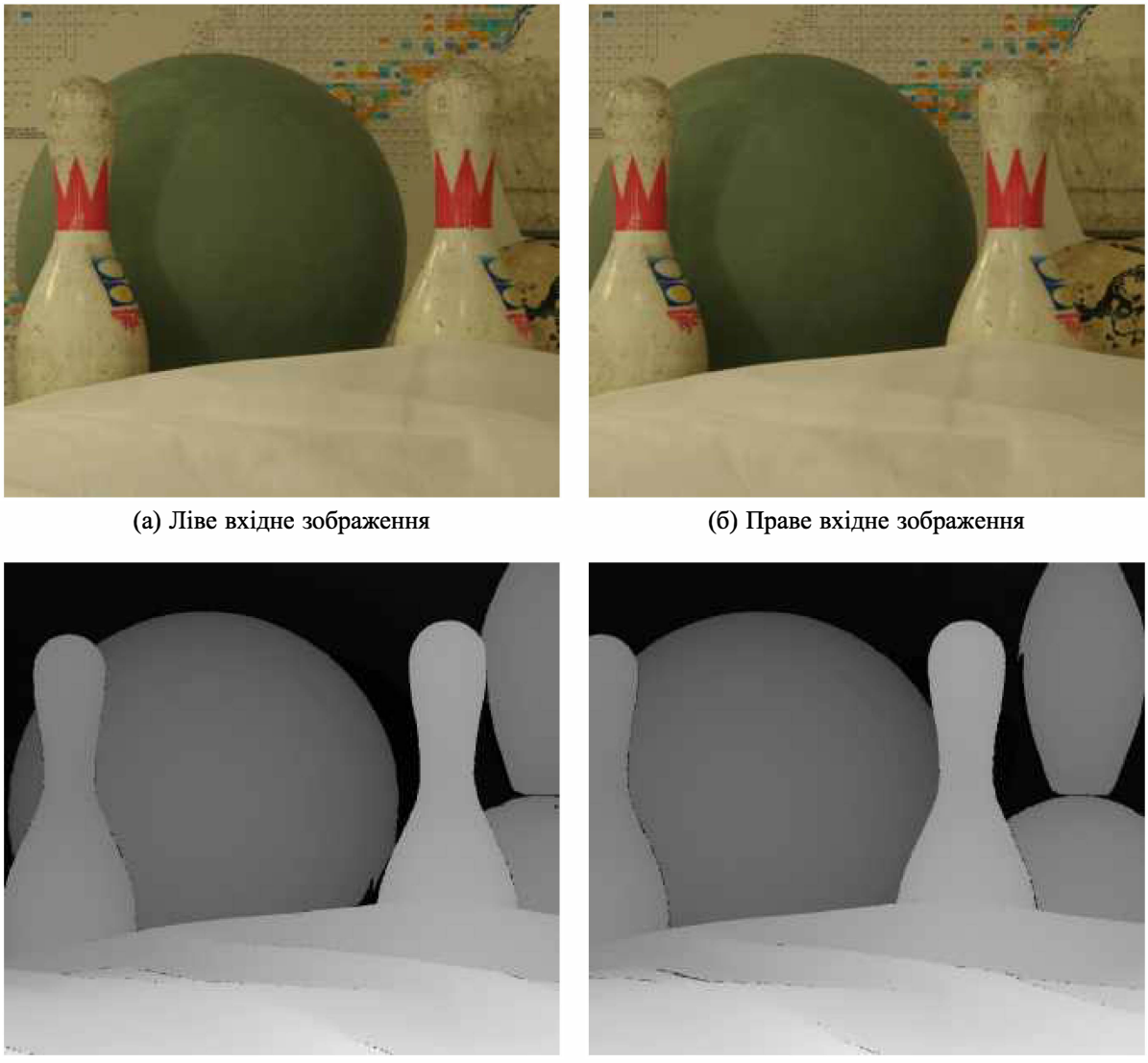
2.1 Попередні дослідження

Ця робота є частковим продовженням дослідження проблеми розмітки в умовах поступового надходження даних. Попередня робота "Задача стереозору в умовах поступового надходження даних"[2] була присвячена більш конкретній задачі розмітки в умовах поступового надходження даних, а саме задачі построкового стереозору (англ. *scanline stereo*). До того ж, у тій роботі розглядалися тільки деякі випадки надходження даних. У попередній роботі[2] було розглянуто ситуацію коли інформація надходила поступово та упорядковано, та було запропоновано алгоритм, який є ефективним для такого випадку. І хоч значна частина методів передачі інформацію передає її упорядковано, можливі ситуації коли характер надходження даних заздалегідь невідомий.

Дослідження теми задачі построкового стереозору в умовах поступового надходження даних було продовжено в роботі[1]. В цій роботі було розглянуто більш загальну ситуацію надходження даних – коли надходження інформації має хаотичний характер. Також в цій роботі було запропоновано алгоритм розв'язання такої задачі, що, для ситуації повільного надходження даних, є більш ефективним ніж звичайні алгоритми пошуку найкращого шляху на графі, адже він дозволяє виконувати передобрахунки по мірі надходження інформації, а не чекати поки не надійдуть всі дані.

2.2 Задача стереозору в умовах повільного надходження даних

У роботі[2] було розглянуто задачу стереозору в умовах повільного надходження даних. Мета розв'язання задачі стереозору полягає у знаходженні карти глибин сцени шляхом аналізу вхідної стереофотографії цієї самої сцени(2.2).



(а) Ліве вхідне зображення

(б) Праве вхідне зображення

(в) Карта глибини 1

(г) Карта глибини 2

Рисунок 2.1 — Приклад роботи алгоритму стереозору

Метод розв'язання цієї задачі полягає у знаходженні скалярного поля зсувів пікселів між зображеннями(2.2).

Задача построкового стереозору формулюється так: нехай на вході маємо

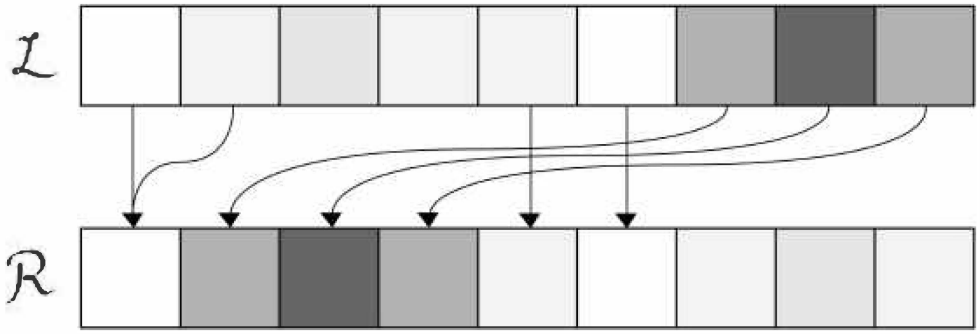


Рисунок 2.2 — Зсуви пікселів між зображеннями

два зображення-стрічки $\mathcal{L} : I \to \mathcal{C}$ та $\mathcal{C} : I \to \mathcal{R}$, де $I = \{1,2,...,n\}$ – множина пікселів у зображені, а $\mathcal{C}$ – множина кольорів. Також маємо множину зсувів $D = \{0,1,...,D_{max}\}$.

Для визначення відповідності пікселів введемо дві функції якості: $h : I \times D \to \mathcal{R}$ – відповідає за схожість кольору пікселів, $g : D \times D \to \mathcal{R}$ – відповідає за гладкість розмітки, α – параметр гладкості.

$$h(i, d_i) = |\mathcal{L}(i) - \mathcal{R}(i - d_i)|,$$
$$g(d_i, d_j) = \alpha|d_i - d_j|.$$

Розв'язком задачі будемо називати таку послідовність $\overline{d} \in D^I$, яка мінімізує "енергію задачі"

$$E(\overline{d}) = \sum_{i=1}^{n} h(i, d_i) + \sum_{i=1}^{n-1} g(d_i, d_{i+1}). \tag{2.1}$$

2.3 Представлення та вирішення задачі як пошук оптимального шляху на графі

Задачу (2.1) можна звести до задачі пошуку кращого шляху на графі спеціального вигляду як вже було описано у розділі (1.2). В результаті отримаємо орієнтований зважений граф такого вигляду:

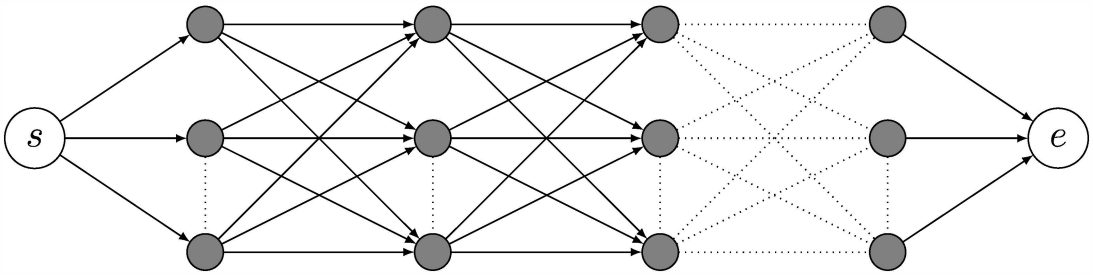


Рисунок 2.3 — Пошук розмітки на ланцюжку.

