4. ЗАДАЧА РОЗМІТКИ НА ДЕРЕВАХ ДЛЯ ПОСТУПОВОГО НАДХОДЖЕННЯ ДАНИХ

Іван Кириленко, аспірант
Кафедра математичного моделювання і аналізу даних
Навчально-науковий Фізико-технічний інститут
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

kirvandr@gmail.com

ВСТУП

Задача розмітки зустрічається в багатьох галузях науки, зокрема має широке поширення у сфері комп'ютерного зору: сегментація зображень [1-3], знешумлення, стереобачення [4, 5], SGM [2, 6] тощо. Такі алгоритми, зазвичай, працюють за умови наявності повної інформації про задачу. Проте, на практиці, отримання всіх даних може бути не миттєвим процесом. В такому випадку, до часу роботи самого алгоритму додається ще й час очікування на дані. Це можна було б частково компенсувати, якби мати змогу проводити розрахунки над вже отриманими даними, тим самим зменшивши роботу необхідну для отримання результату за надходження всіх даних.

В цій праці розглянуто проблему пошуку оптимальної розмітки дерев, в умовах, коли інформація, необхідна для розв'язання задачі, частково, або повністю відсутня на момент початку обрахунків, та надходить поступово, через деякі проміжки часу.

Також запропоновано підхід до вирішення задачі пошуку оптимальної розмітки дерев, який надає змогу проводити обчислення в міру надходження нових даних, при цьому поступово зменшуючи кількість операцій, що буде необхідна для розв'язання задачі по отриманню повної інформації.

4.1. ЗАДАЧА РОЗМІТКИ НА ДЕРЕВАХ

4.1.1. ПОСТАНОВКА ЗАДАЧІ

Припустимо, існує довільне дерево $\mathcal{T} =< \mathcal{V}, \mathcal{E} >$, де $\mathcal{V}$ - множина вузлів дерева, $\mathcal{E}$ – множина ребер, що сполучють ці вузли. Кожному вузлу дерева $v \in$ V може бути присвоєна мітка d_v із деякої множини міток D.

Розміткою дерева $\mathcal{T}$ будемо називати послідовність $\overline{d} \in D^{\mathcal{V}}$. Кожна така розмітка $\overline{d}$ матиме свою відповідну характеристику, яку часто називають ”якістю” або ”енергією”, і визначають як (1) (мінімізація енергії широко використовується в задачах комп’ютерноого зору, особливо після впровадження ефективних алгоритмів [6]).

$$E(d) = \sum_{v \in V} h(o_v, d_v) + \sum_{v,u \in E} g(d_v, d_u), \quad (1)$$

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

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

Тут $o_v \in \mathcal{O}$ це елемент з множини сигналів або спостережень, який відповідає вузлу v. Складова $h(o_v, d_v)$ відповідає за «консистентність» сигналу, отриманого на вузлі v, з міткою, яку ми обираємо для цього вузла. А $g(d_v, d_u)$ – складова «гладкості» розмітки, відповідає за те, щоб сусідні вузли мали схожі мітки.

Задача розмітки дерева $\mathcal{T}$ полягає у знаходженні оптимальної розмітки $\overline{d^*}$, яка б мінімізувала (або максимізувала) енергію $E(\overline{d^*})$.

$$\overline{d^*} \in \underset{\overline{d} \in D^{\mathcal{V}}}{argmin} \left(\sum_{v \in \mathcal{V}} h(o_v, d_v) + \sum_{v,u \in \mathcal{E}} g(d_v, d_u) \right). \quad (2)$$

4.1.2. ЗВЕДЕННЯ ЗАДАЧІ ПОШУКУ ОПТИМАЛЬНОЇ РОЗМІТКИ ДЕРЕВА ДО ЗАДАЧІ ПОШУКУ НАЙКРАЩОГО ШЛЯХУ НА ГРАФІ СПЕЦІАЛЬНОГО ВИГЛЯДУ

Проблема пошуку найкращого шляху на графі відноситься до класичних та ретельно досліджених класів задач. На сьогодні існує багато ефективних алгоритмів вирішення таких проблем [6, 7-12], тому можливість їх використання для вирішення задачі пошуку оптимальної розмітки є доволі привабливою перспективою.

Основна ідея зведення задачі полягає у побудові відповідного оригінальній задачі графа таким чином, щоб позбутися концепту міток на вузлах, перетворивши їх на елементи самого графу.

