НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра математичного моделювання та аналізу даних

До захисту допущено:
Завідувач кафедри
_____ Наталія КУССУЛЬ
«___»_____2022 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Математичні методи моделювання, розпізнавання образів та безпеки даних» спеціальності: 113 «Прикладна математика»
на тему: «Побудова алгоритму Expected Semi-Global Matching для знаходження розв'язку задачі стереозору»

Виконав: здобувач вищої освіти IV курсу, групи ФІ-82
Хмелевський Святослав Олександрович _____
Керівник: к.т.н., с.н.с., асистент кафедри ММАД Водолазський Є.В. _____
Консультант: м.н.с. Кригін В.М. _____
Рецензент: к.т.н., с.н.с., зав. відділом Мацелло В.В. _____

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

Київ – 2022

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра математичного моделювання та аналізу даних

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 113 «Прикладна математика»

Освітня програма «Математичні методи моделювання, розпізнавання образів та безпеки даних»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Наталія КУССУЛЬ
«__» _____ 2022 р.

ЗАВДАННЯ
на дипломну роботу

Студент: Хмелевський Святослав Олександрович

1. Тема роботи: *«Побудова алгоритму Expected Semi-Global Matching для знаходження розв'язку задачі стереозору»*,
керівник: к.т.н., с.н.с., асистент кафедри ММАД Водолазський Є.В.,
затверджені наказом по університету №__ від «__» _____ 2022р.

2. Термін подання студентом роботи: «__» _____ 2022р.

3. Вихідні дані до роботи: *Мапи зсувів тестових стереопар*

4. Зміст роботи: *огляд літератури та публікацій за темою дослідження; реалізація наявних алгоритмів розв'язування задачі стереозору; побудова та реалізація алгоритму Expected Semi-Global Matching; порівняльний аналіз роботи реалізованих алгоритмів*

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): *Презентація доповіді*

6. Дата видачі завдання: 10 вересня 2021 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання	Примітка
1	Узгодження теми роботи із науковим керівником	01-15 вересня 2021 р.	Виконано
2	Огляд опублікованих джерел за тематикою дослідження	Вересень-жовтень 2021 р.	Виконано
3	Реалізація наявних алгоритмів розв’язування задачі стереозору	Листопад-грудень 2021 р.	Виконано
4	Проектування та дослідження алгоритму Expected Semi-Global Matching	Січень-березень 2022 р.	Виконано
5	Отримання результатів роботи реалізованих алгоритмів	Квітень 2022 р.	Виконано
6	Аналіз отриманих результатів і оформлення роботи	Травень 2022 р.	Виконано

Студент _____ Хмелевський С.О.

Керівник _____ Водолазський Є.В.

РЕФЕРАТ

Кваліфікаційна робота містить 54 стор., 6 рисунків, 2 таблиці, 21 джерело.

Метою роботи була побудова та програмна реалізація алгоритму Expected Semi-Global Matching, яка є модифікацією відомого в області комп'ютерного зору алгоритму SGM. Об'єктом дослідження є алгоритми оцінки мапи зсувів. Предметом дослідження є дискретна оптимізація на графових структурах.

В процесі виконання дипломної роботи був здійснений огляд публікацій та літератури за тематикою задачі стереозору та методів її розв'язування. Також в роботі міститься детальний опис деяких алгоритмів з літератури, які надалі використовувалися на етапі тестування.

Здійснений детальний опис побудови алгоритму Expected Semi-Global Matching. Всі алгоритми, які містяться в роботі були реалізовані. В кінці роботи знаходиться порівняння реалізованих алгоритмів.

Ключові слова: КОМП'ЮТЕРНИЙ ЗІР, БІНОКУЛЯРНИЙ СТЕРЕОЗІР, ГРАФОВІ МОДЕЛІ, ДИНАМІЧНЕ ПРОГРАМУВАННЯ, ДИСКРЕТНА ОПТИМІЗАЦІЯ.

ABSTRACT

Qualification work contains 54 pages, 6 figures, 2 tables, 21 references.

The main goal of this thesis was to develop Expected Semi-Global Matching algorithm that is a modification of the famous in computer vision SGM algorithm. The object of research is algorithms for evaluating disparity maps. The subject of research is discrete optimization on graph structures.

In the process of completing the thesis, a review of publications and literature on the stereo matching problem and the methods for solving it was made. You can also find in this thesis a detail description of some of the algorithms from the literature, which were then used in testing.

A detail description of the process of developing an Expected Semi-Global Matching algorithm was made. All algorithms that were mentioned in the thesis were implemented. In the end of the thesis you can find the comparison of implemented algorithms.

Key words: COMPUTER VISION, BINOCULAR STEREOVISION, GRAPHICAL MODELS, DYNAMIC PROGRAMMING, DISCRETE OPTIMIZATION.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

$\mathbb{1}_A(x)$ — індикатор належності x множині A

$\|\cdot\|$ — норма визначена на деякому лінійному просторі

$\|\cdot\|_2$ — евклідова норма

ВСТУП

Актуальність роботи. Задача стереозору є однією з найбільш важливих і досліджених тем в області комп'ютерного зору. Алгоритми та методи, розроблені в процесі розв'язання відповідної задачі, широко застосовуються при обробці супутникових зображень, в 3D реконструкції об'єктів, в роботехніці, при розробці систем допомоги водію тощо [1]. Задача стереозору полягає в побудові мапи зсувів для заданої стереопари — пари зображень певної сцени, які були зроблені під різними кутами. Отримані результати можуть бути використані надалі, зокрема, для побудови мапи глибин. Є безліч алгоритмів для розв'язку наведеної задачі, наприклад, TRW-S (Sequential Tree-Reweighted algorithm) [2] і його модифікації [3]. Надалі ми зосередимося на алгоритмі SGM (Semi-Global Matching).

SGM є одним з найважливіших алгоритмів в області комп'ютерного зору. Його перевагою є компроміс між швидкістю роботи алгоритму та якістю отриманих результатів у процесі розв'язку задачі стереозору. Швидкість роботи алгоритму відіграє велику роль, оскільки на практиці (наприклад, у системах допомоги водію) потрібно обробляти дані в режимі реального часу. Затримка у часі в таких випадках може дорого комусь обійтися. Водночас дослідники в цій області прагнуть отримати якомога якісніші результати з тих самих міркувань. Саме тому компроміс між швидкістю й якістю результату є дуже важливою властивістю SGM.

Мета і завдання дослідження.

Об'єктом дослідження є алгоритми оцінки мапи зсувів.

Предметом дослідження є дискретна оптимізація на графах.

Метою дослідження є побудова модифікації алгоритму SGM, яка мінімізує середньоквадратичне відхилення від еталонів, і порівняння її з іншими методами розв'язку задачі стереозору. Для досягнення мети потрібно виконати завдання:

1) здійснити огляд літератури, публікацій на тему задачі стереозору;
2) реалізувати наявні алгоритми розв’язування задачі стереозору;
3) побудувати та реалізувати алгоритм Expected SGM;
4) здійснити аналіз порівняння роботи реалізованих алгоритмів.

Наукова новизна одержаних результатів полягає в побудові й дослідженні алгоритму розв’язування задачі стереозору, який є узагальненням SGM на випадок пошуку мапи зсуву у вигляді математичного сподівання, а не найбільш ймовірної розмітки. Таке узагальнення має давати кращі візуально результати, зберігаючи властивість SGM компромісу між швидкістю роботи і їхньою якістю.

Апробація результатів. Доповідь на тему «Побудова алгоритму EXPECTED SEMI-GLOBAL MATCHING для знаходження розв’язку задачі стереозору» на XX Всеукраїнській науково-практичній конференції студентів, аспірантів та молодих вчених «Теоретичні і прикладні проблеми фізики, математики та інформатики» (15 – 16 червня 2022 р., м. Київ, Україна).

1 ТЕОРЕТИЧНІ ВІДОМОСТІ ТА ОГЛЯД ЛІТЕРАТУРИ

Цей розділ містить всі необхідні теоретичні відомості, які знадобляться в наступному розділі при побудові нових модифікацій алгоритму SGM для розв'язування задачі стереозору.

1.1 Баєсівська задача

Нехай у нас є деякий об'єкт, який може бути описаний двома параметрами: x і k — *спостережуваним* і *прихованим* відповідно. При цьому x приймає значення зі скінченної множини X, а k — зі скінченного класу K. Символом D позначатимемо *множину можливих розв'язків*.

На множині всіх можливих пар $X \times K$ задається розподіл ймовірностей $p_{XK} : X \times K \to \mathbb{R}$ таким чином, щоб для кожної пари спостереження $x \in X$ та стану $k \in K$ значення $p_{XK}(x,k)$ позначало сумісну ймовірність знаходження об'єкта в стані k і випромінювання спостереження x.

Нехай $W : K \times D \to \mathbb{R}$ є *штрафною функцією* (також називається *функцією втрат*). Значення $W(k,d)$ в такому разі позначатиме втрати об'єкта в стані k при прийнятті рішення d з множини D. Тепер нехай $q : X \to D$ буде позначати функцію, яка називається *стратегією рішення*. Вона кожному спостереженню x ставить у відповідність деяке рішення $q(x) \in D$. Математичне сподівання втрат при використанні стратегії q вважається якістю відповідної стратегії і називається *ризиком*.

Баєсівська задача [4] полягає в тому, щоб для заданих множин X, K, D та відображень $p_{XK} : X \times K \to \mathbb{R}$, $W : K \times D \to \mathbb{R}$ побудувати таку стратегію $q : X \to D$, котра б мінімізувала баєсівський ризик $R(q)$, який визначається формулою

$$R(q) = \sum_{\langle x,k\rangle \in X \times K} p_{XK}(x,k) W(k,q(x)). \tag{1.1}$$

Розв'язком баєсівської задачі є стратегія q^*, котра мінімізує ризик $R(q)$. Вона називається *баєсівською стратегією*.

Зауважимо, що вводять іще поняття *часткового ризику* $R(x,d)$. Його можна визначити як умовне математичне сподівання штрафу, отриманого при умові спостереження x й рішення d, тобто:

$$R(x,d) = \sum_{k \in K} p_{K|X}(k|x) W(k,d). \tag{1.2}$$

В такому вигляді баєсівську задачу для кожного випадку спостереження x можна розв'язувати окремо. І полягає ця задача в пошуку такого рішення d, яке б мінімізовувало значення часткового ризику.

Наведемо приклад. У нас є деяка фотографія з накладеним білим шумом або водяним знаком. Перед нами поставлена задача відтворити зображення у початковому його вигляді до проведених над ним маніпуляцій (*image restoration problem* [5]). Тоді ми можемо представити спостережувану фотографію у вигляді спостереження x. А істинний її вигляд — як прихований стан k. Стратегією q буде метод відтворення початкового вигляду світлини. В такому разі баєсівською задачею буде пошук такої стратегії, яка б мінімізувала ризик поганого (тобто неякісного) відтворення істинного вигляду зображення. Для того, щоб розв'язати сформульовану в прикладі задачу, потрібно іще визначитися з відображеннями $p_{XK} : X \times K \to \mathbb{R}$ і $W : K \times D \to \mathbb{R}$. Почнемо спочатку зі штрафної функції. Будемо надалі вважати, що класи K і D рівні. Розглянемо перший важливий випадок штрафної функції $W : K^2 \to \mathbb{R}$:

$$W(k,k') = \mathbb{1}_{K \setminus \{k\}}(k'). \tag{1.3}$$

Якщо підставити наведену штрафну функцію в частковий ризик, то, виконуючи нескладні перетворення, отримаємо задачу пошуку моди, тобто такого значення k' при якому апостеріорна ймовірність приймає максимальне значення

$$d \in \underset{k' \in K}{\text{Argmax}}\, p_{K|X}(k'|x). \tag{1.4}$$

Розглянутий випадок штрафної функції є доволі наглядним і саме він лежить в основі багатьох алгоритмів, які розв'язують задачу стереозору. Метою таких методів є пошук такої мапи зсувів, яка б максимізувала відповідну апостеріорну ймовірність.

Тепер розглянемо інший випадок штрафної функції $W : K^2 \to \mathbb{R}$, а саме квадратичне відображення втрат, яке має вигляд

$$W(k,k') = \sum_{t \in T} (k_t - k'_t)^2. \tag{1.5}$$

Розв'язком баєсівської задачі з квадратичною штрафною функцією буде умовне математичне сподівання. Тоді поставлена задача зводиться до пошуку умовного математичного сподівання

$$d = \sum_{k \in K} p_{K|X}(k|x)k. \tag{1.6}$$

Щоб визначитися з відображенням $p_{XK} : K \to \mathbb{R}$, потрібно ввести поняття випадкового поля Ґіббза й відповідну термінологію. Саме ним описується розподіл ймовірностей, з яким необхідно розібратися для того, щоб представити повне математичне формулювання баєсівської задачі.

1.2 Випадкове поле Ґіббза

Нехай T позначає деяку множину координат розташування пікселів у зображенні розміру $h \times w$

$$T = \{\langle i,j\rangle : \ i = \overline{1,h}, \ j = \overline{1,w}\}, \tag{1.7}$$

і нехай $\tau \subset T^2$ позначає *структуру сусідства* елементів множини T. Нехай відношення τ є іррефлексивним та симетричним. Отже, впорядкована пара $\langle T, \tau\rangle$ є простим неорієнтованим графом без петель.

Тепер визначимо *множину міток* K. В загальному випадку для кожної вершини $t \in T$ нашої графової структури визначається окремий клас міток K_t. Проте ми будемо розглядати в подальшому виключно ситуацію, коли всі ті класи рівні. Також за замовчуванням ми будемо вважати множину міток K скінченною.

Нехай F є сімейством випадкових величин, визначеним на T, в якому кожна випадкова величина F_t приймає значення k_t з множини K. Тоді клас F називається *випадковим полем*. Функціональне відношення $k : T \to K$ називається *розміткою* і відповідає деякій реалізації випадкового поля F.