Ваги у якому будуть мати такий вигляд:

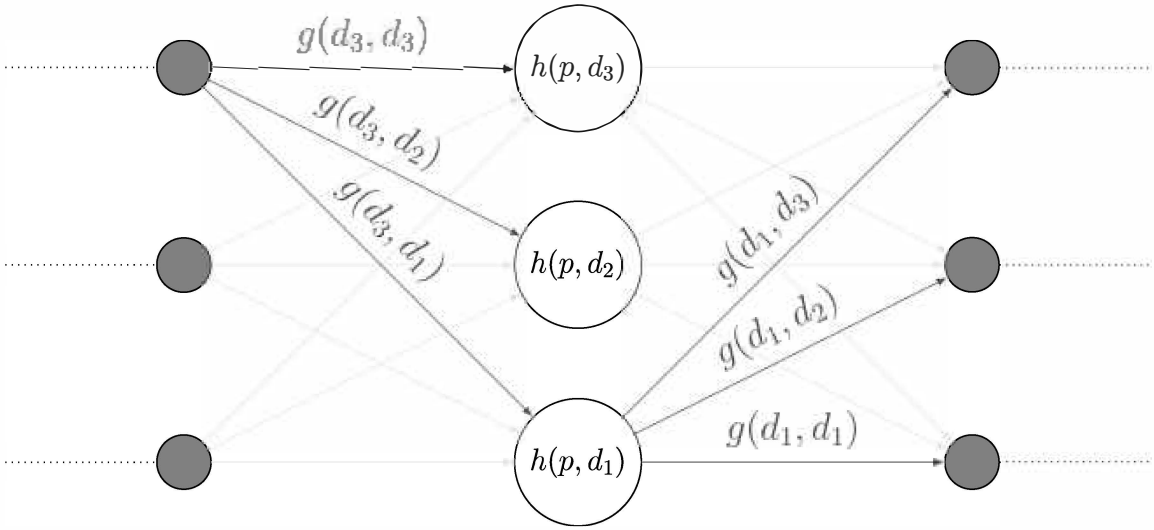


Рисунок 2.4 — Ваги ланки у графі G'

Для знаходження оптимальної послідовності d^*, такої що:

$$d^* \in \arg\min_{\overline{d} \in D^{\mathcal{V}}} \sum_{p \in \mathcal{V}} h(p, d_p) + \sum_{(p,q) \in \mathcal{E}} g(d_p, d_q), \tag{2.2}$$

Використаємо вже описаний метод динамічного програмування(1.3). Тоді еле-

менти послідовності $\overline{d}$ знаходимо за формулами

$$d_n = \arg\min_{d' \in D} \big(f_n(d')\big),$$
$$d_i = \arg\min_{d' \in D} \big(f_i(d') + g(d', d_{i+1})\big),\ i = \overline{n-1, 1}.$$

2.4 Онлайн алгоритм пошуку розмітки

Метод розв'язання задачі стереозору, який був описаний до цього, вимагає щоб вся інформація була в наявності до моменту початку його роботи. У випадку, коли надходження всієї інформації доводиться довго чекати такий підхід не є самим ефективним. Було б краще, якби ми могли виконувати якісь попередні обрахунки з інформацією, яка нам вже доступна, таким чином, щоб це зменшило час необхідний для отримання результату після надходження всіх даних.

2.4.1 Постановка задачі

Як і раніше, маємо зображення-стрічки $\mathcal{L} : I \to \mathcal{C}$ та $\mathcal{C} : I \to \mathcal{R}$, де $I = \{1,2,...,n\}$ – множина пікселів у зображені, а $\mathcal{C}$ – множина кольорів. Також маємо множину зсувів $D = \{0,1,...,D_{max}\}$. Також $h : I \times D \to \mathcal{R}$ – унарний штраф, $g : D \times D \to \mathcal{R}$ – бінарний штраф. Розв'язок задачі – послідовність $\overline{d} \in D^I$, така що:

$$E(\overline{d}) = \sum_{i=1}^{n} h(i, d_i) + \sum_{i=1}^{n-1} g(d_i, d_{i+1}). \tag{2.3}$$

Зводимо задачу до пошуку кращого шляху на графі $G' =< \mathcal{V}', \mathcal{E}' >$, як було описано у розділі (2.3).

Нехай є множина моментів часу $T = \{t_0, t_1, t_2, ...\}$, де кожний елемент t_k буде відповідати моменту надходження нової інформації.

Введемо функцію наявності вершини $o : \mathcal{V}' \times T \to \{0,1\}$. Будемо називати вершину $v \in \mathcal{V}'$ **відкритою** на момент часу t_k, якщо $o(v, t_k) = 1$, і будемо називати вершину **закритою** на момент часу t_k, якщо $o(v, t_k) = 0$. На початку $o(v, t_0) = 0, \forall v \in \mathcal{V}'$, що відповідає повній відсутності жодної інформації.

● «закрита» вершина

◯ «відкрита» вершина

Рисунок 2.5 — Типи вершин.

Тоді для кожного $t_k, k > 0$ якась нова вершина буде ставати відкритою, тобто:

$$\forall t_k \in T, k > 0, \exists v' \in V' : o(v', t_{k-1}) = 0, o(v', t_k) = 1.$$

В загальному випадку, ми не можемо знати які вершини коли нам відкриються. Добре якщо нам відомий порядок у якому вершини будуть ставати відкритими, і ще краще, коли ми знаємо що інформація буде надходити нам упорядковано. Адже тоді можливі більш елегантні та ефективні методи передобрахунку. Така ситуація буде описана у розділі (2.4.2).

І хоч значна частина методів передачі інформацію передає її упорядковано, можливі ситуації коли характер надходження даних заздалегідь невідомий. Саме таку ситуацію розглянуто у розділі (2.4.3). Також для неї запропоновано ефективний алгоритм розв'язання задачі для такого випадку. Він дозволяє проводити переобчислення, які значно прискорять розрахунки для отримання

остаточного результату після надходження всієї інформації.

2.4.2 Поступове надходження

Нехай нам відомо, що інформація буде надходити поступово і упорядковано. Коли нам відомий лише перший піксель кожного зображення (2.6) то граф G' матиме лише одну відкриту вершину $p1_{d0}$ (2.7).

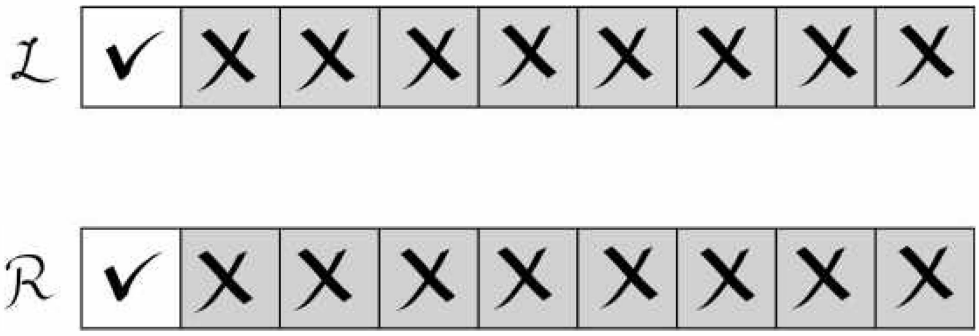


Рисунок 2.6 — Надходження даних

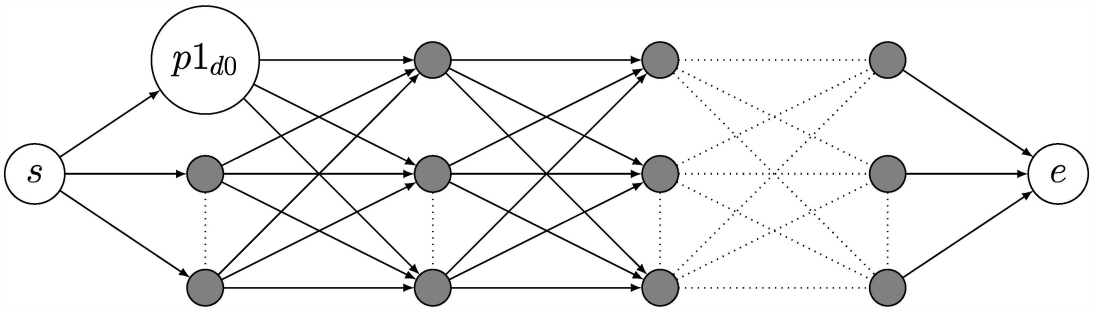


Рисунок 2.7 — Граф G' на момент часу t_1.