Візьмемо для прикладу деяке дерево-ланцюжок T (рис. 1) та покладемо множину міток $D = \{d1, d2, d3\}$,

Рис. 1. Дерево T

Тоді, при зведені задачі, новий граф матиме наступний вигляд (рис. 2):

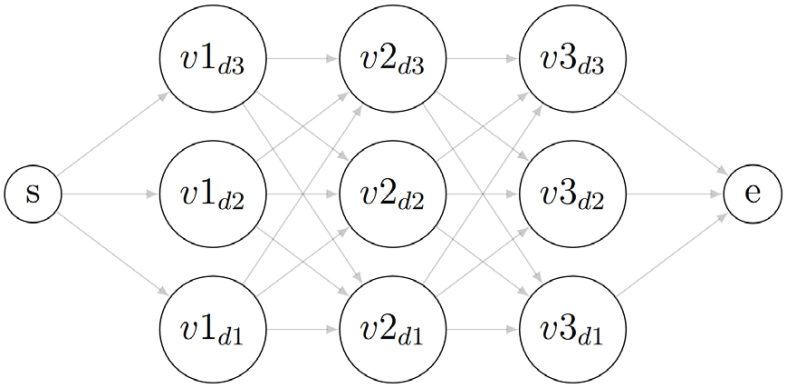


Рис. 2. Відповідний дереву T граф

Тут кожному вузлу v дерева T (окрім кореня дерева) відповідає стовпчик з $|D|$ вершин, кожна з яких з'єднана з усіма відповідними вершинами, що були сусідні до v у T. Тобто, будується новий, відповідний до оригінальної задачі, граф $G' = < \mathcal{V}', \mathcal{E}' >$:

$$V' = \{\sigma\_v(d) \mid v \in V \setminus \{s\}, d \in D\} \cup \{s\} \cup E^*,$$
$$E' = \{\langle \sigma\_v(d), \sigma\_u(d') \rangle \mid \langle v, u \rangle \in \mathcal{N}; d, d' \in D\}, \quad (3)$$

де E^* - множина кінцевих вершин, що додаються до кожного листа дерева T, а $\mathcal{N}$ - відношення сусідства вузлів дерева T.

Введемо функцію ваги вершини:

$$h: V' \to R,\ h\big(\sigma(v,d)\big) \equiv h(v,d). \tag{4}$$

Важливо зазначити, що хоч конкретний вигляд цієї функцій може бути дуже різноманітним і визначається природою задачі, в більшості випадків вона бути залежати від $o_v \in \mathcal{O}$ - сигналу, що надійшов на вузел v.

Функція ваги ребер, в загальному випадку, визначається як $g: \mathcal{E}' \to R$. Але нас цікавить лише "гладкість" розмітки, тож вага ребра не буде залежати від того, між якими вузлами дерева T воно проходить. А отже, можемо визначити функцію ваги ребра як:

$$g: D \times D \to \mathbb{R}. \tag{5}$$

Отримуємо такий зважений граф (рис. 3).

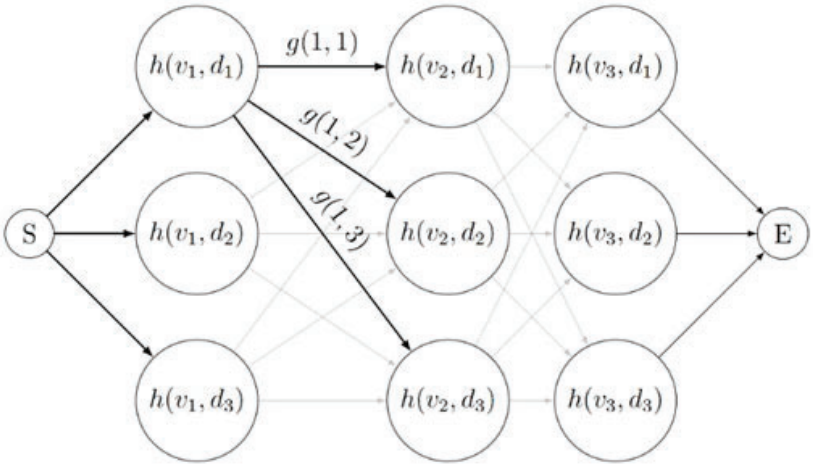


Рис. 3. Граф G

Обравши в побудованому графі G з кожного стовпчика вершин $\{\sigma_v(d), d \in D\}$ по одній вершині (кожній такій вершині відповідає певна мітка d_v), отримаєто деяку розміку $\overline{d}$, та шлях від s до e, загальна вага якого визначатиметься як:

$$\sum_{v\in V} h(v,d_v) + \sum_{v,u\in\mathcal{N}^*} g(d_v,d_u),$$
$$\mathcal{N}^* = \{\langle\sigma(v,d),\sigma(u,d')\rangle \in \mathcal{E}'; d,d' \in \overline{d}\}. \tag{6}$$

Таким чином, задача пошуку таких вершин, через які можна провести оптимальний шлях на побудованому графі G, відповідає задачі розмітки дерева $\mathcal{T}$ (2).

4.2. ЗАДАЧА РОЗМІТКИ ДЛЯ ВИПАДКУ ПОСТУПОВОГО НАДХОДЖЕННЯ ДАНИХ

Метод, описаний в попередньому підрозділі, передбачає, що для всіх вузлів v дерева T всі відповідні їм сигнали o_v з простору спостережень $\mathcal{O}$, вже відомі до початку обчислень. Однак, це не завжди так.

Цілком можливі ситуації, коли надходження даних сповільнено через неякісні/перевантажені канали зв'язку, або зумовлено самою природою практичної задачі - не є ефективним. За відсутності всієї інформації, ми не зможемо розрахувати ваги деяких вершин, а отже, і не зможемо бути певними в оптимальності шуканої розмітки.

Мета цієї роботи - розглянути випадок, коли ми не маємо повної інформації про простір спостережень $\mathcal{O}$, а отримуємо її поступово, через певні інтервали часу. Також на меті запропонувати ефективний алгоритм вирішення проблеми пошуку оптимальної розмітки, який дав би змогу проводи певні попередні обчислення з наявними даними, прискоривши отримання фінального результату при надходженні всієї інформації.

Дещо модифікуємо постановку задачі для такого випадку. Досі маємо дерево $T =< \mathcal{V}, \mathcal{E} >$, але тепер множина вузлів $\mathcal{V}$ буде розбита на дві підмножини:

$$\begin{aligned} \mathcal{V} &= \mathcal{V}^o \cup \mathcal{V}^c, \\ \mathcal{V}^o &\cap \mathcal{V}^c = \emptyset, \end{aligned} \tag{7}$$

де $\mathcal{V}^o$ - підмножина «відкритих» вузлів (ті, про які ми вже маємо інформацію), і $\mathcal{V}^c$ - підмножина «закритих» вузлів (інформація про які ще не надійшла). Так само, «закритими» або «відкритими» будуть і вершини у відповідному до дерева T графі G: $\sigma(v, d)$ – «відкрита», якщо $v \in \mathcal{V}^o$, і «закрита», якщо $v \in \mathcal{V}^c$. Будемо позначати відкриті і закриті вершини білим і чорним кольором відповідно (рис. 4).

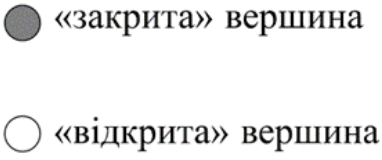


Рис. 4. Приклад позначення вершин

Наприклад, нехай у деякому дереві T відкрита одна вершина $\widehat{v_2}$ (рис. 5), тоді, у відповідному графі G, відкритими буде цілий стовпчик вершин $\sigma(v_2^*, d), d \in D$ (рис. 6).

Рис. 5. Один відкритий вузол

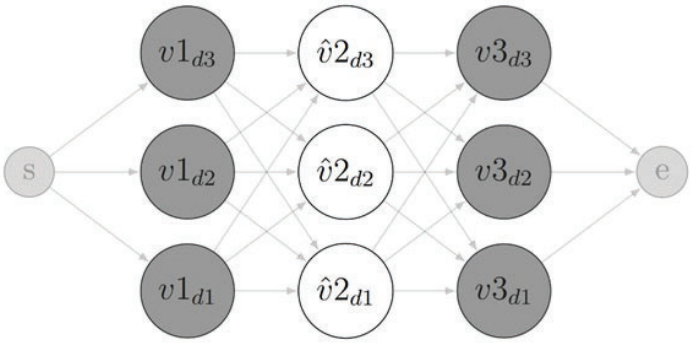


Рис. 6. Відповідний стовпчик відкритих вершин

Суть проблеми знаходження оптимальної розмітки в таких умовах полягає в тому, що обчислення можуть проводитися тільки з вагами відкритих вершин, та сусідніх їм ребер. Та, як буде показано в наступному підрозділі, навіть маючи тільки це, можна робити кроки у напрямку вирішення поставленої задачі.