Випадкове поле з розподілом Ґіббза називається *випадковим полем Ґіббза* [6–8]. Розподіл Ґіббза має форму

$$P(k) = \frac{1}{Z} \exp\left(-\frac{1}{h}U(k)\right), \tag{1.8}$$

де $Z = \sum\limits_{k \in K^T} \exp\left(-\frac{1}{h}U(k)\right)$ — нормалізуюча стала, яка називається *статистичною сумою*, h — константа, яка називається *температурою* і яку ми вважатимемо надалі рівну одиниці, $U : K^T \to \mathbb{R}$ — *енергетична функція*. Відображення U визначається як $U(k) = \sum\limits_{c \in C} q_c(k)$, де C є множиною всіх *клік* графу $\langle T, \tau\rangle$, тобто всіх його повних підграфів, а $q_c : K^T \to \mathbb{R}$ називається *потенціалом кліки*. Значення потенціалу кліки

q_c залежить від локальної розмітки на кліці c.

Надалі ми будемо розглядати випадки, коли потенціали клік з трьома й більше вершинами приймають значення нуль. Тоді розподіл Ґіббза відповідного випадкового поля можна представити у вигляді

$$P(k) = \frac{1}{Z} \exp\left(- \sum_{t \in T} q_t(k_t) - \sum_{tt' \in \tau} g_{tt'}(k_t, k_{t'}) \right). \tag{1.9}$$

1.3 Задача стереозору

Припустимо, що у нас є *стереопара* — два зображення сцени, які були зроблені за допомогою спеціальної відкаліброваної системи камер (рисунок 1.1).

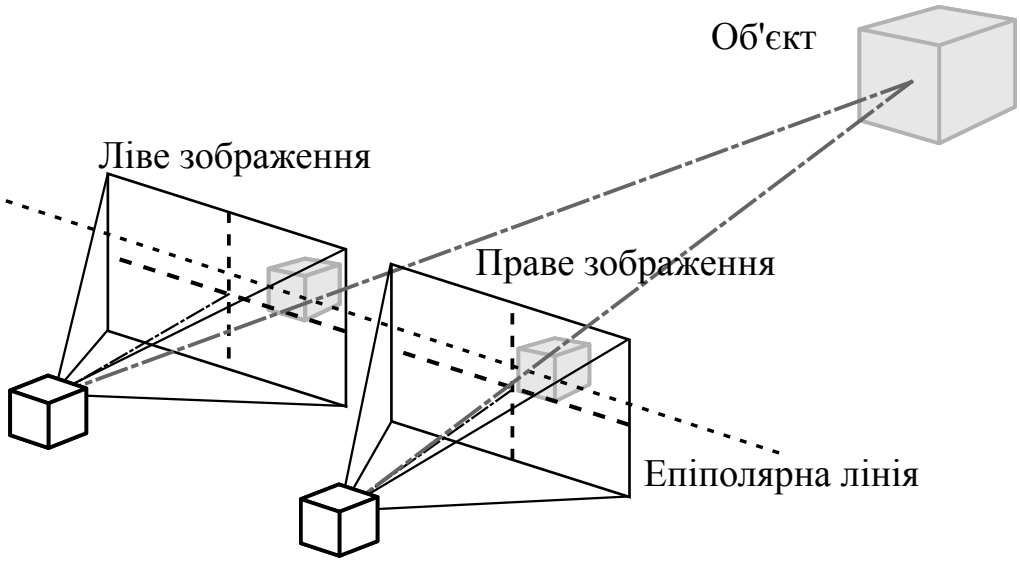


Рисунок 1.1 — Бінокулярний стереозір [9]

Математично множину зображень можна подати як $X_{pict} = \{0,\ldots,255\}^{h \times w \times 3}$, де h — висота зображення, w — його ширина. Тоді множину стереопар визначимо як $X_{sp} = X_{pict}^2$. Клас X_{sp} при спробі сформулювати задачу стереозору у вигляді баєсівської задачі буде нашою множиною спостережень.

Метою задачі стереозору є побудова *мапи зсувів* для заданої стереопари. Мапа

зсувів описує співвідношення між координатами відповідних пікселів пари зображень стереопари. Ми сфокусуємося на випадку горизонтального зсуву, тобто ситуації зміщення одного зображення горизонтально відносно іншого.

В баєсівській постановці задачі стереозору мапа зсувів буде прихованим станом нашого об'єкту, який ми намагаємося віднайти. Визначимо її у вигляді відображення $d_x : T \to K$, де $x \in X_{sp}$, $K \subset \mathbb{R}$, яке кожній координаті пікселя стереопари ставить у відповідність деяке дійсне значення — значення зсуву (в більш загальному вигляді K може бути, наприклад, підмножиною $\mathbb{R}^2$, оскільки зсув може бути в двох напрямках, але ми розглядаємо спрощену ситуацію).

Оберемо штрафною функцією відображення $W(k,k') = \mathbb{1}_{K^T \setminus \{k\}}(k')$. Тоді наша баєсівська задача полягає в пошуку такої мапи зсувів, яка б максимізувала апостеріорну ймовірність. Тобто $d_x \in \underset{k' \in K^T}{\text{Argmax}}\, p_{K^T|X_{sp}}(k'|x)$.

Для задання ймовірнісного розподілу над простором K^T будемо використовувати модель випадкового поля Ґіббза. Тоді відповідно умовні ймовірності, які потрібно визначити у випадку нашої баєсівської задачі, матимуть вигляд

$$p_{K^T|X_{sp}}(k|x) = \frac{1}{Z} \exp\left(- \sum_{t \in T} q_t^x(k_t) - \sum_{tt' \in \tau} g_{tt'}^x(k_t,k_{t'}) \right). \tag{1.10}$$

Оскільки натуральна показникова функція є монотонно зростаючою на всій множині дійсних чисел, то задача стереозору з вище вибраною штрафною функцією може бути спрощена до вигляду

$$d_x \in \underset{k' \in K^T}{\text{Argmin}} \left(\sum_{t \in T} q_t^x(k'_t) + \sum_{tt' \in \tau} g_{tt'}^x(k'_t,k'_{t'}) \right). \tag{1.11}$$

В задачі стереозору функція $q_t^x : K \to \mathbb{R}$ можна визначити формулою $q_t^x(k) = \|x_t^L - x_{t+k}^R\|$, де $\langle x^L, x^R \rangle \in X_{sp}$. Відображення $g_{tt'}^x : K^2 \to \mathbb{R}$ можна визначити,

наприклад, формулою $g^x_{tt'}(k,k') = \alpha\|k - k'\|$, де $\alpha > 0$ є деякою сталою. Також поширеним способом визначення останньої функції є

$$g^x_{tt'}(k,k') = \begin{cases} \alpha\|k - k'\|, & \|k - k'\| < \beta, \\ \alpha\beta, & \|k - k'\| \geqslant \beta, \end{cases} \tag{1.12}$$

де $\alpha,\ \beta > 0$ — деякі завчасно визначені константи [10, 11].

Щоб знайти розв'язок сформульованої задачі стереозору, важливо враховувати вигляд і властивості потенціалів клік, а також структуру графу $\langle T, \tau\rangle$. Звичайний алгоритм перебору розміток буде мати експоненційну складність $O\left(|T|^2|K|^{|T|}\right)$ (не враховуючи складність обчислення потенціалів клік, а виключно операцій сумування та пошуку мінімуму), хоча в багатьох випадках поставлену задачу можна розв'язати за поліноміальний час.

Структура графу $\langle T, \tau\rangle$ вкрай важлива. В задачі стереозору вона відповідає структурі сусідства пікселів зображень заданої стереопари. Ми розглянемо дві структури, які доволі часто використовуються на практиці (рисунок 1.2).

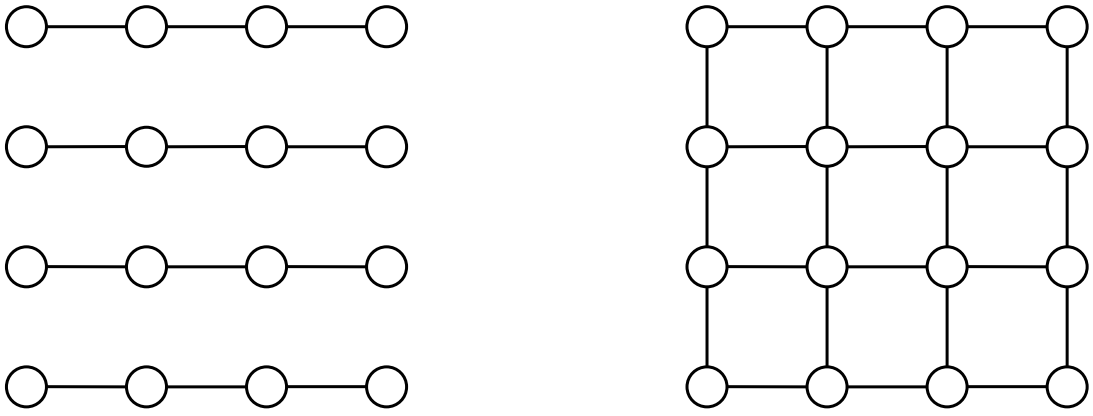


Рисунок 1.2 — Структури графів

Перший випадок характерний тим, що для нього можна знайти точний розв'язок за поліноміальний час. Для того, щоб розібратися з цим випадком більш детальніше,

потрібно ознайомитися з динамічним програмуванням.

1.4 Динамічне програмування

Сформульовану задачу стереозору для випадку незалежних горизонтальних ланцюжків можна розв'язати, використовуючи методи динамічного програмування. Продемонструємо принцип його роботи на практиці [4].

Математична структура $\langle R, \oplus, \otimes \rangle$, тобто множина R з двома визначеними на ній бінарними операціями $\oplus : R^2 \to R,\ \otimes : R^2 \to R$, називається *напівкільцем*, якщо виконуються наступні умови:

1) операції $\oplus,\ \otimes$ є замкненими на множині R;
2) $\forall r_1,\ r_2 \in R :\ r_1 \oplus r_2 = r_2 \oplus r_1$;
3) $\forall r_1,\ r_2,\ r_3 \in R :\ (r_1 \oplus r_2) \oplus r_3 = r_1 \oplus (r_2 \oplus r_3)$;
4) $\exists 0 \in R\, \forall r \in R :\ r \oplus 0 = 0 \oplus r = r$;
5) $\forall r_1,\ r_2,\ r_3 \in R :\ (r_1 \otimes r_2) \otimes r_3 = r_1 \otimes (r_2 \otimes r_3)$;
6) $\forall r_1,\ r_2,\ r_3 \in R :\ r_1 \otimes (r_2 \oplus r_3) = (r_1 \otimes r_2) \oplus (r_1 \otimes r_3)$;
7) $\forall r_1,\ r_2,\ r_3 \in R :\ (r_1 \oplus r_2) \otimes r_3 = (r_1 \otimes r_3) \oplus (r_2 \otimes r_3)$;
8) $\forall r \in R :\ r \otimes 0 = 0 \otimes r = 0$.

Властивість (2) називається *комутативністю* операції $\oplus$ на множині R. (3), (5) є властивостями *асоціативності* операцій $\oplus,\ \otimes$ на R відповідно. Властивість (6) називається властивістю *лівої дистрибутивності*, а (7) — *правою дистрибутивністю* напівкільця $\langle R, \oplus, \otimes \rangle$. (4) властивість визначає існування *нейтрального елементу* 0 за операцією $\oplus$ в напівкільці $\langle R, \oplus, \otimes \rangle$. Властивість (8) називається *мультиплікативною властивістю* нейтрального елементу 0.

Наведемо декілька прикладів напівкілець. Першим прикладом є напівкільце $\langle R, \min, + \rangle$, де множина $R = [a, +\infty) \cup \{+\infty\},\ a \in \mathbb{R}$. Легко переконатися в тому, що

для цього випадку виконуються всі вищеперелічені умови. Нейтральним елементом $\langle R, \min, + \rangle$ буде $+\infty$. Другим прикладом буде напівкільце $\langle R, +, \times \rangle$, де множина R є множиною дійсних чисел $\mathbb{R}$ або множиною $[0, +\infty)$. Нейтральним елементом для такої математичної структури буде число 0. В обох наведених прикладах мультиплікативні операції є комутативними. Такі напівкільця називаються *комутативними напівкільцями*.

Тепер нехай в нас є деяке довільне комутативне напівкільце $\langle R, \oplus, \otimes \rangle$, набір функцій $q_i : K \to R,\ i = \overline{1,n}$ та $g_{i,i+1} : K^2 \to R,\ i = \overline{1,n-1}$, де множина K — скінченна множина міток. Наша задача полягатиме в обчисленні значення

$$G = \bigoplus_{k:\{1,\ldots,n\}\to K} \left(\bigotimes_{i=\overline{1,n}} q_i(k_i) \otimes \bigotimes_{i=\overline{1,n-1}} g_{i,i+1}(k_i, k_{i+1}) \right). \tag{1.13}$$

Якщо матиме сенс, то також потрібно іще знайти таку розмітку $k^* : \{1,\ldots,n\} \to K$, для якої виконується умова

$$G = \bigotimes_{i=\overline{1,n}} q_i\left(k_i^*\right) \otimes \bigotimes_{i=\overline{1,n-1}} g_{i,i+1}(k_i^*, k_{i+1}^*). \tag{1.14}$$

Використовуючи комутативність операцій $\otimes$, $\oplus$ і представлення функції $k : \{1,\ldots,n\} \to K$ у вигляді $\langle k_1,\ldots,k_n \rangle$, ми можемо переписати задачу у наступному вигляді:

$$G = \bigoplus_{k_n \in K} \ldots \bigoplus_{k_1 \in K} \left(q_1(k_1) \otimes g_{1,2}(k_1, k_2) \otimes \ldots \otimes q_n(k_n) \right). \tag{1.15}$$

Скористаємося тепер правою дистрибутивністю. Тоді ми зможемо винести суму $\bigoplus_{k_1 \in K} \left(q_1(k_1) \otimes g_{1,2}(k_1, k_2) \right)$ для кожного $k_2 \in K$. Визначимо $f_2 : K \to R$ у вигляді

$$f_2(k_2) = \bigoplus_{k_1 \in K} \left(q_1(k_1) \otimes g_{1,2}(k_1, k_2) \right) \otimes q_2(k_2), \tag{1.16}$$

і отримаємо вираз

$$G = \bigoplus_{k_n \in K} \ldots \bigoplus_{k_2 \in K} \left(f_2(k_2) \otimes g_{2,3}(k_2,k_3) \otimes \ldots \otimes q_n(k_n) \right). \tag{1.17}$$

Продовжимо виконувати раніше описану процедуру. Спершу виносячи вираз

$$\bigoplus_{k_i \in K} \left(f_i(k_i) \otimes g_{i,i+1}(k_i,k_{i+1}) \right), \tag{1.18}$$

а потім визначаючи функцію $f_{i+1} : K \to R$ у вигляді

$$f_{i+1}(k_{i+1}) = \bigoplus_{k_i \in K} \left(f_i(k_i) \otimes g_{i,i+1}(k_i,k_{i+1}) \right) \otimes q_{i+1}(k_{i+1}). \tag{1.19}$$

Далі переходимо на нову ітерацію. Повторюємо відповідний алгоритм дій допоки не отримаємо вираз

$$G = \bigoplus_{k_n \in K} f_n(k_n). \tag{1.20}$$

Такий спосіб обчислення значення G має поліноміальну складність $O\left(|T||K|^2\right)$. В той же час алгоритм звичайного перебору для обчислення G має експоненційну складність $O\left(|T|^2|K|^{|T|}\right)$.