Коли перші два пікселі стануть нам відомі, то й вершини p^1_{d1} та p^2_{d0} стануть відкритими (2.8). Таким чином, при надходженні перших k пікселів обох зображень, в графі G' будуть відкриті вершини $p^i_{d_i}$, де $i \in I, i \leq k$, а $d_i \in D, 0 \leq d_i \leq \min(ki, D_{max})$. І тільки коли k буде більше за D_{max}, всі вершини в першому стовпчику будуть відкритими(2.9), і ми зможемо починати оптимізацію.

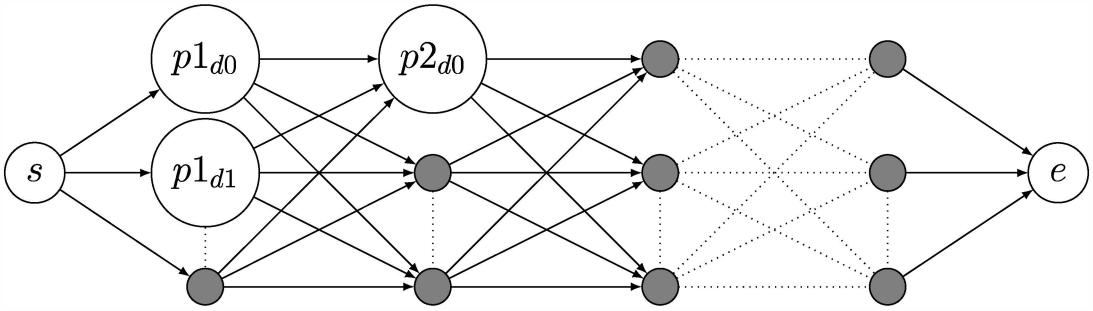


Рисунок 2.8 — Граф G' на момент часу t_2.

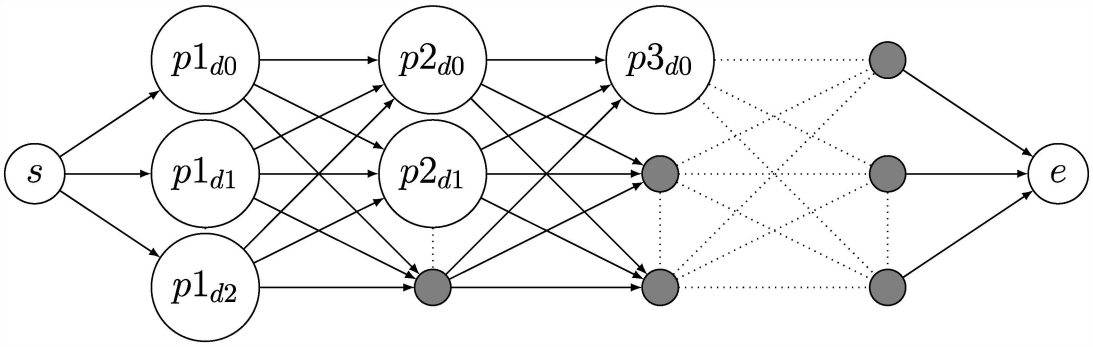


Рисунок 2.9 — Граф G' на момент часу t_3.

Тобто на момент часу t_{Dmax} у нас точно будуть відкритий весь перший стовпчик вершин (2.10). В такому випадку можемо його позбутися всіх вершин та ребер першого стовпчику, перебудувавши граф G' та перерахувавши вагу на нових дугах (2.11).

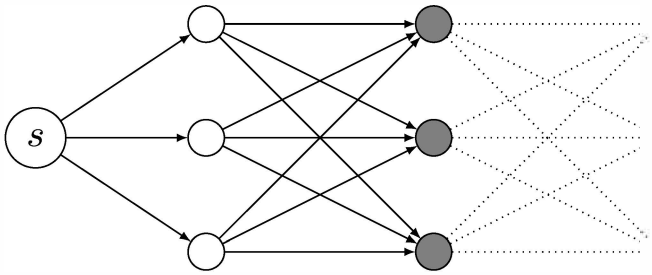


Рисунок 2.10 — Граф G' на момент часу t_{Dmax}.

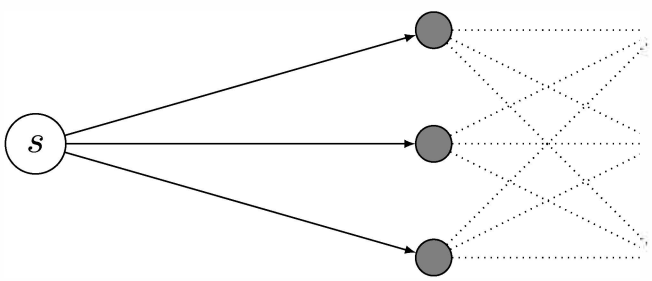


Рисунок 2.11 — Граф G' на момент часу t_{Dmax}.

Нехай хочемо позбутися вершин $p^0_{d'}, d' \in D$ у першому стовпчику, та відповідних цим вершинам ребер $(s, p^0_{d'})$ та $(p^0_{d'}, p^1_{d''}), d', d'' \in D$. Тоді замість них додаємо ребра $(s, p^1_{d''}), d'' \in D$, перераховуємо їх вагу, та запам'ятаємо кращу мітки для цього стовпчику:

$$w(s, p^1_{d''}) = \min_{d' \in D}(h(0, d') + g(d', d'')), \forall d'' \in D,$$
$$d^*_{0d''} = \arg\min_{d' \in D}(h(0, d') + g(d', d'')), \forall d'' \in D,$$

Так, на кожному наступному моменті часу t_l нам буде відкриватися новий стовпець вершин, для якого ми будемо повторювати таку процедуру. Таким чином, після надходження всієї інформації, у нас залишиться тільки початкова та кінцева вершина, що з'єднані одним ребром(2.12), вага якого буде відповідати довжині найкоротшого шляху через граф G'.

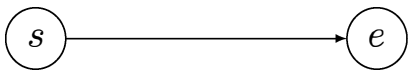


Рисунок 2.12 — Граф G' після надходження всіх даних.

Тепер для розв'язку задачі нам треба лише відновити послідовність d^*, за допомогою відповідних до кращого шляху міток $d^*_{nd''}$.

2.4.3 Хаотичне надходження

Для випадку, коли характер надходження даних заздалегідь невідомий теж можливі попередні обрахунки для прискорення процесу отримання кінцевого результату після надходження всієї інформації. Проте через непрогнозований характер надходження інформації запропонований онлайн алгоритм може використовувати більше пам'яті ніж його офлайн аналог. Проте, як і метод описаний у (2.4.2), запропонований онлайн алгоритм значно прискорює

отримання результату після надходження всіх даних.

Нехай на початку нам відомі тільки розмір зображень n та множина зсувів D, а як будуть надходити дані ми не знаємо. Будем вважати, що при кожному $t_k, k > 0$ нам буде відкриватися випадкова вершина у графі G' (2.14).

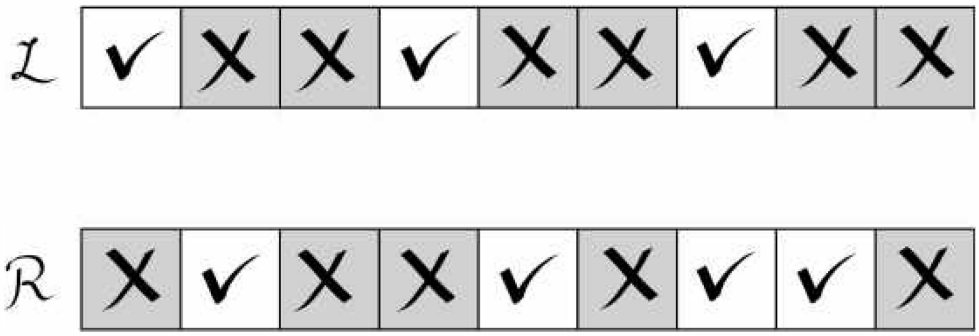


Рисунок 2.13 — Надходження даних

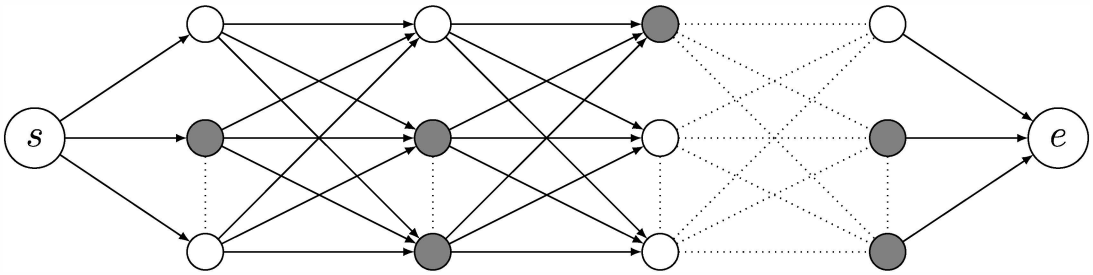


Рисунок 2.14 — Граф G' на момент часу t_k.