4.2.1. ЗАДАЧА РОЗМІТКИ ДЛЯ ВИПАДКУ ПОСТУПОВОГО НАДХОДЖЕННЯ ДАНИХ

Ідея онлайн алгоритму пошуку оптимальної розмітки полягає у поступовій модифікації графу G, відповідно надходженню нової інформації про вузли дерева T, наступним чином: припустимо маємо інформацію про один вузол дерева T, тобто у G буде відкритий відповідний стовпчик вершин (рис. 7).

Ми можемо позбутися цих відкритих вершин таким чином, щоб зберегти сенс самої задачі. Для кожної вершини $\sigma(v', d),\ d \in D$ нас цікавить тільки найкращий можливий шлях у кожну вершину $\sigma(v'', d'),\ d' \in D$. Обрахувавши їх, нам більше не будуть потрібні жодна з вершин $\sigma(v^*, d^*),\ d^* \in D$, а також ребра

$\langle \sigma(v', d'), \sigma(v^*, d^*) \rangle$ та $\langle \sigma(v^*, d^*), \sigma(v'', d'') \rangle$, $d', d'', d^* \in D$. Замість них ми вводимо нові ребра (рис. 7), $\langle \sigma(v', d'), \sigma(v'', d'') \rangle$, $d', d'' \in D$.

Вага цих нових ребер буде містити у собі інформацію про видалені вершини, та визначатиметься як:

$$g_{new}\left(\sigma_{v'}(d'), \sigma_{v''}(d'')\right) = \arg\min_{d^* \in D} \left(g\left(\sigma_{v'}(d'), \sigma_{v^*}(d^*)\right) + h(v^*, d^*) + g\left(\sigma_{v^*}(d^*), \sigma_{v''}(d'')\right)\right), \quad (8)$$

$$d_{v^*} \in \arg\min_{d^* \in D} \left(g\left(\sigma_{v'}(d'), \sigma_{v^*}(d^*)\right) + h(v^*, d^*) + g\left(\sigma_{v^*}(d^*), \sigma_{v''}(d'')\right)\right).$$

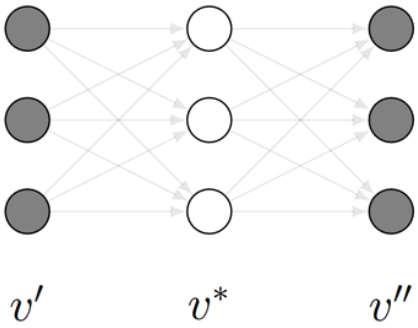


Рис. 7. До вилучення вершин

Позбувшись цього стовпчика вершин, ми зменшили загальну кількість ребер у графі G на $|D|^2$, та зменшили кількість вершин на $|D|$. При цьому ми зберегли загальну структуру задачі, а отже, для отримання фінального результату нам буде необхідно виконати меншу кількість операцій (рис. 8).

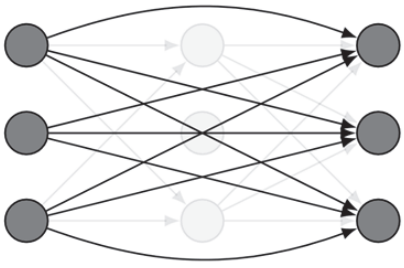


Рис. 8. Після вилучення стовпчику вершин

Для вузлів, що мають більше двох сусідів, або якщо один із сусідів є коренем, або листом дерева, алгоритм модифікації графа G буде схожим.

Для випадку вузла, з більш ніж двома сусідами (рис. 9), та якщо потужність множини $|D| = 2$, тоді матимемо таку конфігурацію графу G, як на рисунку (рис. 10).