Щоб знайти розмітку $k^* : \{1,\ldots,n\} \to K$, скористаємося тим, що її можна представити у векторному вигляді $\langle k_1^*,\ldots,k_n^* \rangle$. Тоді k_n^* можна знайти як $G = f_n(k_n^*)$. Далі пригадуємо, що

$$f_n(k_n) = \bigoplus_{k_{n-1} \in K} \left(f_{n-1}(k_{n-1}) \otimes g_{n-1,n}(k_{n-1},k_n) \right) \otimes q_n(k_n). \tag{1.21}$$

Відповідно k_{n-1}^* можна знайти як таке значення, для якого виконується співвідно-

шення

$$G = f_{n-1}(k^*_{n-1}) \otimes c_{n-1}(k^*_{n-1}), \tag{1.22}$$

де $c_{n-1}(k_{n-1}) = g_{n-1,n}(k_{n-1},k^*_n) \otimes q_n(k^*_n)$. Оскільки значення k^*_n уже є відомим, нам потрібно перебрати усього $|K|$ різних значень k_{n-1}, а не $|K|^2$ пар $\langle k_{n-1},k_n\rangle$. Далі розписуємо функцію $f_{n-1} : K \to R$ і визначаємо $c_{n-2} : K \to R$ як

$$c_{n-2}(k_{n-2}) = g_{n-2,n-1}(k_{n-2},k^*_{n-1}) \otimes q_{n-1}(k^*_{n-1}) \otimes c_{n-1}(k^*_{n-1}). \tag{1.23}$$

Продовжуючи виконувати ітеративним чином описану процедуру, ми врешті-решт знайдемо розмітку $k^* : \{1,\ldots,n\} \to K$, для якої матимемо

$$G = \bigotimes_{i=\overline{1,n}} q_i\,(k^*_i) \otimes \bigotimes_{i=\overline{1,n-1}} g_{i,i+1}(k^*_i,k^*_{i+1}). \tag{1.24}$$

Легко переконатися, що представлений спосіб пошуку розмітки $k^* : \{1,\ldots,n\} \to K$ буде мати поліноміальну складність $O(|T||K|)$.

З повним алгоритмом для пошуку значення G та знаходження розмітки $k^* : \{1,\ldots,n\} \to K$ можна ознайомитися нижче (алгоритм 1).

Використовуючи властивості комутативного напівкільця, ми змогли побудувати алгоритм, який знаходить точний розв'язок поставленої задачі за поліноміальний час. Продемонстрований алгоритм є реалізацією ідеї метода *динамічного програмування* [12].

У випадку напівкільця $\langle R, \min, + \rangle$ вищенаведений алгоритм буде розв'язувати задачу пошуку найкоротшого шляху у графі. Значення функцій $q_i : K \to R,\ i = \overline{1,n}$ означатимуть ваги k_i-ої вершини в i-му об'єкті, а значення $g_{i,i+1}(k_i,k_{i+1}),\ i = \overline{1,n-1}$ — ваги ребер, які сполучають k_i-ту й k_{i+1}-шу вершини між i-им і $i+1$-им об'єктами (рисунок 1.3). Задача полягатиме у виборі з кожного об'єкту по одній

вершині, через які в сукупності проходив би найкоротший маршрут.

Algorithm 1 Алгоритм знаходження G і розмітки k^*

Require: функції $q_i : K \to R,\ i = \overline{1,n},\ g_{i,i+1} : K^2 \to R,\ i = \overline{1,n-1}$ визначені, $n \geqslant 2$

Ensure: $G = \bigoplus_{k:\{1,\ldots,n\}\to K} (q_1(k_1) \otimes \ldots \otimes q_n(k_n)),\ k^* \in K^{\{1,\ldots,n\}} :\ G = q_1(k_1^*) \otimes \ldots \otimes q_n(k_n^*)$

$f_2(k_2) \leftarrow \bigoplus_{k_1 \in K} (q_1(k_1) \otimes g_{1,2}(k_1,k_2)) \otimes q_2(k_2),\ k_2 \in K$
$i \leftarrow 2$
while $i \neq n$ **do**
    $f_{i+1}(k_{i+1}) \leftarrow \bigoplus_{k_i \in K} (f_i(k_i) \otimes g_{i,i+1}(k_i,k_{i+1})) \otimes q_{i+1}(k_{i+1}),\ k_{i+1} \in K$
    $i \leftarrow i + 1$
end while
$G \leftarrow \bigoplus_{k_n \in K} f_n(k_n)$
if потрібно знайти $k^* : \{1,\ldots,n\} \to K$ **then**
    $S \leftarrow K$
    вибрати $k_n^* \in S$
    while $G \neq f_n(k_n^*)$ **do**
        $S \leftarrow S \setminus \{k_n^*\}$
        вибрати $k_n^* \in S$
    end while
    $c_{n-1}(k_{n-1}) \leftarrow g_{n-1,n}(k_{n-1},k_n^*) \otimes q_n(k_n^*),\ k_{n-1} \in K$
    $i \leftarrow n - 1$
    while $i \neq 1$ **do**
        $S \leftarrow K$
        вибрати $k_i^* \in S$
        while $G \neq f_i(k_i^*) \otimes c_i(k_i^*)$ **do**
            $S \leftarrow S \setminus \{k_i^*\}$
            вибрати $k_i^* \in S$
        end while
        $c_{i-1}(k_{i-1}) \leftarrow g_{i-1,i}(k_{i-1},k_i^*) \otimes q_i(k_i^*) \otimes c_i(k_i^*),\ k_{i-1} \in K$
        $i \leftarrow i - 1$
    end while
    $S \leftarrow K$
    вибрати $k_1^* \in S$
    while $G \neq q_1(k_1^*) \otimes c_1(k_1^*)$ **do**
        $S \leftarrow S \setminus \{k_1^*\}$
        вибрати $k_1^* \in S$
    end while
end if

Підхід динамічного програмування знайшов доволі широке застосування на практиці. Наприклад, метод динамічного програмування застосовується для знаходження розв'язку задачі пошуку відстані найменшого редагування — відстані Левенштейна. В теорії формальних граматик цей спосіб застосовується в алгоритмі Кока-Янґера-Касамі. Іще одним прикладом є алгоритм Вітербі, який є частковим випадком алгоритму 1 для напівкільця $\langle [0,1], \max, \times \rangle$. Він використовується для пошуку найбільш ймовірної послідовності прихованих станів і знайшов широке застосування, зокрема, у задачах біоінформатики [13] та обробки природної мови [14].

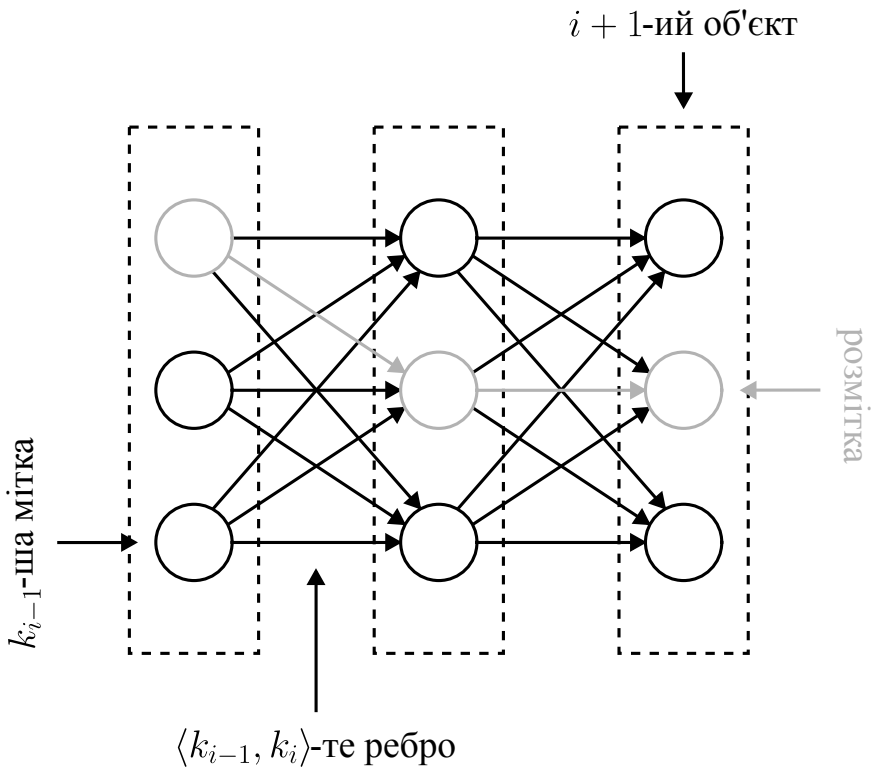


Рисунок 1.3 — Розмітка

1.5 Semi-Global Matching

В попередньому підрозділі ми ознайомилися з методом динамічного програмування. З його допомогою можна знайти точний розв'язок сформульованої раніше задачі стереозору для випадку незалежних горизонтальних ланцюжків за поліноміальний час. Проте з другою запропонованою графовою структурою все набагато складніше. Для цього випадку не існує загального алгоритму, який би за поліноміальний час знаходив точний розв'язок поставленої задачі на глобальну оптимізацію [15]. Натомість з'явилося багато методів, які розв'язують задачу наближено.

Одним із запропонованих алгоритмів для розв'язання задачі стереозору у випадку більш складної структури сусідства є алгоритм *Semi-Global Matching* (SGM) [16]. Головна ідея алгоритму полягає у використанні динамічного програмування для знаходження локальних розв'язків задачі на глобальну оптимізацію. Таким чи-

ном алгоритм дозволяє за поліноміальний час наближено знайти розв’язок, а результат виглядає не гірше, ніж у випадку незалежних горизонтальних ланцюжків.

Метод динамічного програмування можна застосовувати не тільки для ланцюжкової графової структури сусідства, але також і для більш загальної задачі оптимізації на *деревах*, тобто ациклічних зв’язних графах. Алгоритм SGM для кожного значення мапи зсувів знаходить розв’язок у вигляді локального розв’язку кореня відповідного хрестоподібного дерева (рисунок 1.4).

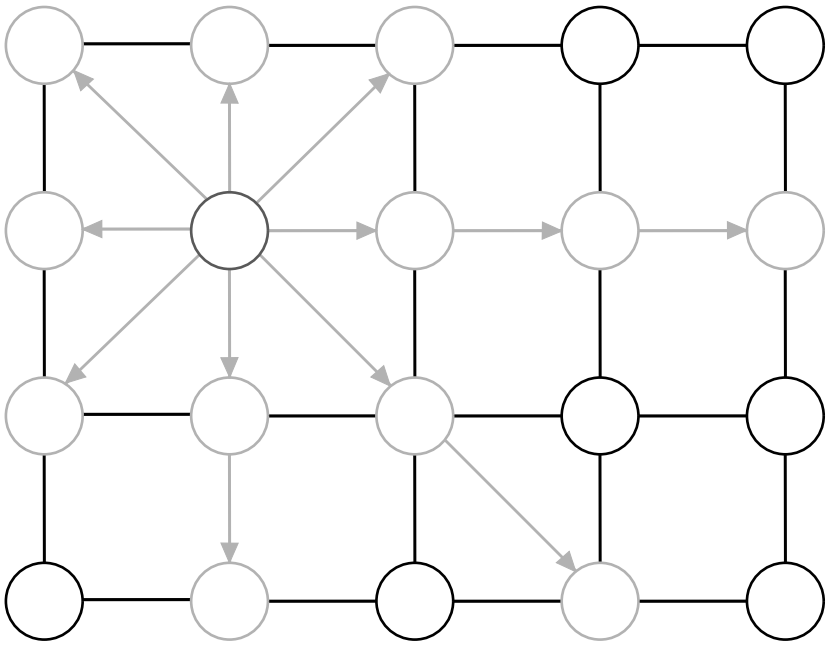


Рисунок 1.4 — Хрестоподібне дерево

Оригінальний алгоритм SGM розглядає ситуацію з більш складною структурою сусідства, коли сусідами вважаються іще сусідні по діагоналі об’єкти. Ми будемо розглядати спрощений випадок з чотирма бічними сусідами. З версією алгоритму SGM для такої ситуації можна ознайомитися нижче (алгоритм 2).

Є багато робіт про алгоритми, які були створенні під натхненням SGM. До них відносяться, наприклад, алгоритми SGM-Net [17], SGM-Forest [18]. В другому розділі ми запропонуємо нову його модифікацію.