Основна ідея методу полягає в тому, щоб після відкриття нової вершини, перебудовувати граф G у граф G' так, щоб цієї вершини можна було б позбутися. Таким чином, після надходження всіх даних та відкриття всіх вершин, граф G' буде являти собою лише початкову та кінцеву вершини, що з'єднані одним ребром, вага якого і буде довжиною найкоротшого шляху через початковий граф G.

Для того, щоб правильно позбавитись якоїсь вершини, нам необхідно зберегти всі шляхи що проходять через цю вершину. Тобто замість ребер які ми вилучаємо разом із вершиною, треба додати нові ребра, вага яких містила б у собі і вагу вилучених ребер, і вагу вилученої вершини.

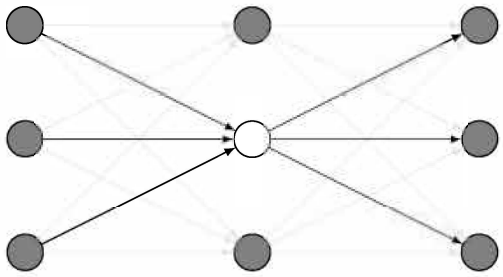


Рисунок 2.15 — До вилучення вершини

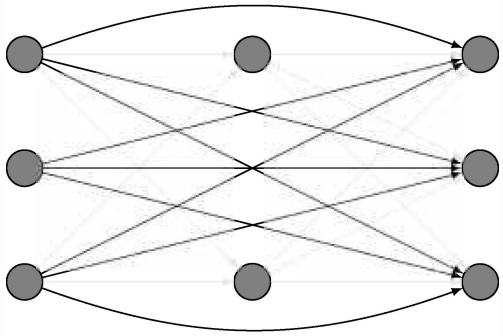


Рисунок 2.16 — Після вилучення вершини

Нехай, на кроці t маємо граф $G'^t =< \mathcal{V}'^t, \mathcal{E}'^t >$, і нам відкривається вершина $v^* \in \mathcal{V}'^t$. Для того, щоб її позбутися будуємо новий граф G'^{t+1}, для якого:

- $\mathcal{V}'^{t+1} = \mathcal{V}'^t \setminus \{v^*\}$

- $\mathcal{E}'^{t+1} = \mathcal{E}'^t \setminus \Big(\bigcup\limits_{(p,v^*) \in \mathcal{E}'^t} (p,v^*) \cup \bigcup\limits_{(v^*,q) \in \mathcal{E}'^t} (v^*,q) \Big)$

Якщо у графі G'^t ще не існує ребра $(p,q) \in \mathcal{E}'^t$, де $(p,v^*) \in \mathcal{E}'^t$ і $(v^*,q) \in \mathcal{E}'^t$, тоді додаємо таке ребро до графу G'^{t+1}:

1) $\mathcal{E}'^{t+1} \cup (p,q)$

2) Вага нового ребра:

$w^{t+1}(p,q) = w^t(p,v^*) + w^t(v^*) + w^t(v^*,q)$

Якщо ж у графі G'^t таке ребро (p,q) вже існує, тоді перерахуємо його

вагу як:

$$w^{t+1}(p,q) = \min\left\{w^t(p,v^*) + w^t(v^*) + w^t(v^*,q), w^t(p,q)\right\}$$

2.4.4 Найгірший випадок для випадку хаотичного надходження даних

Оскільки жодних обмежень на порядок надходження даних, а от же і на порядок відкриття вершин немає, розглянемо найгірший можливий випадок для запропонованого алгоритму.

Найгіршим випадком є ситуація, коли у кожному ”стовпчику” відкрито по одній вершині (рис. 2.17).

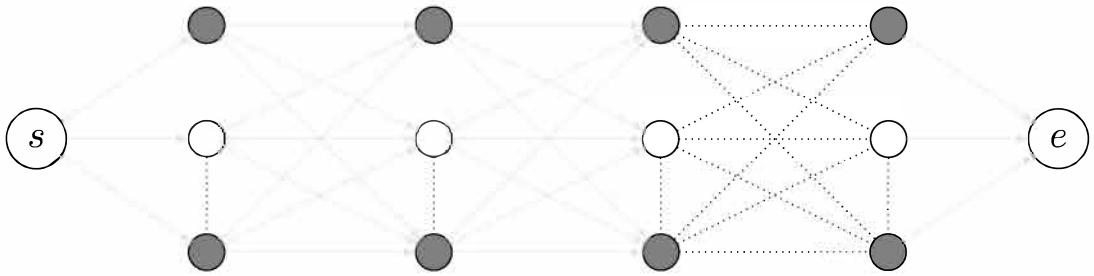


Рисунок 2.17 — Найгірша ситуація (а).

Адже тоді, кожна закрита вершина матиме додаткове ребро до кожної закритої вершини у наступних стовпчиках (рис. 2.18).

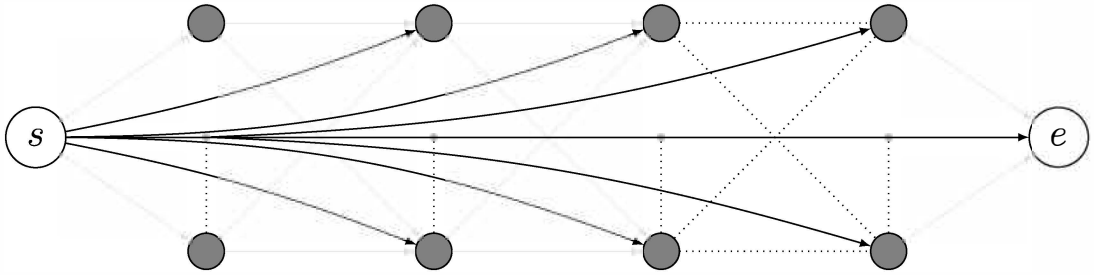


Рисунок 2.18 — Дуги однієї вершини.

У такій ситуації загальна кількість ребер не буде перевищувати $D^2_{max}n^2$.

До того ж будь-яка нова відкрита вершина буде тільки зменшувати загальну кількість ребер у графі.

Висновки до другого розділу

У другому розділі було детально розглянуто основні теоретичні результати досліджень попередніх робіт схожої тематики, необхідних для подальшого дослідження цієї теми.

Була поставлена задача розмітки для випадку повільного упорядкованого та хаотичного надходження даних. Було описано метод зведення цієї задачі до задачі пошуку найкоротшого шляху на орієнтованому зваженому графі спеціального вигляду, та метод поступової оптимізації такого графу, для значного прискорення процесу отримання кінцевого результату після отримання всієї інформації.

Постановка задачі розмітки для випадку повільного надходження даних, її зведення до задачі пошуку найкоротшого шляху на графі, та метод поступової оптимізації такого графу, для значного прискорення процесу отримання кінцевого результату після отримання всієї інформації буде узагальнена для більш широкого класу задач розмітки на деревах у наступному розділі.

3 ЗАДАЧА РОЗМІТКИ ДЛЯ ВИПАДКУ ПОСТУПОВОГО НАДХОДЖЕННЯ ДАНИХ

Третій розділ присвячено постановці задачі розмітки на деревах для випадку повільного надходження даних. Наведена постановка задачі розмітки на деревах для випадку повільного надходження даних. Пропонуються алгоритми розв'язання цієї задачі для повних даних та для випадку повільного надходження даних.

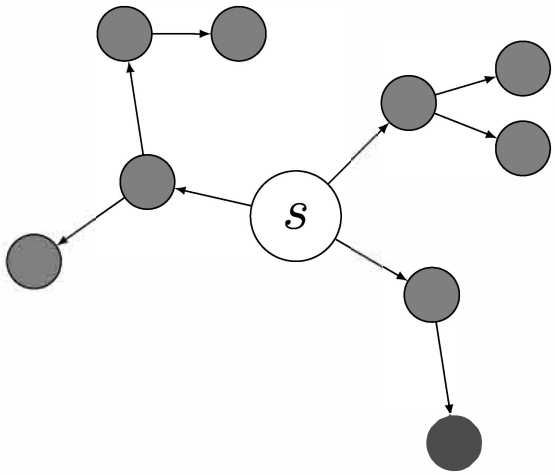


Рисунок 3.1 — Дерево G.