Algorithm 2 Алгоритм SGM

$D_{i,j}(k) \leftarrow 0,\ i = \overline{1,h+1},\ j = \overline{1,w},\ k \in K$
$R_{i,j}(k) \leftarrow 0,\ i = \overline{1,h},\ j = \overline{1,w+1},\ k \in K$
$U_{i,j}(k) \leftarrow 0,\ i = \overline{0,h},\ j = \overline{1,w},\ k \in K$
$L_{i,j}(k) \leftarrow 0,\ i = \overline{1,h},\ j = \overline{0,w},\ k \in K$
for $i = \overline{h,1}$ **do**
  for $j = \overline{w,1}$ **do**
    for $k \in K$ **do**
      $D_{i,j}(k) \leftarrow \min\limits_{k' \in K} \left(D_{i+1,j}(k') + q^x_{i+1,j}(k') + g^x_{i,j,i+1,j}(k,k')\right)$
      $R_{i,j}(k) \leftarrow \min\limits_{k' \in K} \left(R_{i,j+1}(k') + q^x_{i,j+1}(k') + g^x_{i,j,i,j+1}(k,k')\right)$
    end for
  end for
end for
for $i = \overline{1,h}$ **do**
  for $j = \overline{1,w}$ **do**
    for $k \in K$ **do**
      $U_{i,j}(k) \leftarrow \min\limits_{k' \in K} \left(U_{i-1,j}(k') + q^x_{i-1,j}(k') + g^x_{i,j,i-1,j}(k,k')\right)$
      $L_{i,j}(k) \leftarrow \min\limits_{k' \in K} \left(L_{i,j-1}(k') + q^x_{i,j-1}(k') + g^x_{i,j,i,j-1}(k,k')\right)$
    end for
  end for
end for
for $i = \overline{1,h}$ **do**
  for $j = \overline{1,w}$ **do**
    $d_x(i,j) \leftarrow \operatorname{argmin}\limits_{k \in K} \left(D_{i,j}(k) + R_{i,j}(k) + q^x_{i,j}(k) + U_{i,j}(k) + L_{i,j}(k)\right)$
  end for
end for
return $d_x : H \times W \to K$

Висновки до розділу 1

В першому розділі ми проаналізували задачу і різні методи її розв'язування. Ми зробили висновок, що досі дослідники в області комп'ютерного зору займаються пошуком нових алгоритмів, які б ефективно розв'язували задачу стереозору.

Існує багато алгоритмів, які розв'язують задачу стереозору і широко застосовуються на практиці. Проте в них часто робиться ряд припущень, які можуть суттєво зменшити область їхнього застосування.

Ми вирішили зосередитися у цій роботі на побудові модифікації популярного алгоритму SGM, використовуючи альтернативне визначення мапи зсувів у вигляді математичного сподівання усіх можливих розміток. Головними факторами при виборі SGM стали його простота і компроміс між швидкістю його роботи і якістю отриманих результатів.

2 ПОБУДОВА І ОЦІНКА АЛГОРИТМУ EXPECTED SGM

На початку розділу наведений детальний опис процесу побудови алгоритму пошуку математичного сподівання для випадку ланцюжкової структури сусідства. Після цього ми ознайомимося з детальною побудовою алгоритму Expected Semi-Global Matching (Expected SGM) — модифікацією алгоритму SGM, а також іще однієї альтернативної модифікації. В кінці розділу буде міститися інформація про порівняння побудованих алгоритмів з іншими методами знаходження розв'язку задачі стереозору.

2.1 Задача пошуку математичного сподівання

В попередньому розділі була розглянута задача пошуку розмітки на комутативному напівкільці для випадку ланцюжкової графової структури сусідства за допомогою використання принципів динамічного програмування. Тепер ми аналогічним чином, використовуючи методи динамічного програмування, побудуємо алгоритм для знаходження математичного сподівання на ланцюжках.

Задача пошуку математичного сподівання виникає, якщо в баєсівському формулюванні задачі стереозору в ролі штрафної функції взяти квадратичну функцію втрат $W(k,k') = \sum\limits_{t\in T}(k_t - k'_t)^2$. Тобто задача полягатиме у пошуку мапи зсувів у формі

$$d_x = \sum_{k\in K^T} \frac{1}{Z} \exp\left(-\sum_{t\in T} q_t^x(k_t) - \sum_{tt'\in\tau} g_{tt'}^x(k_t,k_{t'})\right) \cdot k. \tag{2.1}$$

Враховуючи ланцюжкову структуру сусідства об'єктів, мапа зсувів може бути

представлена у вигляді

$$d_x = \sum_{k_1 \in K} \dots \sum_{k_n \in K} \frac{1}{Z} \exp\left(- \sum_{i=\overline{1,n}} q_i^x(k_i) - \sum_{i=\overline{1,n-1}} g_{i,i+1}^x(k_i,k_{i+1}) \right) \cdot \langle k_1,\dots,k_n \rangle. \quad (2.2)$$

Оскільки Z не залежить від $k_1,\dots,k_n$, то його можна винести і окремо розглядати задачу з його обчисленням паралельно.

$$Z = \sum_{k_1 \in K} \dots \sum_{k_n \in K} \exp\left(- \sum_{i=\overline{1,n}} q_i^x(k_i) - \sum_{i=\overline{1,n-1}} g_{i,i+1}^x(k_i,k_{i+1}) \right). \quad (2.3)$$

Задача з пошуку Z є ідентичною до задачі обчислення значення G з попереднього розділу для випадку комутативного напівкільця $\langle [0, +\infty), +, \times \rangle$. Натомість для обчислення $Z \cdot d_x$ маємо трохи іншу ситуацію.

Використовуючи властивості асоціативності й комутативності операції $+$, перегрупуємо елементи в експоненційній функції. Далі скористаємося властивістю лівої дистрибутивності. Тоді таким чином у нас з'явиться можливість винести лінійну комбінацію $\sum_{k_n \in K} \exp\left(-g_{n-1,n}^x(k_{n-1},k_n) - q_n^x(k_n)\right) \cdot \langle k_1,\dots,k_n \rangle$ для кожного $k_{n-1} \in K$. Внесемо коефіцієнти $\exp\left(-g_{n-1,n}^x(k_{n-1},k_n) - q_n^x(k_n)\right)$ у вектор $\langle k_1,\dots,k_n \rangle$ разом з сумою. Тоді $Z \cdot d_x$ вдасться спростити до вигляду

$$Z \cdot d_x = \sum_{k_1 \in K} \dots \sum_{k_{n-1} \in K} \alpha_{n-1}(k_1,\dots,k_{n-1}) \cdot \langle Z_{n-1}(k_{n-1})k_1,\dots,f_{n-1}(k_{n-1}) \rangle, \quad (2.4)$$

де маємо $\alpha_{n-1}(k_1,\dots,k_{n-1}) = \exp\left(- \sum_{i=\overline{1,n-1}} q_i^x(k_i) - \sum_{i=\overline{1,n-2}} g_{i,i+1}^x(k_i,k_{i+1}) \right)$, визначено відображення $Z_{n-1}(k_{n-1}) = \sum_{k_n \in K} \exp\left(-g_{n-1,n}^x(k_{n-1},k_n) - q_n^x(k_n)\right)$, а також функцію $f_{n-1}(k_{n-1}) = \sum_{k_n \in K} \exp\left(-g_{n-1,n}^x(k_{n-1},k_n) - q_n^x(k_n)\right) \cdot k_n$. Далі ітеративним чином

повторюємо подібні кроки. Спочатку визначаємо відображення $Z_{n-i}(k_{n-i}) : K \to \mathbb{R}$ у вигляді

$$Z_{n-i}(k_{n-i}) = \sum_{k_{n-i+1} \in K} \alpha_{n-i,n-i+1}(k_{n-i},k_{n-i+1}) \cdot Z_{n-i+1}(k_{n-i+1}), \tag{2.5}$$

де $\alpha_{n-i,n-i+1}(k_{n-i},k_{n-i+1}) = \exp\left(-g^x_{n-i,n-i+1}(k_{n-i},k_{n-i+1}) - q^x_{n-i+1}(k_{n-i+1})\right)$. Тепер потрібно визначити відображення $f_j : K \to \mathbb{R}^{n-j}$ (де $j = n - i$) як

$$f_j(k_j) = \sum_{k_{j+1} \in K} \alpha_{j,j+1}(k_j,k_{j+1}) \cdot \langle Z_{j+1}(k_{j+1})k_{j+1}, f_{j+1}(k_{j+1})\rangle, \tag{2.6}$$

і тоді $Z \cdot d_x$ після введених функцій можна представити у формі

$$Z \cdot d_x = \sum_{k_1 \in K} \dots \sum_{k_{n-i} \in K} \alpha_{n-i}(k_1,\dots,k_{n-i}) \cdot \langle Z_{n-i}(k_{n-i})k_1,\dots,f_{n-i}(k_{n-i})\rangle. \tag{2.7}$$

Продовжуючи виконувати вищенаведені кроки, нам вдасться скоротити $Z \cdot d_x$ до визначеного вигляду

$$Z \cdot d_x = \sum_{k_1 \in K} \exp\left(-q_1(k_1)\right) \cdot \langle Z_1(k_1)k_1, f_1(k_1)\rangle. \tag{2.8}$$

З використанням методів динамічного програмування нам вдалося побудувати алгоритм для знаходження математичного сподівання на ланцюжках, який має поліноміальну складність $O(|T|^2|K|^2)$. Весь алгоритм наведений на наступній сторінці (алгоритм 3).

Незважаючи на всю лаконічність і простоту, наведений алгоритм має одну суттєву проблему, яка може досить серйозно нашкодити при його застосуванні на практиці. Вона полягає у його нестійкості.

Algorithm 3 Алгоритм пошуку математичного сподівання на ланцюжках

for $k \in K$ **do**
    $Z_{n-1}(k) \leftarrow \sum\limits_{k' \in K} \exp\left(-g^x_{n-1,n}(k,k') - q^x_n(k')\right)$
    $f_{n-1}(k) \leftarrow \sum\limits_{k' \in K} \exp\left(-g^x_{n-1,n}(k,k') - q^x_n(k')\right) \cdot k'$
end for
for $i = \overline{n-2,1}$ **do**
    for $k \in K$ **do**
        $Z_i(k) \leftarrow \sum\limits_{k' \in K} \exp\left(-g^x_{i,i+1}(k,k') - q^x_{i+1}(k')\right) \cdot Z_{i+1}(k')$
        $f_i(k) \leftarrow \sum\limits_{k' \in K} \exp\left(-g^x_{i,i+1}(k,k') - q^x_{i+1}(k')\right) \cdot \langle Z_{i+1}(k')k', f_{i+1}(k')\rangle$
    end for
end for
$Z \leftarrow \sum\limits_{k \in K} \exp\left(-q_1(k)\right) \cdot Z_1(k)$
$d_x \leftarrow \dfrac{1}{Z} \sum\limits_{k \in K} \exp\left(-q_1(k)\right) \cdot \langle Z_1(k)k, f_1(k)\rangle$
return $d_x : \{1,\ldots,n\} \to \mathbb{R}$

2.2 Покращення стійкості алгоритму

Нестійкість наведеного у попередньому підрозділі алгоритму є досить серйозною проблемою. Вона полягає в тому, що $Z_i : K \to \mathbb{R}$ може з експоненційною швидкістю прямувати до нуля або до нескінченності. Така ж проблема характерна і для набору функцій $f_i : K \to \mathbb{R}^i$.

Оскільки послідовність $Z_i : K \to \mathbb{R}$ функцій є скінченною і експоненти при їхньому обчисленні приймають виключно додатні скінченні значення, то в теорії $Z_i : K \to \mathbb{R}$ не досягають ніколи нуля або нескінченності. Відповідно це стосується й нормалізуючої сталої Z.

На практиці проблема залишається актуальною, оскільки через експоненційну швидкість прямування до нуля (або нескінченності) виникає *проблема машинного нуля* — ситуація, у котрій комп'ютерна система при дуже малих значеннях округлює їх до нуля (або при великих значеннях до верхньої межі).

Частково вирішити проблему машинного нуля нам допоможе введення множників $\beta_i,\ \overline{1,n}$. Наведемо більш детальний опис процесу їхнього використання на прикладі послідовності відображень $Z_i : K \to \mathbb{R}$.

Нехай у нас є деяке мале значення $\varepsilon > 0$. Нагадаємо, що функція $Z_{n-1} : K \to \mathbb{R}$ має вигляд

$$Z_{n-1}(k_{n-1}) = \sum_{k_n \in K} \exp\left(-g^x_{n-1,n}(k_{n-1},k_n) - q^x_n(k_n)\right). \tag{2.9}$$

Тоді для того, щоб функція приймала значення не менше за ε потрібно її помножити на деякий коефіцієнт β_{n-1}, який можна знайти за формулою

$$\beta_{n-1} = \frac{\varepsilon}{\min\limits_{k_{n-1} \in K} Z_{n-1}(k_{n-1})}. \tag{2.10}$$

Відповідний коефіцієнт потрібно буде помножити й на кожне значення функції $f_{n-1} : K \to \mathbb{R}$. Це забезпечить у майбутньому можливість отримання точного результату обчислення $d_x : T \to K$.

Аналогічним чином ми продовжимо діяти і на наступних ітераціях. Спочатку потрібно обчислити значення функції $Z_{n-i} : K \to \mathbb{R}$. Далі обчислюється коефіцієнт β_{n-i}:

$$\beta_{n-i} = \frac{\varepsilon}{\min\limits_{k_{n-i} \in K} Z_{n-i}(k_{n-i})}. \tag{2.11}$$

Після цього отримане значення множиться на кожне значення функції $Z_{n-i} : K \to \mathbb{R}$. Водночас його потрібно помножити і на кожне значення завчасно обчисленої функції $f_{n-i} : K \to \mathbb{R}^i$.

В кінці алгоритму при обчисленні константи Z і мапи зсувів $d_x : T \to \mathbb{R}$ рахувати коефіцієнт для забезпечення чисельної стійкості немає необхідності. Вся модифікація алгоритму наведена нижче (алгоритм 4).

Представлена модифікація не вирішує проблему з верхньою межею. Частково для вирішення цієї проблеми при реалізації програмної частини треба використовувати такий тип даних, в якому можна представити якомога більші числові значення. Наприклад, при реалізації на мові програмування Python з бібліотекою NumPy мо-

жна використовувати тип даних numpy.float128. Такий тип дозволяє працювати зі значеннями, які знаходяться в межах від -2^{16384} до 2^{16384}.

Algorithm 4 Модифікація алгоритму пошуку математичного сподівання на ланцюжках

for $k \in K$ **do**
  $Z_{n-1}(k) \leftarrow \sum_{k' \in K} \exp\left(-g^x_{n-1,n}(k,k') - q^x_n(k')\right)$
end for
$\beta_{n-1} \leftarrow \dfrac{\varepsilon}{\min\limits_{k' \in K} Z_{n-1}(k')}$
for $k \in K$ **do**
  $Z_{n-1}(k) \leftarrow \beta_{n-1} Z_{n-1}(k)$
  $f_{n-1}(k) \leftarrow \beta_{n-1} \sum_{k' \in K} \exp\left(-g^x_{n-1,n}(k,k') - q^x_n(k')\right) \cdot k'$
end for
for $i = \overline{n-2,1}$ **do**
  for $k \in K$ **do**
    $Z_i(k) \leftarrow \sum_{k' \in K} \exp\left(-g^x_{i,i+1}(k,k') - q^x_{i+1}(k')\right) \cdot Z_{i+1}(k')$
  end for
  $\beta_i \leftarrow \dfrac{\varepsilon}{\min\limits_{k' \in K} Z_i(k')}$
  for $k \in K$ **do**
    $Z_i(k) \leftarrow \beta_i Z_i(k)$
    $f_i(k) \leftarrow \beta_i \sum_{k' \in K} \exp\left(-g^x_{i,i+1}(k,k') - q^x_{i+1}(k')\right) \cdot \langle Z_{i+1}(k')k', f_{i+1}(k')\rangle$
  end for
end for
$Z \leftarrow \sum_{k \in K} \exp\left(-q_1(k)\right) \cdot Z_1(k)$
$d_x \leftarrow \dfrac{1}{Z} \sum_{k \in K} \exp\left(-q_1(k)\right) \cdot \langle Z_1(k)k, f_1(k)\rangle$
return $d_x : \{1,\ldots,n\} \to \mathbb{R}$

2.3 Expected Semi-Global Matching

Задачу пошуку математичного сподівання на деревах можна також вирішити методами динамічного програмування за поліноміальний час. Але з більш складною графовою структурою (яка містить цикли) виникають такі самі проблеми, як і в попередньому розділі. Тому замість того, щоб знаходити точний розв'язок задачі пошуку математичного сподівання, ми побудуємо алгоритм, що буде розв'язувати поставлену задачу локально — тобто буде шукати математичне сподівання для кожно-

го об'єкта (пікселя в нашому випадку) як для кореня відповідного хрестоподібного дерева на графі. Описана ситуація є простішою за загальний випадок з пошуком математичного сподівання на всьому хрестоподібному дереві, оскільки потребує лише обчислення функцій $Z^c_{i,j} : K \to \mathbb{R},\ c \in \{D,R,U,L\}$:

$$Z^D_{i,j}(k) = \sum_{k' \in K} \exp\left(-q^x_{i+1,j}(k') - g^x_{i,j,i+1,j}(k,k')\right) \cdot Z^D_{i+1,j}(k'), \tag{2.12}$$

$$Z^R_{i,j}(k) = \sum_{k' \in K} \exp\left(-q^x_{i,j+1}(k') - g^x_{i,j,i,j+1}(k,k')\right) \cdot Z^R_{i,j+1}(k'), \tag{2.13}$$

$$Z^U_{i,j}(k) = \sum_{k' \in K} \exp\left(-q^x_{i-1,j}(k') - g^x_{i,j,i-1,j}(k,k')\right) \cdot Z^U_{i-1,j}(k'), \tag{2.14}$$

$$Z^L_{i,j}(k) = \sum_{k' \in K} \exp\left(-q^x_{i,j-1}(k') - g^x_{i,j,i,j-1}(k,k')\right) \cdot Z^L_{i,j-1}(k'). \tag{2.15}$$

На початку алгоритму робиться ініціалізація для всіх функій $Z^c_{i,j} : K \to \mathbb{R}$ і значень $c \in \{D,R,U,L\}$, де вони всі прирівнюються до нейтрального елементу операції множення — одиниці. В кінці алгоритму для знаходження мапи зсувів $d_x : T \to \mathbb{R}$ потрібно в циклі знайти всі відповідні значення $d_x(i,j)$, обчислюючи їх за формулою

$$d_x(i,j) = \frac{1}{Z_{i,j}} \sum_{k \in K} \exp\left(-q^x_{i,j}(k)\right) \cdot Z^D_{i,j}(k) Z^R_{i,j}(k) Z^U_{i,j}(k) Z^L_{i,j}(k) k, \tag{2.16}$$

де значення $Z_{i,j} = \sum\limits_{k \in K} \exp\left(-q^x_{i,j}(k)\right) \cdot Z^D_{i,j}(k) Z^R_{i,j}(k) Z^U_{i,j}(k) Z^L_{i,j}(k)$ для всіх значень $i = \overline{1,h},\ j = \overline{1,w}$. В такому випадку алгоритм має поліноміальну складність $O(|T||K|^2)$, де $|T| = h \cdot w$. З повним псевдокодом описаного алгоритму Expected SGM можна ознайомитися на наступній сторінці (алгоритм 5).

Другим способом узагальнити алгоритм SGM на випадок пошуку математичного сподівання для складної структури сусідства (чотирма сусідами) є алгоритм More Expected SGM. Головною ідеєю є пошук математичного сподівання у всіх хре-

Algorithm 5 Алгоритм Expected SGM

$Z^D_{i,j}(k) \leftarrow 1,\ i = \overline{1,h+1},\ j = \overline{1,w},\ k \in K$
$Z^R_{i,j}(k) \leftarrow 1,\ i = \overline{1,h},\ j = \overline{1,w+1},\ k \in K$
$Z^U_{i,j}(k) \leftarrow 1,\ i = \overline{0,h},\ j = \overline{1,w},\ k \in K$
$Z^L_{i,j}(k) \leftarrow 1,\ i = \overline{1,h},\ j = \overline{0,w},\ k \in K$
for $i = \overline{h,1}$ **do**
  for $j = \overline{w,1}$ **do**
    for $k \in K$ **do**
      $Z^D_{i,j}(k) \leftarrow \sum_{k' \in K} \exp\left(-q^x_{i+1,j}(k') - g^x_{i,j,i+1,j}(k,k')\right) \cdot Z^D_{i+1,j}(k')$
      $Z^R_{i,j}(k) \leftarrow \sum_{k' \in K} \exp\left(-q^x_{i,j+1}(k') - g^x_{i,j,i,j+1}(k,k')\right) \cdot Z^R_{i,j+1}(k')$
    end for
    $\beta^D_{i,j} \leftarrow \dfrac{\varepsilon}{\min\limits_{k' \in K} Z^D_{i,j}(k')}$
    $\beta^R_{i,j} \leftarrow \dfrac{\varepsilon}{\min\limits_{k' \in K} Z^R_{i,j}(k')}$
    for $k \in K$ **do**
      $Z^D_{i,j}(k) \leftarrow \beta^D_{i,j} Z^D_{i,j}(k)$
      $Z^R_{i,j}(k) \leftarrow \beta^R_{i,j} Z^R_{i,j}(k)$
    end for
  end for
end for
for $i = \overline{1,h}$ **do**
  for $j = \overline{1,w}$ **do**
    for $k \in K$ **do**
      $Z^U_{i,j}(k) \leftarrow \sum_{k' \in K} \exp\left(-q^x_{i-1,j}(k') - g^x_{i,j,i-1,j}(k,k')\right) \cdot Z^U_{i-1,j}(k')$
      $Z^L_{i,j}(k) \leftarrow \sum_{k' \in K} \exp\left(-q^x_{i,j-1}(k') - g^x_{i,j,i,j-1}(k,k')\right) \cdot Z^L_{i,j-1}(k')$
    end for
    $\beta^U_{i,j} \leftarrow \dfrac{\varepsilon}{\min\limits_{k' \in K} Z^U_{i,j}(k')}$
    $\beta^L_{i,j} \leftarrow \dfrac{\varepsilon}{\min\limits_{k' \in K} Z^L_{i,j}(k')}$
    for $k \in K$ **do**
      $Z^U_{i,j}(k) \leftarrow \beta^U_{i,j} Z^U_{i,j}(k)$
      $Z^L_{i,j}(k) \leftarrow \beta^L_{i,j} Z^L_{i,j}(k)$
    end for
  end for
end for
for $i = \overline{1,h}$ **do**
  for $j = \overline{1,w}$ **do**

$$d_x(i,j) \leftarrow \frac{\sum\limits_{k \in K} \exp\left(-q^x_{i,j}(k)\right) \cdot Z^D_{i,j}(k) Z^R_{i,j}(k) Z^U_{i,j}(k) Z^L_{i,j}(k) k}{\sum\limits_{k \in K} \exp\left(-q^x_{i,j}(k)\right) \cdot Z^D_{i,j}(k) Z^R_{i,j}(k) Z^U_{i,j}(k) Z^L_{i,j}(k)}$$

  end for
end for
return $d_x : H \times W \to K$

Algorithm 6 Алгоритм More Expected SGM

$Z^D_{i,j}(k) \leftarrow 1,\ f^D_{i,j}(k) \leftarrow \varnothing,\ i = \overline{1,h+1},\ j = \overline{1,w},\ k \in K$ ▷ тут $\varnothing$ — нейтральний елемент конкатенації

$Z^R_{i,j}(k) \leftarrow 1,\ f^R_{i,j}(k) \leftarrow \varnothing,\ i = \overline{1,h},\ j = \overline{1,w+1},\ k \in K$

$Z^U_{i,j}(k) \leftarrow 1,\ f^U_{i,j}(k) \leftarrow \varnothing,\ i = \overline{0,h},\ j = \overline{1,w},\ k \in K$

$Z^L_{i,j}(k) \leftarrow 1,\ f^L_{i,j}(k) \leftarrow \varnothing,\ i = \overline{1,h},\ j = \overline{0,w},\ k \in K$

for $i = \overline{h,1}$ **do**

  for $j = \overline{w,1}$ **do**

$$Z^D_{i,j}(k) \leftarrow \sum_{k' \in K} \exp\left(-q^x_{i+1,j}(k') - g^x_{i,j,i+1,j}(k,k')\right) \cdot Z^D_{i+1,j}(k'),\ k \in K$$

$$Z^R_{i,j}(k) \leftarrow \sum_{k' \in K} \exp\left(-q^x_{i,j+1}(k') - g^x_{i,j,i,j+1}(k,k')\right) \cdot Z^R_{i,j+1}(k'),\ k \in K$$

$$\beta^D_{i,j} \leftarrow \frac{\varepsilon}{\min\limits_{k' \in K} Z^D_{i,j}(k')},\ \beta^R_{i,j} \leftarrow \frac{\varepsilon}{\min\limits_{k' \in K} Z^R_{i,j}(k')}$$

$$Z^D_{i,j}(k) \leftarrow \beta^D_{i,j} Z^D_{i,j}(k),\ k \in K$$

$$Z^R_{i,j}(k) \leftarrow \beta^R_{i,j} Z^R_{i,j}(k),\ k \in K$$

$$f^D_{i,j}(k) \leftarrow \beta^D_{i,j} \sum_{k' \in K} \exp\left(-q^x_{i+1,j}(k') - g^x_{i,j,i+1,j}(k,k')\right) \cdot \langle Z^D_{i+1,j}(k')k', f^D_{i+1,j}(k')\rangle^T,\ k \in K$$

$$f^R_{i,j}(k) \leftarrow \beta^R_{i,j} \sum_{k' \in K} \exp\left(-q^x_{i,j+1}(k') - g^x_{i,j,i,j+1}(k,k')\right) \cdot \langle Z^R_{i,j+1}(k')k', f^R_{i,j+1}(k')\rangle,\ k \in K$$

  end for

end for

for $i = \overline{1,h}$ **do**

  for $j = \overline{1,w}$ **do**

$$Z^U_{i,j}(k) \leftarrow \sum_{k' \in K} \exp\left(-q^x_{i-1,j}(k') - g^x_{i,j,i-1,j}(k,k')\right) \cdot Z^U_{i-1,j}(k'),\ k \in K$$

$$Z^L_{i,j}(k) \leftarrow \sum_{k' \in K} \exp\left(-q^x_{i,j-1}(k') - g^x_{i,j,i,j-1}(k,k')\right) \cdot Z^L_{i,j-1}(k'),\ k \in K$$

$$\beta^U_{i,j} \leftarrow \frac{\varepsilon}{\min\limits_{k' \in K} Z^U_{i,j}(k')},\ \beta^L_{i,j} \leftarrow \frac{\varepsilon}{\min\limits_{k' \in K} Z^L_{i,j}(k')}$$

$$Z^U_{i,j}(k) \leftarrow \beta^U_{i,j} Z^U_{i,j}(k),\ k \in K$$

$$Z^L_{i,j}(k) \leftarrow \beta^L_{i,j} Z^L_{i,j}(k),\ k \in K$$

$$f^U_{i,j}(k) \leftarrow \beta^U_{i,j} \sum_{k' \in K} \exp\left(-q^x_{i-1,j}(k') - g^x_{i,j,i-1,j}(k,k')\right) \cdot \langle f^U_{i-1,j}(k'), Z^U_{i-1,j}(k')k'\rangle^T,\ k \in K$$

$$f^L_{i,j}(k) \leftarrow \beta^L_{i,j} \sum_{k' \in K} \exp\left(-q^x_{i,j-1}(k') - g^x_{i,j,i,j-1}(k,k')\right) \cdot \langle f^L_{i,j-1}(k'), Z^L_{i,j-1}(k')k'\rangle,\ k \in K$$

  end for

end for

for $i = \overline{1,h}$ **do**

  for $j = \overline{1,w}$ **do**

$$Z^c_{i,j}(k) \leftarrow Z^D_{i,j}(k) Z^R_{i,j}(k) Z^U_{i,j}(k) Z^L_{i,j}(k),\ k \in K$$

$$Z_{i,j} \leftarrow \sum_{k \in K} \exp\left(-q^x_{i,j}(k)\right) \cdot Z^c_{i,j}(k)$$

$$T^x_{i,j} \leftarrow \frac{1}{Z_{i,j}} \sum_{k \in K} \exp\left(-q^x_{i,j}(k)\right) Z^c_{i,j}(k) \begin{bmatrix} & (Z^U_{i,j}(k))^{-1} f^U_{i,j}(k) & \\ (Z^L_{i,j}(k))^{-1} f^L_{i,j}(k) & k & (Z^R_{i,j}(k))^{-1} f^R_{i,j}(k) \\ & (Z^D_{i,j}(k))^{-1} f^D_{i,j}(k) & \end{bmatrix}$$

  end for

end for

for $i = \overline{1,h}$ **do**

  for $j = \overline{1,w}$ **do**

$$d_x(i,j) \leftarrow \frac{1}{h+w-1} \sum_{\langle l,m \rangle \in H \times W} T^x_{l,m}[i,j]$$ ▷ $[i,j]$ в $T^x_{l,m}[i,j]$ — координата в $d_x(i,j)$