3.1 Постановка задачі

Сформулюємо задачу розмітки на деревах для повних даних. Нехай є деякий граф дерево $G =< \mathcal{V}, \mathcal{E} >$ (3.17), де $\mathcal{V}$ – множина вершин дерева, а $\mathcal{E}$ – множина ребер дерева. Нехай кожна вершина $v \in \mathcal{V}$ може мати мітку d_v із множини міток D. Введемо функцію розмітки вершини $l : \mathcal{V} \to D$. Розміткою дерева G будемо називати так послідовність $\overline{d} = \{d_1, d_2, ..., d_k, ...\}$, таку що:

$$\forall v_k \in \mathcal{V}\, \exists! d_k \in \overline{d} : l(v_k) = d_k. \tag{3.1}$$

Для того, щоб визначити якість розмітки $\overline{d}$ введемо функції унарної якості h та бінарної якості g:

$$h : \mathcal{V} \times D \to \mathcal{R},$$
$$g : D \times D \to \mathcal{R}.$$

Функція h буде відповідати за якість самої вершини, а функція g буде відповідати за якість сусідства та умовну ”гладкість” всієї розмітки загалом. Таким чином загальну якість, або ”енергією” E всього дерева G будемо визначати як суму якостей вершин та зв’язків між ними:

$$E = \sum_{p \in \mathcal{V}} h(p, d_p) + \sum_{(p,q) \in \mathcal{E}} g(d_p, d_q), \tag{3.2}$$

Задача розмітки графу G полягає у знаходженні такої оптимальної розмітки $d^* \in D^{\mathcal{V}}$, яка б мінімізувала (або максимувала) енергію графу G.

$$d^* \in \arg\min_{\overline{d} \in D^{\mathcal{V}}} \sum_{p \in \mathcal{V}} h(p, d_p) + \sum_{(p,q) \in \mathcal{E}} g(d_p, d_q), \tag{3.3}$$

Зведемо задачу до пошуку кращого шляху на графі спеціального вигляду $G' =< \mathcal{V}', \mathcal{E}' >$, як було описано у розділі (2.3).

Побудуємо орієнтований зважений граф $G' =< \mathcal{V}', \mathcal{E}' >$ відповідний до задачі (3.2). Кожній вершині v дерева G буде відповідати стовпчик із $|D|$ вершин у графі G', а кожному ребру у дереві G буде відповідати $|D|^2$ ребер у

графі G' таким чином, що:

$$v \in \mathcal{V} \implies v_d \in \mathcal{V}', \forall d \in D,$$
$$(v,u) \in \mathcal{E} \implies (v_{d1}, u_{d2}) \in \mathcal{E}', \forall d1 \in D, \forall d2 \in D.$$

Кожна вершина (ланка) (3.2) графу G "розкривається" у повнозв'язний стовпчик вершин (3.3) у графі G'.

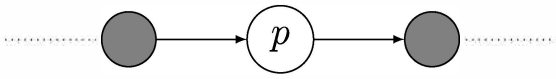


Рисунок 3.2 — Ланка у дереві G

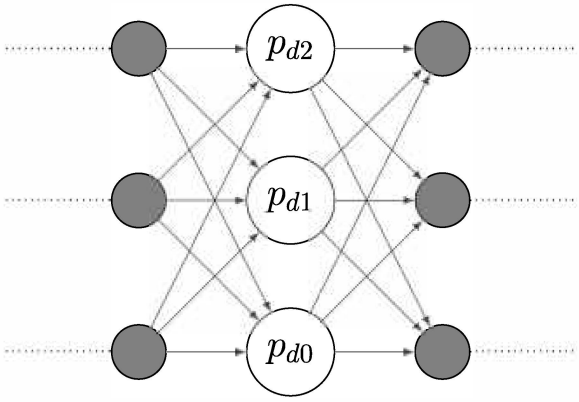


Рисунок 3.3 — Відповідна ланка у графі G'

Введемо функцію ваги вершини $w : \mathcal{V}' \to \mathcal{R}$. Кожній вершині $v_d \in \mathcal{V}'$ призначимо вагу

$$w(v_d) = h(v, d).$$

Введемо також функцію ваги ребер $w : \mathcal{E}' \to \mathcal{R}$. Кожному ребру $(v_{d1}, u_{d2}) \in \mathcal{E}'$ призначимо вагу

$$w(v_{d1}, u_{d2}) = g(v_{d1}, u_{d2}).$$

Тоді зважена ланка (3.3) буде виглядати як на рисунку (**??**).

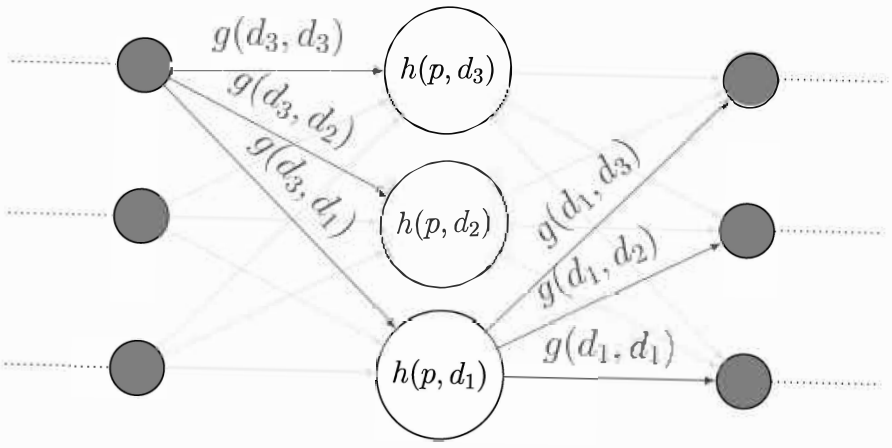


Рисунок 3.4 — Ваги ланки у графі G'

Оберемо в кожному стовпчику виберемо по одній вершині v_{d^*} (3.18), та назвемо множину таких вершин $\hat{\Delta}$.

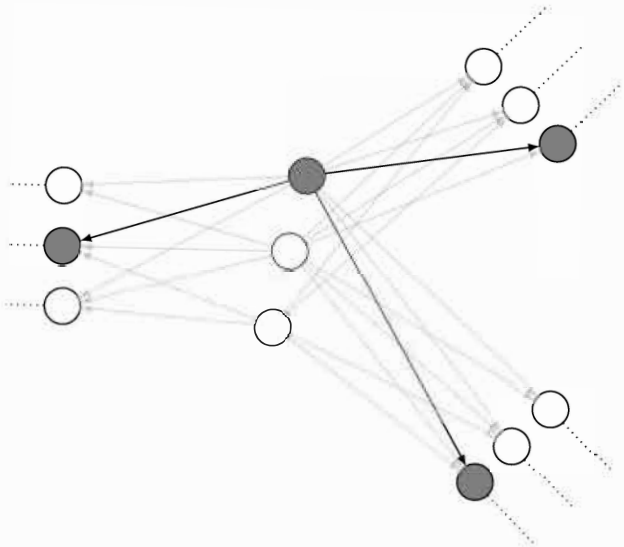


Рисунок 3.5 — Шлях на G'.

Множина таких вершин буде відповідати деякій розмітці $\bar{d}$. А енергія дерева G з такою розміткою $\bar{d}$, буде відповідати довжині шляху, проведеного через обрані вершини і буде дорівнювати

$$E(\bar{d}) = \sum_{p_d \in \hat{\Delta}} h(p, d_p) + \sum_{\substack{(p_d, q_{d'}) \in \mathcal{E}' \\ p_d, q_{d'} \in \hat{\Delta}}} g(d_p, d_q), \tag{3.4}$$

Виходить, що для того щоб знайти оптимальну розмітку d^* графу G, необхідно знайти таку послідовність вершин у графі G', яка відповідає кращому шляху через граф G'.

3.2 Онлайн алгоритм пошуку розмітки

Нехай є множина моментів часу $T = \{t_0, t_1, t_2, ...\}$, де кожний елемент t_k буде відповідати моменту надходження нової інформації.

Введемо функцію наявності вершини $o : \mathcal{V}' \times T \to \{0,1\}$. Будемо називати вершину $v \in \mathcal{V}'$ **відкритою** на момент часу t_k, якщо $o(v, t_k) = 1$, і будемо називати вершину **закритою** на момент часу t_k, якщо $o(v, t_k) = 0$. На початку $o(v, t_0) = 0, \forall v \in \mathcal{V}'$, що відповідає повній відсутності жодної інформації.

● «закрита» вершина

○ «відкрита» вершина

Рисунок 3.6 — Типи вершин.

Нехай на початку нам відомі тільки структура дерева G та множина міток D, а всі вершини цього дерева закриті. Будем вважати, що при кожному $t_k, k > 0$ нам буде відкриватися випадкова вершина у графі G' (2.14).

Основна відмінність задачі розмітки на деревах від задачі розмітки на ланцюгах полягає у наявності розгалужень (3.7).

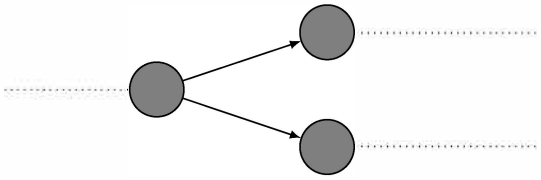


Рисунок 3.7 — Розгалуження у дереві G

При кожній новій відкритій вершині можлива одна з наступних ситуацій.

3.2.1 Випадок А

Випадок, коли у нової відкритої вершини v^* дерева G рівно дві сусідні закриті вершини (3.8). Для такої ситуації отпимизація проходить так само, як і для задачі розмітки на ланцюжку (2.4).

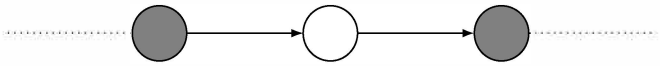


Рисунок 3.8 — Видадок А - Дерево G

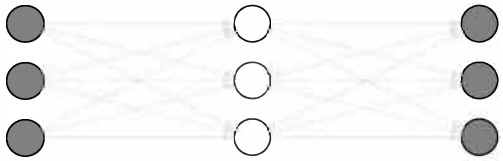


Рисунок 3.9 — До вилучення вершини.

Рисунок 3.10 — Після вилучення вершини.

Для того, щоб правильно позбавитись якоїсь вершини, нам необхідно зберегти всі шляхи що проходять через цю вершину. Тобто замість ребер які ми вилучаємо разом із вершиною, треба додати нові ребра, вага яких містила б у собі і вагу вилучених ребер, і вагу вилученої вершини.

Нехай, на кроці t маємо граф $G'^t =< \mathcal{V}'^t, \mathcal{E}'^t >$, і нам відкриваються вершини $v^*_{di} \in \mathcal{V}'^t, di \in D$. Для того, щоб їх позбутися будуємо новий граф G'^{t+1}, для якого:

- $\mathcal{V}'^{t+1} \setminus \{v^*_{di}\}, \forall di \in D$

- $\mathcal{E}'^{t+1} \setminus \Big(\bigcup\limits_{(p,v^*_{di}) \in \mathcal{E}'^t} (p,v^*_{di}) \cup \bigcup\limits_{(v^*_{di},q) \in \mathcal{E}'^t} (v^*_{di},q) \Big), \forall di \in D$

- $\mathcal{E}'^{t+1} \cup \{(p,q) \mid (p,v^*_{di}) \in \mathcal{E}'^t, (v^*_{di},q) \in \mathcal{E}'^t \forall di \in D\}$

Вага нових ребер:

$$\forall p,q \in \mathcal{V}', (p, v^*_{di}) \in \mathcal{E}', (v^*_{di},q) \in \mathcal{E}' :$$
$$w^{t+1}(p,q) = \min_{di \in D}(w^t(p, v^*_{di}) + w^t(v^*_{di}) + w^t(v^*_{di},q)).$$

3.2.2 Випадок Б

Випадок, коли у нова відкрита вершина v^* дерева G є листом (3.11).

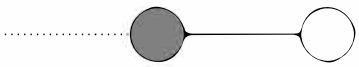


Рисунок 3.11 — Видадок Б - Дерево G

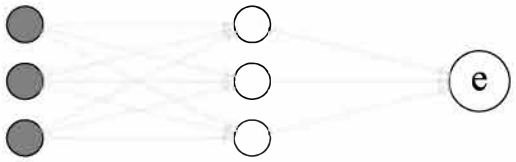


Рисунок 3.12 — До вилучення вершин.

Рисунок 3.13 — Після вилучення вершини.

Для того, щоб правильно позбавитись якоїсь вершини, нам необхідно зберегти всі шляхи що проходять через цю вершину. Тобто замість ребер які ми вилучаємо разом із вершиною, треба додати нові ребра, вага яких містила б у собі і вагу вилучених ребер, і вагу вилученої вершини.

Нехай, на кроці t маємо граф $G'^t =< \mathcal{V}'^t, \mathcal{E}'^t >$, і нам відкриваються вершини $v^*_{di} \in \mathcal{V}'^t, di \in D$. Для того, щоб їх позбутися будуємо новий граф G'^{t+1}, для якого:

- $\mathcal{V}'^{t+1} \setminus \{v^*_{di}\}, \forall di \in D$

- $\mathcal{E}'^{t+1} \setminus \Big(\bigcup\limits_{(p,v^*_{di}) \in \mathcal{E}'^t} (p,v^*_{di}) \cup \bigcup\limits_{(v^*_{di},q) \in \mathcal{E}'^t} (v^*_{di},q) \Big), \forall di \in D$

- $\mathcal{E}'^{t+1} \cup \{(p,q) \mid (p,v^*_{di}) \in \mathcal{E}'^t, (v^*_{di},q) \in \mathcal{E}'^t \forall di \in D\}$

Вага нових ребер:

$$\forall p,q \in \mathcal{V}', (p,v^*_{di}) \in \mathcal{E}', (v^*_{di},q) \in \mathcal{E}' :$$
$$w^{t+1}(p,q) = \min_{di \in D}(w^t(p,v^*_{di}) + w^t(v^*_{di}) + w^t(v^*_{di},q)).$$

3.2.3 Випадок B

Випадок, коли у нової відкритої вершини v^* дерева G рівно дві сусідні закриті вершини (3.14).

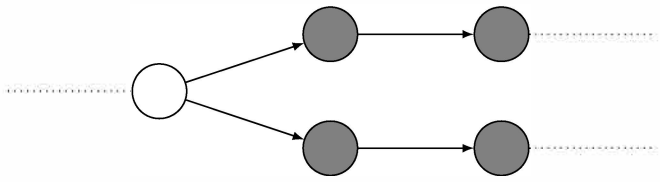


Рисунок 3.14 — Видадок B - Дерево G

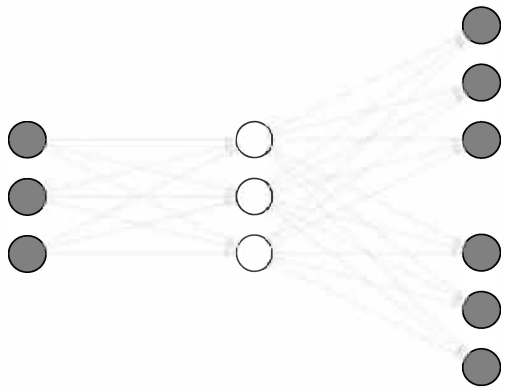


Рисунок 3.15 — До вилучення вершини.

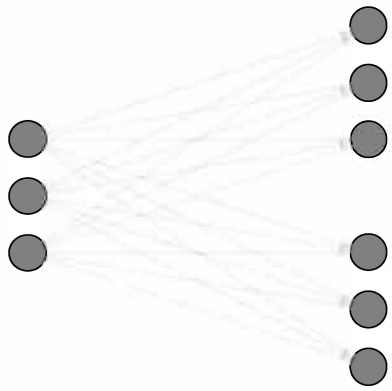


Рисунок 3.16 — Після вилучення вершини.

Для того, щоб правильно позбавитись якоїсь вершини, нам необхідно зберегти всі шляхи що проходять через цю вершину. Тобто замість ребер які ми вилучаємо разом із вершиною, треба додати нові ребра, вага яких містила б у собі і вагу вилучених ребер, і вагу вилученої вершини.

Нехай, на кроці t маємо граф $G'^t =< \mathcal{V}'^t, \mathcal{E}'^t >$, і нам відкриваються вершини $v^*_{di} \in \mathcal{V}'^t, di \in D$. Для того, щоб їх позбутися будуємо новий граф G'^{t+1}, для якого:

- $\mathcal{V}'^{t+1} \setminus \{v^*_{di}\}, \forall di \in D$

- $\mathcal{E}'^{t+1} \setminus \Big(\bigcup\limits_{(p,v^*_{di}) \in \mathcal{E}'^t} (p,v^*_{di}) \cup \bigcup\limits_{(v^*_{di},q) \in \mathcal{E}'^t} (v^*_{di},q) \Big), \forall di \in D$

- $\mathcal{E}'^{t+1} \cup \{(p,q) \mid (p,v^*_{di}) \in \mathcal{E}'^t, (v^*_{di},q) \in \mathcal{E}'^t \forall di \in D\}$

Вага нових ребер:

$$\forall p,q \in \mathcal{V}', (p,v^*_{di}) \in \mathcal{E}', (v^*_{di},q) \in \mathcal{E}' :$$
$$w^{t+1}(p,q) = \min_{di \in D}(w^t(p,v^*_{di}) + w^t(v^*_{di}) + w^t(v^*_{di},q)).$$

3.2.4 Отримання результату

Після надходження всіх даних, в результаті запропонованої оптимізації дерево G буде мати наступний вигляд:

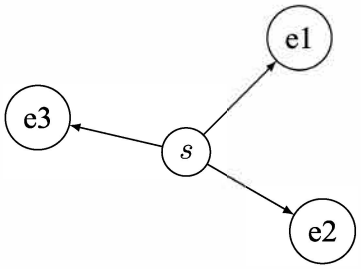


Рисунок 3.17 — Дерево G після надходження всієї інформації.