  end for ▷ якщо $T^x_{l,m}[i,j]$ не існує, тоді $T^x_{l,m}[i,j] = 0$

end for

return $d_x : H \times W \to K$

стоподібних деревах нашого графу, а потім для кожного об'єкта потрібно усереднити всі відповідні йому значення вузлів усіх дерев, у яких він присутній. Тобто на відміну від попереднього алгоритму цей метод враховує обчислені значення у всьому хрестоподібному дереві, а не лише в його корені. Для реалізації цього алгоритму потрібно водночас з обчисленням відображень $Z^c_{i,j} : K \to \mathbb{R},\ c \in \{D,R,U,L\}$ рахувати значення функцій $f^D_{i,j} : K \to \mathbb{R}^{h-i+1},\ f^R_{i,j} : K \to \mathbb{R}^{w-j+1},\ f^U_{i,j} : K \to \mathbb{R}^i$ і $f^L_{i,j} : K \to \mathbb{R}^j$. Обчислюватися перераховані функції мають за формулами:

$$f^D_{i,j}(k) = \sum_{k' \in K} \exp\left(-q^x_{i+1,j}(k') - g^x_{i,j,i+1,j}(k,k')\right) \cdot \langle Z^D_{i+1,j}(k')k', f^D_{i+1,j}(k')\rangle^T, \quad (2.17)$$

$$f^R_{i,j}(k) = \sum_{k' \in K} \exp\left(-q^x_{i,j+1}(k') - g^x_{i,j,i,j+1}(k,k')\right) \cdot \langle Z^R_{i,j+1}(k')k', f^R_{i,j+1}(k')\rangle\ , \quad (2.18)$$

$$f^U_{i,j}(k) = \sum_{k' \in K} \exp\left(-q^x_{i-1,j}(k') - g^x_{i,j,i-1,j}(k,k')\right) \cdot \langle f^U_{i-1,j}(k'), Z^U_{i-1,j}(k')k'\rangle^T, \quad (2.19)$$

$$f^L_{i,j}(k) = \sum_{k' \in K} \exp\left(-q^x_{i,j-1}(k') - g^x_{i,j,i,j-1}(k,k')\right) \cdot \langle f^L_{i,j-1}(k'), Z^L_{i,j-1}(k')k'\rangle\ . \quad (2.20)$$

Тобто обчислення математичного сподівання на кожному дереві відбувається за тим самим принципом динамічного програмування, що й для випадку його пошуку на ланцюжках. Суттєвою відмінністю між випадками хрестоподібного дерева й ланцюжка полягає в останньому кроці алгоритму при безпосередньому обчисленні $T^x_{i,j}$ (математичне сподівання хрестоподібного дерева з коренем у вузлі $\langle i,j\rangle$):

$$T^x_{i,j} = \frac{1}{Z_{i,j}} \sum_{k \in K} \exp\left(-q^x_{i,j}(k)\right) Z^s_{i,j}(k) \begin{bmatrix} & \tilde{f}^U_{i,j}(k) & \\ \tilde{f}^L_{i,j}(k) & k & \tilde{f}^R_{i,j}(k) \\ & \tilde{f}^D_{i,j}(k) & \end{bmatrix}, \quad (2.21)$$

де визначене скорочення $Z^s_{i,j}(k) = Z^D_{i,j}(k)Z^R_{i,j}(k)Z^U_{i,j}(k)Z^L_{i,j}(k),\ k \in K$, константа $Z_{i,j} = \sum\limits_{k \in K} \exp\left(-q^x_{i,j}(k)\right) \cdot Z^s_{i,j}(k)$ й набір відображень $\tilde{f}^c_{i,j}(k) = (Z^c_{i,j}(k))^{-1} f^c_{i,j}(k)$, $k \in K,\ c \in \{D,R,U,L\}$. В кінці спроектованого алгоритму для отримання мапи зсувів

$d_x : T \to \mathbb{R}$ пропонується для кожного об'єкту $\langle i,j \rangle \in T$ усереднити відповідні їм значення з усіх дерев, де вони зустрічалися. З повною версією цієї модифікації алгоритму SGM для знаходження мапи зсувів $d_x : T \to \mathbb{R}$ можна ознайомитися на попередніх сторінках (алгоритм 6). Весь наведений алгоритм має поліноміальну складність $O(|T|^2|K|^2)$, де $|T| = h \cdot w$.

2.4 Порівняння реалізованих алгоритмів

Всі алгоритми були реалізовані з використанням мови програмування Python. В процесі виконання роботи були використані бібліотеки numpy для обчислень, imageio для забезпечення роботи із зображеннями і matplotlib для отримання візуального представлення результату. Також використовувалася бібліотека time для вимірювання часу роботи алгоритмів.

Побудовані алгоритми тестувалися на зображеннях з Middlebury Stereo Datasets 2006 [19, 20], а саме Cloth1 (ширина: 417, висота: 370) і Bowling2 (ширина: 443, висота: 370).

Тестування проводилося для звичайного алгоритму на ланцюжках, який шукає розмітку з найбільшою апостеріорною ймовірністю (CA), для алгоритму на ланцюжках, який шукає математичне сподівання (Expected CA), для SGM та модифікації SGM — Expected SGM.

В ролі метрики для оцінки якості й ефективності реалізованих алгоритмів часто використовується середньоквадратичне відхилення (*root-mean-squared error* [21]). Якщо у нас є істинна мапа зсувів (*ground truth*) $d_x^T : T \to \mathbb{R}$ і знайдена $d_x^C : T \to \mathbb{R}$,

то середньоквадратичне відхилення може бути пораховане за формулою

$$R = \left(\frac{1}{|T|} \sum_{\langle i,j \rangle \in T} \left(d_x^C(i,j) - d_x^T(i,j) \right)^2 \right)^{\frac{1}{2}}. \tag{2.22}$$

Іншою метрикою для оцінки роботи алгоритмів, яку традиційно використовують у задачі стереозору, є відсоткова міра кількості погано визначених пікселів (*percentage of bad matching pixels* [21]) . Вона може бути обчислена у вигляді

$$B = \frac{1}{|T|} \sum_{\langle i,j \rangle \in T} \mathbb{1}_{(\delta,+\infty)} \left(\left| d_x^C(i,j) - d_x^T(i,j) \right| \right), \tag{2.23}$$

де $\delta \geqslant 0$ — порогове значення чутливості до помилки зсуву. В цій роботі ми будемо надалі при тестуванні вважати, що $\delta = 1$.

Ми будемо використовувати модифікації перелічених метрик, оскільки істинні мапи зсувів з використаного набору зображень приймають значення нуль при невідомому зсуву. Будемо їх обчислювати, використовуючи наступні формули:

$$R^* = \left(\frac{1}{|T'_x|} \sum_{\langle i,j \rangle \in T'_x} \left(d_x^C(i,j) - d_x^T(i,j) \right)^2 \right)^{\frac{1}{2}}, \tag{2.24}$$

$$B^* = \frac{1}{|T'_x|} \sum_{\langle i,j \rangle \in T'_x} \mathbb{1}_{(\delta,+\infty)} \left(\left| d_x^C(i,j) - d_x^T(i,j) \right| \right), \tag{2.25}$$

де множина $T'_x = T \setminus \{ \langle i,j \rangle : d_x^T(i,j) = 0,\ \langle i,j \rangle \in T \}$.

Також окрім вищенаведених метрик в роботі використовується модифікація середнього абсолютного відхилення (*mean absolute error*), яку можна обчислити, використовуючи формулу

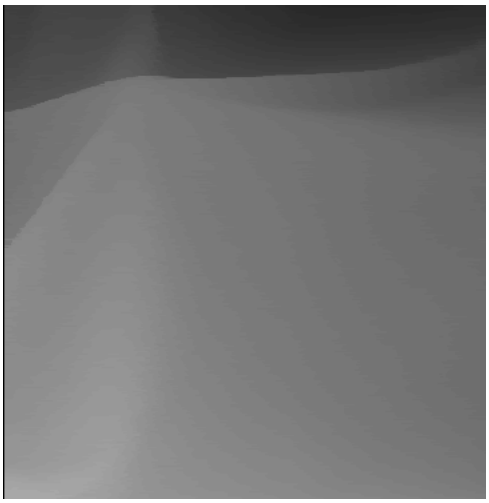
$$M^* = \frac{1}{|T'_x|} \sum_{\langle i,j \rangle \in T'_x} \left| d_x^C(i,j) - d_x^T(i,j) \right|. \tag{2.26}$$

В усіх алгоритмах використовувалися функції $q_t^x(k) = \|x_t^L - x_{t+k}^R\|_2$, $\langle x^L, x^R \rangle \in X_{sp}$ й $g_{tt'}^x(k,k') = \alpha\|k - k'\|_2$, де $k \in K$ і $K = \{0,\ldots,n\}$. Параметр $\alpha > 0$ є деякою визначеною сталою.

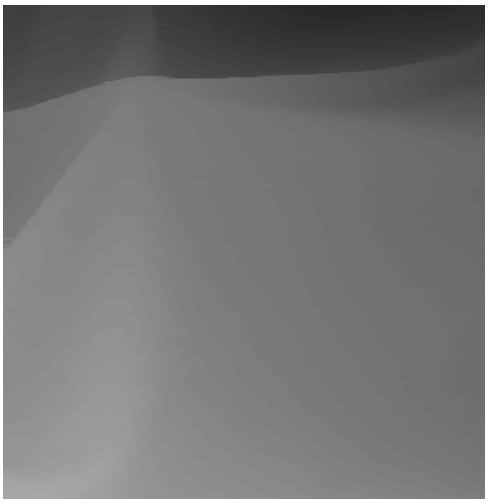
Отримані мапи зсувів мають таку саму висоту, що й вхідні зображення. Ширина отриманих мап зсувів: 361 (Cloth1), 376 (Bowling2). З результати тестування алгоритмів можна ознайомитися на наступних сторінках (рисунок 2.1, таблиця 2.1 — Cloth1, рисунок 2.2, таблиця 2.2 — Bowling2).

З отриманих результатів можна зробити висновок, що алгоритми, які вирішують задачу мінімізації ризику з квадратичною штрафною функцією — тобто обчислюють математичне сподівання, виконують своє завдання набагато повільніше за алгоритми, які шукаю найбільш ймовірну розмітку. З іншого боку, як бачимо, алгоритми Expected CA й Expected SGM краще мінімізують середньоквадратичне відхилення і навіть середнє абсолютне відхилення (хоча таке завдання не ставилося), ніж алгоритми CA й звичайний SGM.

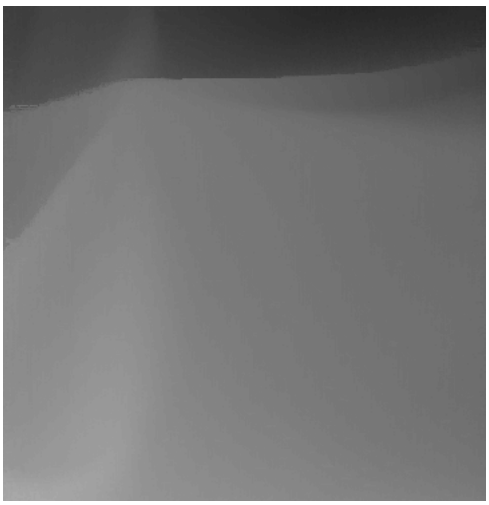
Візуально результати, отримані при використанні алгоритмів SGM і Expected SGM, виглядають краще. Видно, що враховується зв’язок між горизонтальними лініями стереопар. Алгоритми, які з відповідними лініями працюють незалежно, не володіють такою властивістю.



(а) CA



(б) Expected CA



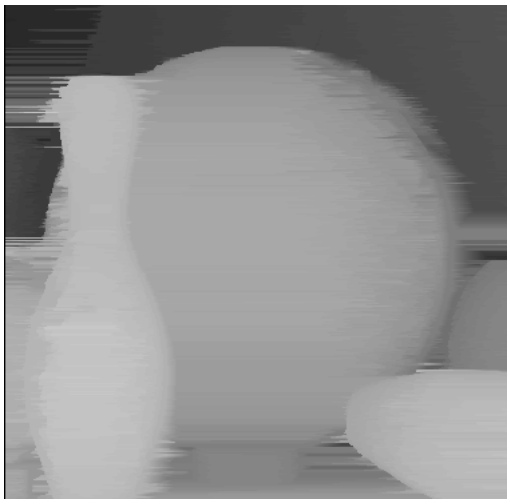
(в) SGM

(г) Expected SGM

Рисунок 2.1 — Мапи зсувів (Cloth1)

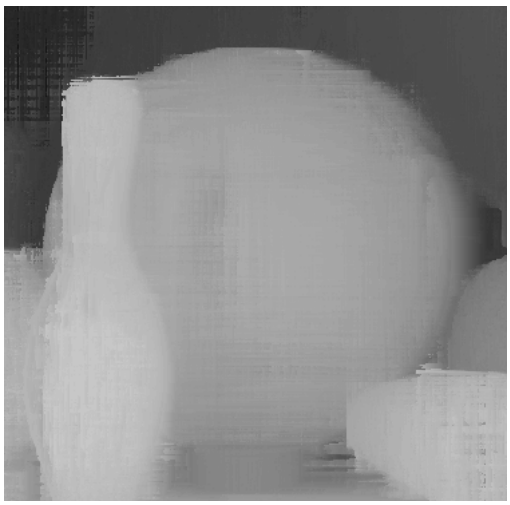
Таблиця 2.1 — Результати (Cloth1)

Алгоритм	$\|K\|$	α	Час, с	R^*	B^*, %	M^*
CA	57	18	19.773	2.214	0.549	0.382
Expected CA	57	18	1045.988	0.383	0.314	0.258
SGM	57	28	349.977	0.546	1.175	0.331
Expected SGM	57	28	3729.535	0.532	1.228	0.319

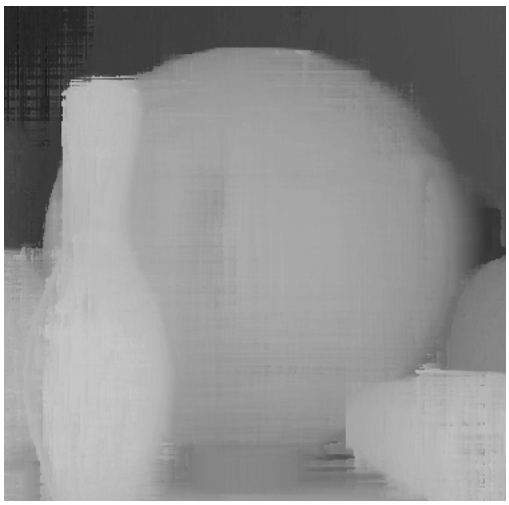


(а) CA

(б) Expected CA



(в) SGM



(г) Expected SGM

Рисунок 2.2 — Мапи зсувів (Bowling2)

Таблиця 2.2 — Результати (Bowling2)

Алгоритм	$\|K\|$	α	Час, с	R^*	B^*, %	M^*
CA	68	20	13.739	5.375	24.253	1.807
Expected CA	68	20	1397.967	4.876	26.516	1.683
SGM	68	10	387.747	3.621	31.757	1.574
Expected SGM	68	10	4800.321	3.597	33.814	1.545

Висновки до розділу 2

В другому розділі нам вдалося побудувати алгоритм Expected SGM. Проте під час реалізації алгоритму виявилося, що виникає проблема машинного нуля, тобто алгоритм є чисельно нестійким. Проблему вдалося частково вирішити шляхом введення спеціальних коефіцієнтів, які використовуються при обчисленні математичного сподівання. Ці коефіцієнти не впливають на коректність результатів обчислення і водночас збільшують стабільність роботи алгоритму.