Тепер для пошуку кращого шляху на початковому дереві G необхіно всього лиш обрати таку v_{d^*}, для якої:

$$d^* \in \arg\min_{d \in D}(w(e_1, v_d) + w(e_2, v_d) + w(e_3, v_d))$$

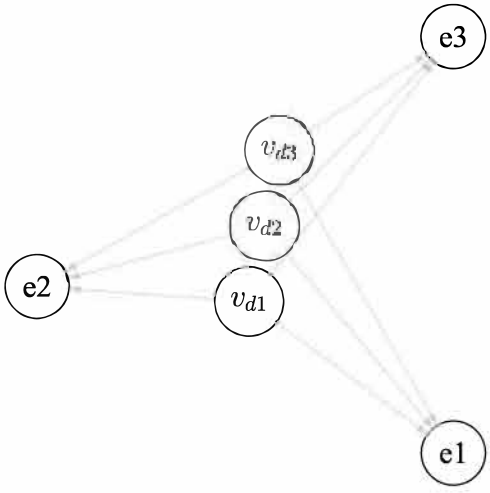


Рисунок 3.18 — Фінальний G'.

Висновки до другого розділу

У третьому розділі було розглянуто задачу розмітки на деревах для ситуації повільного надходження даних.

Була поставлена задача розмітки на деревах для випадку повільного упорядкованого. Було описано метод зведення цієї задачі до задачі пошуку найкоротшого шляху на орієнтованому зваженому графі спеціального вигляду, та метод поступової оптимізації такого графу, для значного прискорення процесу отримання кінцевого результату після отримання всієї інформації.

Програмна реалізація алгоритмів розв’язання задачі розмітки на деревах для та порівняння online та offline підходів буде представлено у наступному розділі.

4 ПРАКТИЧНІ РЕЗУЛЬТАТИ

У четвертому розділі представлена програмна реалізація алгоритмів розв'язання задачі розмітки на деревах для повних даних та для повільного надходження даних. Також представлено порівняння часу роботи online та offline алгоритмів для повних даних, та для ситуації повільного надходження даних.

4.1 Умови тестування

У якості конкретної задачі розмітки було взято задачу стереозору, з такими штрафними функціями та параметрами

- $h(i,d) = |\mathcal{L}(i) - \mathcal{R}(i-d)|$
- $g(d, d') = \alpha|d - d'|$
- $D_{max} = 100$.
- $\alpha = 10$.

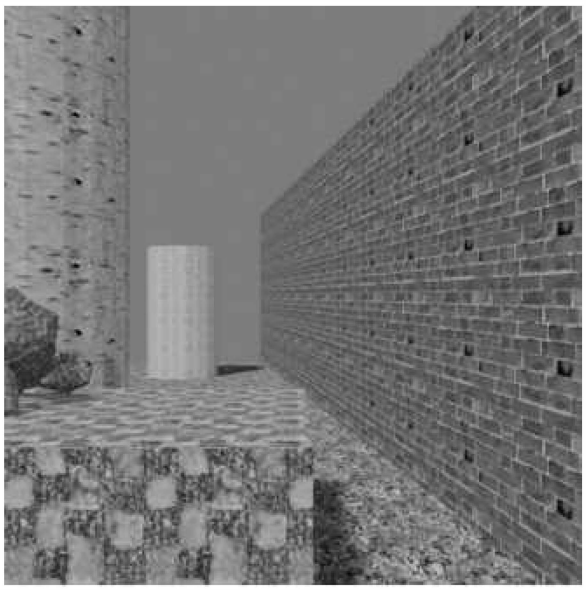
Тестування проходило на зображеннях з **Middlebury Stereo Datasets**.

Обидва алгоритми було реалізовано мовою **C++** , з використанням бібліотеки **OpenCV**, для зручної роботи з зображеннями.

4.2 Порівняння результатів роботи

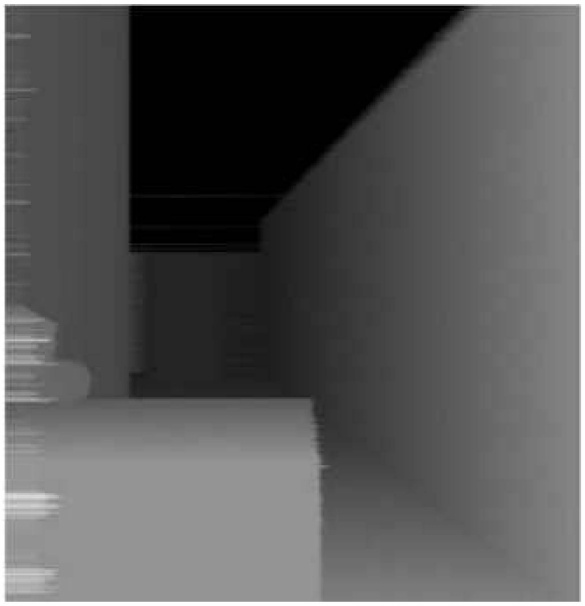
Порівняння результатів роботи online та offline алгоритмів для повних даних, та для ситуації повільного надходження даних (4.1).

Результати експерименту підтверджують теоретичні здогадки – обидва алгоритми дають однаковий результат. Це добре, адже це значить що, запропонований алгоритм дає розв'язок задачі розмітки не гірший аніж звичайний алгоритм.

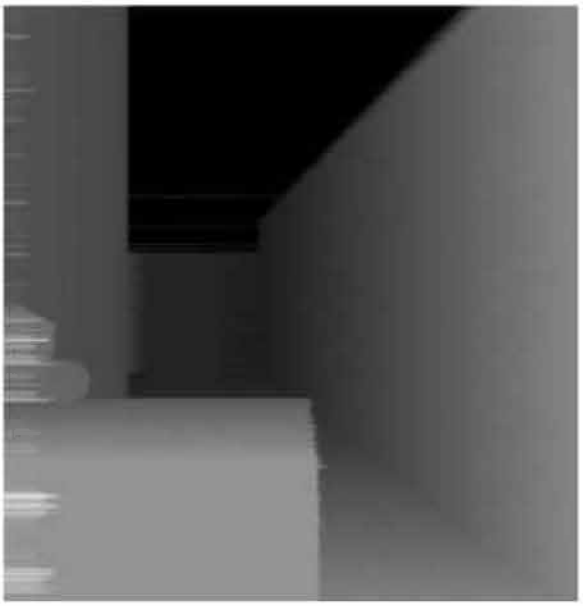


(а) Ліве вхідне зображення

(б) Праве вхідне зображення



(в) Результат роботи алгоритму для повних даних



(г) Результат роботи алгоритму для повільного надходження даних

Рисунок 4.1 — Порівняння результатів роботи

4.3 Порівняння швидкості роботи

4.3.1 Порівняння швидкості роботи при повних даних

Порівняння швидкості роботи online та offline алгоритмів для повних даних, та для ситуації повільного надходження даних.

Порівняння швидкості роботи для випадку повних даних(4.1). Вся ін-

формація наявна на момент початку розрахунків. Бачимо очікуваний резуль-

При повних даних	
$T_{offline}$	6.2c
T_{online}	7.8c

Таблиця 4.1 — Час роботи при повних даних

тат. Через те, що online алгоритм загалом виконує більшу кількість операцій (додає умовно ”зайві” ребра), для випадку повних даних online алгоритм розв’язку задачі розмітки є менш ефективним ніж звичайний.

4.3.2 Порівняння швидкості роботи для випадку повільного надходження даних

Порівняння швидкості роботи для випадку повільного надходження даних(4.2). Інформація надходить із затримкою. Час між інтервалами надходження даних $\hat{t} = 10^{-4}c$. Відлік часу після надходження останньої частини даних. Результати експерименту підтверджують теоретичні здогадки – offline алго-

При повільних даних	
$T_{offline}$	6.2c
T_{online}	0.08c

Таблиця 4.2 — Час роботи для ситуації повільного надходження даних.

ритм не міг почати роботу поки вся інформація не була доступна, тому час роботи в нього такий же, як і в (4.1). А online алгоритму, завдяки виконанню передобрахунків над поступаючою інформацією, вдалося значно зменшити час необхідний для отримання результату після надходження всіх даних.

Висновки до четвертого розділу

У четвертому розділі було проаналізовано результати програмної реалізації online та offline алгоритмів розв'язання задачі розмітки для повних даних та ситуації повільного надходження даних.