Під час проведення тестування алгоритмів були практично підтверджені теоретичні припущення, які були сформульовані в попередньому розділі. Модифікація Expected SGM, яка шукає мапи зсувів у вигляді математичного сподівання усіх можливих розміток, краще мінімізує середньоквадратичне відхилення від істинної мапи зсувів, ніж оригінальний SGM.

ВИСНОВКИ

Під час виконання роботи вдалося побудувати нову модифікацію алгоритму SGM — Expected SGM. Ідея модифікації ґрунтується на перевизначенні мапи зсувів в оригінальному алгоритмі. Тепер вона шукається у вигляді математичного сподівання усіх можливих розміток. Теоретично це має допомогти зробити квадратичний штраф меншим, ніж в оригінальному SGM, що підтвердилося експериментально.

Для побудови Expected SGM ми проаналізували задачу стереозору, а також різні методи пошуку її розв'язку. Під час аналізу задачі ми виявили, що однією з ключових проблем є велика складність нашої задачі при виборі складної структури сусідства. Не існує відомого алгоритму, який би знаходив точний розв'язок відповідної задачі за поліноміальний час. Тому ми далі вирішили зосередитися на алгоритмі SGM, який шукає наближений розв'язок, і побудові його модифікації через його простоту й ефективність.

При програмній реалізації модифікації виникла проблема машинного нуля, тобто чисельна нестійкість алгоритму. В роботі проблема була частково вирішена визначенням спеціального коефіцієнта, який множиться на відповідні значення. Але це не вирішує проблему повністю. Тому ця задача може стати предметом наступних досліджень, які будуть намагатися покращити описаний у цій роботі алгоритм.

За результатами порівняння алгоритму Expected SGM з іншими алгоритмами розв'язування задачі стереозору нам вдалося експериментальним чином підтвердити, що побудований алгоритм краще мінімізує середньоквадратичне відхилення, а також середнє абсолютне відхилення, ніж інші алгоритми.

Подальші дослідження в цій області можуть також бути спрямовані на спроби реалізації алгоритму на більш низькому рівні, що може значним чином прискорити його роботу.

ПЕРЕЛІК ПОСИЛАНЬ

1 Hirschmüller, Heiko. Semi-Global Matching: Motivation, Development and Applications [Text] / Heiko Hirschmüller. — 2011. — 09. — Pp. 173–184.

2 Kolmogorov, V. Convergent Tree-Reweighted Message Passing for Energy Minimization [Text] / V. Kolmogorov // *IEEE Transactions on Pattern Analysis and Machine Intelligence*. — 2006. — Vol. 28, no. 10. — Pp. 1568–1583.

3 Fast hierarchical implementation of sequential tree-reweighted belief propagation for probabilistic inference [Text] / Skand Hurkat, Jungwook Choi, Eriko Nurvitadhi et al. // 2015 25th International Conference on Field Programmable Logic and Applications (FPL). — 2015. — Pp. 1–8.

4 Schlesinger, Michail. Ten Lectures on Statistical and Structural Pattern Recognition [Text] / Michail Schlesinger, Vaclav Hlavac. — 2002. — 01. — Vol. 24.

5 Geman, Stuart. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images [Text] / Stuart Geman, Donald Geman // *IEEE Transactions on Pattern Analysis and Machine Intelligence*. — 1984. — Vol. PAMI-6, no. 6. — Pp. 721–741.

6 Li, S. Markov Random Field Modeling in Computer Vision [Text] / S. Li // Computer Science Workbench. — 1995.

7 Winkler, Gerhard. Random Fields [Text] / Gerhard Winkler // Image Analysis, Random Fields and Dynamic Monte Carlo Methods: A Mathematical Introduction. — Berlin, Heidelberg: Springer Berlin Heidelberg, 1995. — Pp. 47–61. https://doi.org/10.1007/978-3-642-97522-6_4.

8 Markov Random Fields and Their Applications [Text] / R. Kindermann, J.L. Snell,

American Mathematical Society, Karreman Mathematics Research Collection // Contemporary mathematics - American Mathematical Society. — American Mathematical Society, 1980. https://books.google.com.ua/books?id=NeVQAAAAMAAJ.

9 Hariyama, Masanori. Design of a Trinocular-Stereo-Vision VLSI Processor Based on Optimal Scheduling [Text] / Masanori Hariyama, Naoto Yokoyama, Michitaka Kameyama // *IEICE Transactions on Electronics*. — 2008. — 04. — Vol. 91-C. — Pp. 479–486.

10 Szeliski, Richard. Computer Vision: Algorithms and Applications [Text] / Richard Szeliski. — 2022. — 01.

11 A Comparative Study of Energy Minimization Methods for Markov Random Fields with Smoothness-Based Priors [Text] / Richard Szeliski, Ramin Zabih, Daniel Scharstein et al. // *IEEE Transactions on Pattern Analysis and Machine Intelligence*. — 2008. — Vol. 30, no. 6. — Pp. 1068–1080.

12 Bellman, R. Dynamic Programming [Text] / R. Bellman, R.E. Bellman, Rand Corporation // Rand Corporation research study. — Princeton University Press, 1957. https://books.google.com.ua/books?id=rZW4ugAACAAJ.

13 Lesk, Arthur. Introduction to Bioinformatics [Text] / Arthur Lesk. — 2002. — 01.

14 Jurafsky, Daniel. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition [Text] / Daniel Jurafsky, James Martin. — 2008. — 02. — Vol. 2.

15 Veksler, Olga. Efficient Graph-Based Energy Minimization Methods In Computer Vision [Text] / Olga Veksler, Olga D. — 2001. — 05.

16 Hirschmuller, Heiko. Stereo Processing by Semiglobal Matching and Mutual Information [Text] / Heiko Hirschmuller // *IEEE Transactions on Pattern Analysis and Machine Intelligence*. — 2008. — Vol. 30, no. 2. — Pp. 328–341.

17 Seki, Akihito. SGM-Nets: Semi-Global Matching with Neural Networks [Text] / Akihito Seki, Marc Pollefeys. — 2017. — 07. — Pp. 6640–6649.

18 Schönberger, Johannes. Learning to Fuse Proposals from Multiple Scanline Optimizations in Semi-Global Matching [Text] / Johannes Schönberger, Sudipta Sinha, Marc Pollefeys. — 2018. — 09.

19 Scharstein, Daniel. Learning Conditional Random Fields for Stereo [Text] / Daniel Scharstein, Chris Pal // 2007 IEEE Conference on Computer Vision and Pattern Recognition. — 2007. — Pp. 1–8.

20 Hirschmuller, Heiko. Evaluation of Cost Functions for Stereo Matching [Text] / Heiko Hirschmuller, Daniel Scharstein // 2007 IEEE Conference on Computer Vision and Pattern Recognition. — 2007. — Pp. 1–8.

21 Scharstein, Daniel. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms [Text] / Daniel Scharstein, Richard Szeliski // *International Journal of Computer Vision*. — 2002. — 04. — Vol. 47. — Pp. 7–42.