Було проведено порівняння результатів роботи обох алгоритмів. Також було проведено порівняння часу роботи обох алгоритмів для ситуації повних даних та ситуації повільного надходження даних.

Результати експерименту підтверджують теоретичні здогадки.

ВИСНОВКИ

В даній роботі було розглянуто задачу розмітки на деревах для випадку повільного надходження даних, її зведення до задачі пошуку найкоротшого шляху на орієнтованому зваженому графі спеціального вигляду та offline алгоритм розв'язання цієї задачі. Для випадку повільного надходження даних було розроблено online алгоритми, який для цих випадків, є ефективніши-ми за offline алгоритм. Було створено програмна реалізація цих алгоритмів, та проведено порівняння швидкості їх роботи для випадку повних даних, та для випадку повільного надходження даних. Практичні результати підтверди-ли теоретичні здогадки: для випадку повних даних offline алгоритм є більш ефективним, а для випадків повільного надходження даних більш ефективни-ми є запропонований online алгоритми.

БІБЛІОҐРАФІЯ

1 Кириленко І.А. *Задача стерео-зору в умовах хаотичного надходження даних*, Науково-практична конференція студентів, аспірантів та молодих вчених, 2022.

2 Кириленко І.А. *Задача стерео-зору в умовах поступового надходження даних*, Науково-практична конференція студентів, аспірантів та молодих вчених, 2020. – 58-60с.

3 G. FACCIOLO, C. DE FRANCHIS, E. MEINHARDT, *A Significantly More Global Matching for Stereovision*.

4 Michail Schlesinger, Vaclav Hlavac, *Ten Lectures on Statistical and Structural Pattern Recognition*, 2002.

5 Middlebury Stereo Datasets. *http://vision.middlebury.edu/stereo/data/* [Електронний ресурс].

6 Hirschmuller, Heiko., *Evaluation of Cost Functions for Stereo Matching*, IEEE Conference on Computer Vision and Pattern Recognition, 2007.

7 G. Bradski, *The OpenCV Library*, Dr. Dobb's Journal of Software Tools, 2000.

8 Daniel Scharstein, Chris Pal, *Learning Conditional Random Fields for Stereo*, CVPR, 2007.

9 Chai, Yu Yang, Fei, *Semi-Global Stereo Matching Algorithm Based on Minimum Spanning Tree*, IMCEC.2018.8469306. , CVPR, 2018.

ДОДАТОК А

Код програми

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include <limits>
#include "functions.h"

int WindowStartX = 500, WindowStartY = 100, WindowMargin = 10, Scale = 1;

int main()
{
Mat imL = imread("../imgs/3d/view1.png", IMREAD_GRAYSCALE);
Mat imR = imread("../imgs/3d/view2.png", IMREAD_GRAYSCALE);
int width = imL.cols, height = imL.rows;
int MAX_DISP = 100; //< 255
Mat depth_map = Mat(height, width, CV_8U);


//For one row
for (int row = 0; row < height; row++)
{
        Mat L = getRow(row, imL);
        Mat R = getRow(row, imR);
        Mat P = initPrevMatrix(MAX_DISP + 1, width);
        Mat W = initWeightsMatrix(MAX_DISP + 1);
        Mat W2 = initWeightsMatrix(MAX_DISP + 1);
        Mat G = initBinaryPenalty(MAX_DISP, 10);
        Mat H = Mat(MAX_DISP + 1, width, CV_32S);

        //--Computing PrevMatrix--
        for (int t = 1; t < width; t++)
        {
                for (int d = 0; d < MAX_DISP + 1; d++)
                {
                        int temp_sum;
                        int  Wmin = std::numeric_limits<int>::max();
                        for (int b = 0; b < MAX_DISP + 1; b++)
                        {
                                //
                                if (b > t)
                                        H.at<int>(b, t) = std::numeric_limits<int>::infinity();
                                else
                                        H.at<int>(b, t) = abs(L.at<uchar>(0, t) - R.at<uchar>(0, t - b));
                                //
                                if (H.at<int>(b, t) == std::numeric_limits<int>::infinity())
                                {
                                        temp_sum = std::numeric_limits<int>::infinity();
                                        //std::cout << "H backfired" << std::endl;
                                }
```

```
48                                  /*
49                                  else if (W.at<int>(b, 0) == std::numeric_limits<int>::infinity())
50                                  {
51                                          temp_sum = std::numeric_limits<int>::infinity();
52                                          //std::cout << "W backfired" << std::endl;
53                                  }
54
55                                  else if (G.at<int>(d, b) == std::numeric_limits<int>::infinity())
56                                  {
57                                          temp_sum = std::numeric_limits<int>::infinity();
58                                          std::cout << "G backfired" << std::endl;
59                                  }
60                                  */
61                                  else
62                                          temp_sum = W.at<int>(b, 0) + H.at<int>(b, t) + G.at<int>(d, b);
63
64                                  //
65                                  if (temp_sum < Wmin)
66                                  {
67                                          Wmin = temp_sum;
68                                          W2.at<int>(d, 0) = Wmin;
69                                          P.at<uchar>(d, t) = b;
70                                  }
71                          }
72                          W = W2;
73                  }
74          }
75
76
77          //--Restore depth_string--
78          Mat depth_string = Mat(1, width, CV_8U);
79          //Finding d_n
80          int temp_sum, d, n = width - 1;
81          int  Wmin = std::numeric_limits<int>::max();
82          for (int b = 0; b < MAX_DISP + 1; b++)
83          {
84                  temp_sum = W.at<int>(b, 0) + h(n, b, L, R);
85                  if (temp_sum < Wmin)
86                  {
87                          Wmin = temp_sum;
88                          d = b;
89                  }
90          }
91          //Computing depth_string
92          depth_string.at<uchar>(0, n) = d;
93          for (int i = n - 1; i > 0; i--)
94          {
95                  int k = static_cast<int>(depth_string.at<uchar>(0, i + 1));
96                  depth_string.at<uchar>(0, i) = P.at<uchar>(k, i + 1);
97          }
98
```

```
99
100         //Writing data to depth_map
101         for (int i = 0; i < width; i++)
102         {
103                 depth_map.at<uchar>(row, i) = depth_string.at<uchar>(0, i);
104         }
105         std::cout << "[ " << row << " / " << height << " ] \r";
106 }
107
108
109
110
111
112
113 namedWindow("Window1", WINDOW_FREERATIO);
114 imshow("Window1", depth_map);
115 //resizeWindow("Window1", width * Scale, height * Scale);
116 //moveWindow("Window1", WindowStartX, WindowStartY);
117 waitKey();
118 }
```

ДОДАТОК Б

Код програми

```
#pragma once
#include "opencv2/opencv.hpp"
using namespace cv;

int h(int i, int d, Mat L, Mat R);
int g(int d, int b, int alpha = 1);

Mat initBinaryPenalty(int MAX_DISP, float alpha);
Mat initPrevMatrix(int height, int width);
Mat initWeightsMatrix(int MAX_DISP);
Mat getRow(int row, Mat im);
```

ДОДАТОК В

Код програми

```
#include "functions.h"
#include <iostream>


int h(int i, int d, Mat L, Mat R)
{
    if (d > i)
        return abs(L.at<uchar>(0, i) - R.at<uchar>(0, 0));
    else
        return abs(L.at<uchar>(0, i) - R.at<uchar>(0, i - d));
}


int g(int d, int b, int alpha)
{
    return alpha * abs(d - b);
}

Mat initBinaryPenalty(int MAX_DISP, float alpha)
{
    Mat g = Mat(MAX_DISP + 1, MAX_DISP + 1, CV_32S);
    int maxdi = g.cols;
    for (int di = 0; di < maxdi; di++)
    {
        for (int dj = 0; dj < maxdi; dj++)
        {
            g.at<int>(di, dj) = alpha * abs(di - dj);
        }
    }

    return g;
}


Mat initPrevMatrix(int height, int width)
{
    Mat P = Mat(height, width, CV_8U);
    for (int i = 0; i < height; i++)
    {
        for (int j = 0; j < width; j++)
        {
            P.at<uchar>(i, j) = 0;
        }
        //P.at<uchar>(i, 0) = 0;
    }
    return P;
}
```

```
48
49
50  Mat initWeightsMatrix(int height)
51  {
52      Mat W = Mat(height, 1, CV_32S);
53      for (int d = 0; d < height; d++)
54      {
55          W.at<int>(d, 0) = 0;
56      }
57      return W;
58  }
59
60
61  Mat getRow(int row, Mat im)
62  {
63      Mat R = Mat(1, im.cols, CV_8U);
64      for (int i = 0; i < im.cols; i++)
65      {
66          R.at<uchar>(0, i) = im.at<uchar>(row, i);
67      }
68      return R;
69  }
```