ДОДАТОК А ТЕКСТИ ПРОГРАМ

```
import numpy as np
import imageio as im
import matplotlib.pyplot as plt
import time



def alg_data(disp_c, disp_t, delt = 1):
   h, w = disp_t.shape[0], disp_t.shape[1]
   rec_val = np.where(disp_t != 0, np.ones((h,w)), np.zeros((h,w)))
   R = np.sqrt(np.sum(((disp_c - disp_t) ** 2) * rec_val) / np.sum(rec_val))
   B = np.sum(np.where(np.abs(disp_c - disp_t) > delt, np.ones((h,w)),
      np.zeros((h,w))) * rec_val) / (np.sum(rec_val))
   M = np.sum(np.abs(disp_c - disp_t) * rec_val) / (np.sum(rec_val))
   return R, B, M



def q_ca(XL, XR, j, k):
   x, y = XL.shape[0], XL.shape[1]
   return np.linalg.norm(XL[:,j] - np.where(np.rollaxis((j + np.arange(0,k+1) < y)
      + np.zeros((x,3,k+1)), 2), np.swapaxes(XR[:,(j + np.arange(0,k+1)) * (j +
      np.arange(0,k+1) < y)],0,1), np.inf + np.zeros((k+1,x,3))), axis = 2).T

def CA(XL, XR, c = 18., k = 10):
   x, y = XL.shape[0], XL.shape[1] - k
   X = np.mgrid[0:x,0:y,0:k+1][2].astype(float)
   X[:,0] = q_ca(XL, XR, 0, k)

   for i in range(1, y):
```

```
        X[:,i] = q_ca(XL, XR, i, k)
        X[:,i] = X[:,i] + np.min(X[:,i - 1] + c * np.rollaxis(np.zeros((x,k+1,k+1))
            + np.abs(np.mgrid[0:k+1,0:k+1][1] - np.mgrid[0:k+1,0:k+1][1].T), 1), axis
            = 2).T
        X[:,i - 1] = np.argmin(X[:,i - 1] + c * np.rollaxis(np.zeros((x,k+1,k+1)) +
            np.abs(np.mgrid[0:k+1,0:k+1][1] - np.mgrid[0:k+1,0:k+1][1].T), 1), axis =
            2).T

    D = np.zeros((x,y), dtype = int)
    D[:,y - 1] = np.argmin(X[:,y - 1], axis = 1)
    for i in reversed((range(1, y - 1))):
        D[:,i] = X[np.arange(0,x),i + np.zeros((x,), dtype = int),D[:,i +
            1]].astype(int)
    return D



def q_eca(XL, XR, j, k):
    x, y = XL.shape[0], XL.shape[1]
    return np.linalg.norm(XL[:,j] - np.where(np.rollaxis((j + np.arange(0,k+1) < y)
        + np.zeros((x,3,k+1)), 2), np.swapaxes(XR[:,(j + np.arange(0,k+1)) * (j +
        np.arange(0,k+1) < y)],0,1), np.inf + np.zeros((k+1,x,3))), axis = 2).T

def Exp_CA(XL, XR, c = 18., k = 10):
    x, y = XL.shape[0], XL.shape[1] - k
    X = np.mgrid[0:x,0:y,0:k+1][2].astype(np.float128)
    Z = np.ones((x,k+1)).astype(np.float128)

    eps = 300
    g = c * np.swapaxes(np.zeros((x,k+1,k+1)) + np.abs(np.mgrid[0:k+1,0:k+1][0] -
        np.mgrid[0:k+1,0:k+1][1]),0,1)
```

```
    for i in range(y-1):
        X[:,i] = Z * X[:,i]
        gamma = np.min(np.sum(np.exp(-g) * (np.exp(-q_eca(XL,XR,i,k)) * Z), axis =
            2).T, axis = 1)
        Z = (np.sum(np.exp(-g) * (np.exp(-q_eca(XL,XR,i,k)) * Z), axis = 2) *
            np.exp(-eps - np.log(gamma))).T
        X[:,:i+1] = np.sum(np.moveaxis(np.broadcast_to(np.exp(-g),
            (i+1,k+1,x,k+1)),0,1) * (np.exp(-q_eca(XL,XR,i,k)) *
            np.moveaxis(X[:,:i+1],0,1)), axis = 3).T *
            np.moveaxis(np.broadcast_to(np.exp(-eps - np.log(np.broadcast_to(gamma,
            (k+1,x)).T)), (i+1,x,k+1)),0,1)

    X[:,y-1] = Z * X[:,y-1]
    D = np.sum(np.moveaxis(np.broadcast_to(np.exp(-q_eca(XL,XR,y-1,k)),
        (y,x,k+1)),0,1) * X, axis = 2) /
        np.broadcast_to(np.sum(np.exp(-q_eca(XL,XR,y-1,k)) * Z, axis = 1), (y,x)).T

    return D



def SGM(kx,ky,c,XL,XR):
    nL = XL[:,:-kx].astype(float)
    nR = np.pad(XR.astype(float), ((ky,ky),(0,0),(0,0)), 'constant',
        constant_values = np.inf)

    XL = XL[:,:-kx]
    XR = XR[:,:-kx]

    h, w = XL.shape[0], XL.shape[1]
```

```
D = np.zeros((h, w, 2*ky+1, kx+1))
R = np.zeros((h, w, 2*ky+1, kx+1))
L = np.zeros((h, w, 2*ky+1, kx+1))
U = np.zeros((h, w, 2*ky+1, kx+1))

M = np.mgrid[0:2*ky+1,0:kx+1,0:2*ky+1,0:kx+1]
g = np.linalg.norm(np.concatenate((np.array([M[0]]),np.array([M[1]]))) -
    np.concatenate((np.array([M[2]]),np.array([M[3]]))), axis = 0)

for i in reversed(range(h - 1)):
   D[i] = np.min(np.rollaxis(np.broadcast_to(D[i+1],
       (2*ky+1,kx+1,w,2*ky+1,kx+1)), 2) +
       np.rollaxis(np.broadcast_to(np.swapaxes(np.linalg.norm(nL[i+1] -
       np.array([np.roll(nR[(i+np.arange(2*ky+1)) % nR.shape[0]], -j, axis = 1)
       for j in range(kx+1)])[:,:,0:w], axis = 3),0,1),
       (2*ky+1,kx+1,2*ky+1,kx+1,w)), 4)+c*np.broadcast_to(g,
       (w,2*ky+1,kx+1,2*ky+1,kx+1)), axis = (3,4))

for i in reversed(range(w - 1)):
   R[:,i] = np.min(np.rollaxis(np.broadcast_to(R[:,i+1],
       (2*ky+1,kx+1,h,2*ky+1,kx+1)), 2) +
       np.rollaxis(np.broadcast_to(np.linalg.norm(nL[:,i+1] -
       np.array([np.roll(np.swapaxes(nR, 0,1)[(i+np.arange(kx+1)) %
       nR.shape[1]], -j, axis = 1) for j in range(2*ky+1)])[:,:,ky:h+ky], axis =
       3), (2*ky+1,kx+1,2*ky+1,kx+1,h)), 4)+c*np.broadcast_to(g,
       (h,2*ky+1,kx+1,2*ky+1,kx+1)), axis = (3,4))

for i in range(1,w):
   L[:,i] = np.min(np.rollaxis(np.broadcast_to(L[:,i-1],
       (2*ky+1,kx+1,h,2*ky+1,kx+1)), 2) +
```

```
            np.rollaxis(np.broadcast_to(np.linalg.norm(nL[:,i-1] -
            np.array([np.roll(np.swapaxes(nR, 0,1)[(i+np.arange(kx+1)) %
            nR.shape[1]], -j, axis = 1) for j in range(2*ky+1)])[:,:,ky:h+ky], axis =
            3), (2*ky+1,kx+1,2*ky+1,kx+1,h)), 4)+c*np.broadcast_to(g,
            (h,2*ky+1,kx+1,2*ky+1,kx+1)), axis = (3,4))

    for i in range(1,h):
        U[i] = np.min(np.rollaxis(np.broadcast_to(U[i-1],
            (2*ky+1,kx+1,w,2*ky+1,kx+1)), 2) +
            np.rollaxis(np.broadcast_to(np.swapaxes(np.linalg.norm(nL[i-1] -
            np.array([np.roll(nR[(i+np.arange(2*ky+1)) % nR.shape[0]], -j, axis = 1)
            for j in range(kx+1)])[:,:,0:w], axis = 3),0,1),
            (2*ky+1,kx+1,2*ky+1,kx+1,w)), 4)+c*np.broadcast_to(g,
            (w,2*ky+1,kx+1,2*ky+1,kx+1)), axis = (3,4))

    Q = (D+R+L+U+np.rollaxis(np.rollaxis(np.linalg.norm(nL -
        np.array([[np.roll(np.roll(nR,-j,axis=1),-i,axis=0) for j in range(kx+1)]
        for i in range(2*ky+1)])[:,:,ky:h+ky,0:w],
        axis=4),3),3)).reshape((nL.shape[0],nL.shape[1],(2*ky+1)*(kx+1)))
    return np.argmin(Q, axis = 2) % (kx+1)



def Exp_SGM(kx,ky,c,XL,XR):
    nL = XL[:,:-kx].astype(float)
    nR = np.pad(XR.astype(float), ((ky,ky),(0,0),(0,0)), 'constant',
        constant_values = np.inf)

    XL = XL[:,:-kx]
    XR = XR[:,:-kx]
```

```
h, w = XL.shape[0], XL.shape[1]

ZD = np.ones((h, w, 2*ky+1, kx+1), dtype = np.float128)
ZR = np.ones((h, w, 2*ky+1, kx+1), dtype = np.float128)
ZL = np.ones((h, w, 2*ky+1, kx+1), dtype = np.float128)
ZU = np.ones((h, w, 2*ky+1, kx+1), dtype = np.float128)

M = np.mgrid[0:2*ky+1,0:kx+1,0:2*ky+1,0:kx+1]
g = c * np.linalg.norm(np.concatenate((np.array([M[0]]),np.array([M[1]]))) -
    np.concatenate((np.array([M[2]]),np.array([M[3]]))), axis = 0)

for i in reversed(range(h - 1)):
   gamma = np.broadcast_to(np.min(np.sum(np.exp(-g) *
       (np.exp(-np.rollaxis(np.broadcast_to(np.swapaxes(np.linalg.norm(nL[i+1] -
       np.array([np.roll(nR[(i+np.arange(2*ky+1)) % nR.shape[0]], -j, axis = 1)
       for j in range(kx+1)])[:,:,0:w], axis = 3),0,1),
       (2*ky+1,kx+1,2*ky+1,kx+1,w)), 4)) * np.rollaxis(np.broadcast_to(ZD[i+1],
       (2*ky+1,kx+1,w,2*ky+1,kx+1)),2)), axis = (3,4)), axis = (1,2)),
       (kx+1,2*ky+1,w)).T
   ZD[i] = np.sum(np.exp(-g) *
       (np.exp(-np.rollaxis(np.broadcast_to(np.swapaxes(np.linalg.norm(nL[i+1] -
       np.array([np.roll(nR[(i+np.arange(2*ky+1)) % nR.shape[0]], -j, axis = 1)
       for j in range(kx+1)])[:,:,0:w], axis = 3),0,1),
       (2*ky+1,kx+1,2*ky+1,kx+1,w)), 4)) * np.rollaxis(np.broadcast_to(ZD[i+1],
       (2*ky+1,kx+1,w,2*ky+1,kx+1)),2)), axis = (3,4))
   ZD[i] = ZD[i] * np.exp(-300 - np.log(gamma))

gamma = np.broadcast_to(np.min(ZD, axis = (2,3)).T, (kx+1,2*ky+1,w,h)).T
ZD = ZD * np.exp(-150 - np.log(gamma))

for i in reversed(range(w - 1)):
```

```
    gamma = np.broadcast_to(np.min(np.sum(np.exp(-g) *
        (np.exp(-np.rollaxis(np.broadcast_to(np.linalg.norm(nL[:,i+1] -
        np.array([np.roll(np.swapaxes(nR, 0,1)[(i+np.arange(kx+1)) %
        nR.shape[1]], -j, axis = 1) for j in range(2*ky+1)])[:,:,ky:h+ky], axis =
        3), (2*ky+1,kx+1,2*ky+1,kx+1,h)), 4)) *
        np.rollaxis(np.broadcast_to(ZR[:,i+1], (2*ky+1,kx+1,h,2*ky+1,kx+1)),2)),
        axis = (3,4)), axis = (1,2)), (kx+1,2*ky+1,h)).T
    ZR[:,i] = np.sum(np.exp(-g) *
        (np.exp(-np.rollaxis(np.broadcast_to(np.linalg.norm(nL[:,i+1] -
        np.array([np.roll(np.swapaxes(nR, 0,1)[(i+np.arange(kx+1)) %
        nR.shape[1]], -j, axis = 1) for j in range(2*ky+1)])[:,:,ky:h+ky], axis =
        3), (2*ky+1,kx+1,2*ky+1,kx+1,h)), 4)) *
        np.rollaxis(np.broadcast_to(ZR[:,i+1], (2*ky+1,kx+1,h,2*ky+1,kx+1)),2)),
        axis = (3,4))
    ZR[:,i] = ZR[:,i] * np.exp(-300 - np.log(gamma))

gamma = np.broadcast_to(np.min(ZR, axis = (2,3)).T, (kx+1,2*ky+1,w,h)).T
ZR = ZR * np.exp(-150 - np.log(gamma))

for i in range(1,w):
    gamma = np.broadcast_to(np.min(np.sum(np.exp(-g) *
        (np.exp(-np.rollaxis(np.broadcast_to(np.linalg.norm(nL[:,i-1] -
        np.array([np.roll(np.swapaxes(nR, 0,1)[(i+np.arange(kx+1)) %
        nR.shape[1]], -j, axis = 1) for j in range(2*ky+1)])[:,:,ky:h+ky], axis =
        3), (2*ky+1,kx+1,2*ky+1,kx+1,h)), 4)) *
        np.rollaxis(np.broadcast_to(ZL[:,i-1], (2*ky+1,kx+1,h,2*ky+1,kx+1)),2)),
        axis = (3,4)), axis = (1,2)), (kx+1,2*ky+1,h)).T
    ZL[:,i] = np.sum(np.exp(-g) *
        (np.exp(-np.rollaxis(np.broadcast_to(np.linalg.norm(nL[:,i-1] -
        np.array([np.roll(np.swapaxes(nR, 0,1)[(i+np.arange(kx+1)) %
        nR.shape[1]], -j, axis = 1) for j in range(2*ky+1)])[:,:,ky:h+ky], axis =
```

```
        3), (2*ky+1,kx+1,2*ky+1,kx+1,h)), 4)) *
        np.rollaxis(np.broadcast_to(ZL[:,i-1], (2*ky+1,kx+1,h,2*ky+1,kx+1)),2)),
        axis = (3,4))
    ZL[:,i] = ZL[:,i] * np.exp(-300 - np.log(gamma))

gamma = np.broadcast_to(np.min(ZL, axis = (2,3)).T, (kx+1,2*ky+1,w,h)).T
ZL = ZL * np.exp(-150 - np.log(gamma))

for i in range(1,h):
    gamma = np.broadcast_to(np.min(np.sum(np.exp(-g) *
        (np.exp(-np.rollaxis(np.broadcast_to(np.swapaxes(np.linalg.norm(nL[i-1] -
        np.array([np.roll(nR[(i+np.arange(2*ky+1)) % nR.shape[0]], -j, axis = 1)
        for j in range(kx+1)])[:,:,0:w], axis = 3),0,1),
        (2*ky+1,kx+1,2*ky+1,kx+1,w)), 4)) * np.rollaxis(np.broadcast_to(ZU[i-1],
        (2*ky+1,kx+1,w,2*ky+1,kx+1)),2)), axis = (3,4)), axis = (1,2)),
        (kx+1,2*ky+1,w)).T
    ZU[i] = np.sum(np.exp(-g) *
        (np.exp(-np.rollaxis(np.broadcast_to(np.swapaxes(np.linalg.norm(nL[i-1] -
        np.array([np.roll(nR[(i+np.arange(2*ky+1)) % nR.shape[0]], -j, axis = 1)
        for j in range(kx+1)])[:,:,0:w], axis = 3),0,1),
        (2*ky+1,kx+1,2*ky+1,kx+1,w)), 4)) * np.rollaxis(np.broadcast_to(ZU[i-1],
        (2*ky+1,kx+1,w,2*ky+1,kx+1)),2)), axis = (3,4))
    ZU[i] = ZU[i] * np.exp(-300 - np.log(gamma))

gamma = np.broadcast_to(np.min(ZU, axis = (2,3)).T, (kx+1,2*ky+1,w,h)).T
ZU = ZU * np.exp(-150 - np.log(gamma))

K = np.broadcast_to(np.arange(kx+1), (h,w,2*ky+1,kx+1))
Q = np.sum(np.exp(-np.rollaxis(np.rollaxis(np.linalg.norm(nL -
    np.array([[np.roll(np.roll(nR,-j,axis=1),-i,axis=0) for j in range(kx+1)]
    for i in range(2*ky+1)])[:,:,ky:h+ky,0:w], axis=4),3),3)) * (ZD * ZR * ZL *
```

```
        ZU * K), axis = (2,3)) /
        np.sum(np.exp(-np.rollaxis(np.rollaxis(np.linalg.norm(nL -
        np.array([[np.roll(np.roll(nR,-j,axis=1),-i,axis=0) for j in range(kx+1)]
        for i in range(2*ky+1)])[:,:,ky:h+ky,0:w], axis=4),3),3)) * (ZD * ZR * ZL *
        ZU), axis = (2,3))
    return Q



def disp_test(c1, c2, n_val, sp):
    L = np.array(im.imread(f'drive/MyDrive/SGM_alg/{sp}/view5.png')).astype(int)
    R = np.array(im.imread(f'drive/MyDrive/SGM_alg/{sp}/view1.png')).astype(int)
    T_im = np.array(im.imread(f'drive/MyDrive/SGM_alg/{sp}/disp5.png')).astype(int)

    print('CA info:\n')
    start_time = time.time()
    disp_map = CA(L, R, c = c1, k = n_val)
    print('time of execution: ',time.time() - start_time)
    print(alg_data(disp_map, T_im[:,:-n_val] / 3))
    fig = plt.figure(figsize=(12, 8))
    ax1 = fig.add_subplot( 111 )
    ax1.imshow( disp_map, cmap=plt.get_cmap( 'gray' ) )
    plt.show()

    print('Expected CA info:\n')
    start_time = time.time()
    disp_map = Exp_CA(L, R, c = c1, k = n_val)
    print('time of execution: ',time.time() - start_time)
    print(alg_data(disp_map, T_im[:,:-n_val] / 3))
    fig = plt.figure(figsize=(12, 8))
    ax1 = fig.add_subplot( 111 )
```

```
    ax1.imshow( disp_map, cmap=plt.get_cmap( 'gray' ) )
    plt.show()

    print('SGM info:\n')
    start_time = time.time()
    disp_map = SGM(kx = n_val, ky = 2, c = c2, XL = L, XR = R)
    print('time of execution: ',time.time() - start_time)
    print(alg_data(disp_map, T_im[:,:-n_val] / 3))
    fig = plt.figure(figsize=(12, 8))
    ax1 = fig.add_subplot( 111 )
    ax1.imshow( disp_map, cmap=plt.get_cmap( 'gray' ) )
    plt.show()

    print('Expected SGM info:\n')
    start_time = time.time()
    disp_map = Exp_SGM(kx = n_val, ky = 2, c = c2, XL = L, XR = R)
    print('time of execution: ',time.time() - start_time)
    print(alg_data(disp_map, T_im[:,:-n_val] / 3))
    fig = plt.figure(figsize=(12, 8))
    ax1 = fig.add_subplot( 111 )
    ax1.imshow( disp_map, cmap=plt.get_cmap( 'gray' ) )
    plt.show()



disp_test(18, 28, 56, 'cloth1_2')
disp_test(20, 10, 67, 'bowl2_2')
```