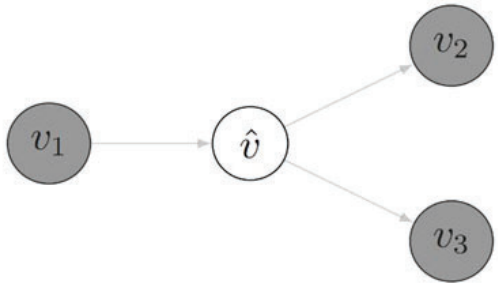


Рис. 9. Відкрита вершина з трьома сусідами

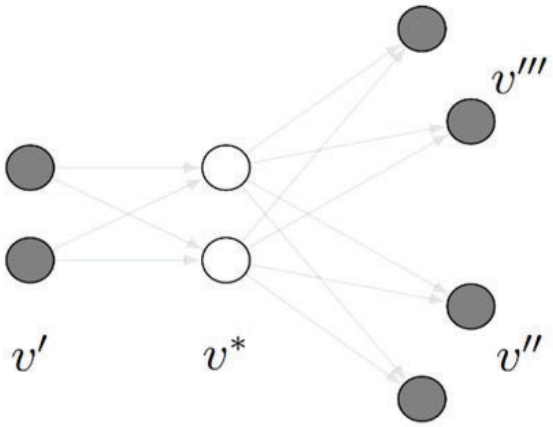


Рис. 10. До вилучення вершин

Вилучивши всі вершини $\sigma(v^*, d^*), d^* \in D$, а також ребра $\langle\sigma(v', d'), \sigma(v^*, d^*)\rangle$ та$\langle\sigma(v^*, d^*), \sigma(v'', d'')\rangle$ та $\langle\sigma(v^*, d^*), \sigma(v''', d''')\rangle$, $d', d'', d''', d^* \in D$, замінимо їх новими ребрами (рис. 11, рис. 12).

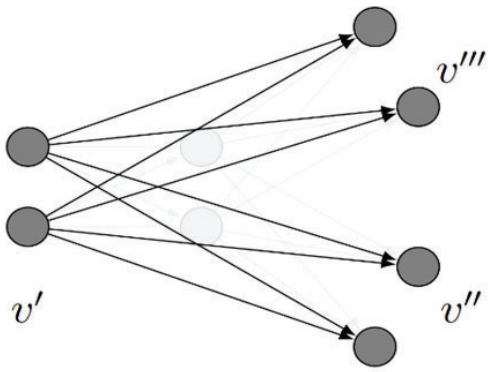


Рис. 11. Після вилучення вершин

$$g_{new}\left(\sigma_{v'}(d'), \sigma_{v''}(d'')\right) = \min_{d^* \in D}\left(g\left(\sigma_{v'}(d'), \sigma_{v^*}(d^*)\right) + h(v^*, d^*) + g\left(\sigma_{v^*}(d^*), \sigma_{v''}(d'')\right)\right), \quad (9)$$

$$g_{new}\left(\sigma_{v'}(d'), \sigma_{v'''}(d''')\right) = \min_{d^* \in D}\left(g\left(\sigma_{v'}(d'), \sigma_{v^*}(d^*)\right) + h(v^*, d^*) + g\left(\sigma_{v^*}(d^*), \sigma_{v'''}(d''')\right)\right). \quad (10)$$

$$d^{*'} \in arg \min_{d^* \in D}\left(g\left(\sigma_{v'}(d'), \sigma_{v^*}(d^*)\right) + h(v^*, d^*) + g\left(\sigma_{v^*}(d^*), \sigma_{v''}(d'')\right)\right), \quad (11)$$

$$d^{*''} \in arg \min_{d^* \in D}\left(g\left(\sigma_{v'}(d'), \sigma_{v^*}(d^*)\right) + h(v^*, d^*) + g\left(\sigma_{v^*}(d^*), \sigma_{v'''}(d''')\right)\right). \quad (12)$$

Модифікація графу G для вузлів, що є листами, або сусідами кореня дерева є найпростішим випадком. Наведемо приклад для вузла, який є сусідом кореня дерева T (рис. 13, рис. 14). Для вузлів-листків алгоритм ідентичний.

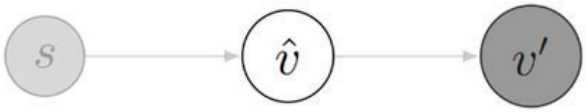


Рис. 12. Після вилучення вершин

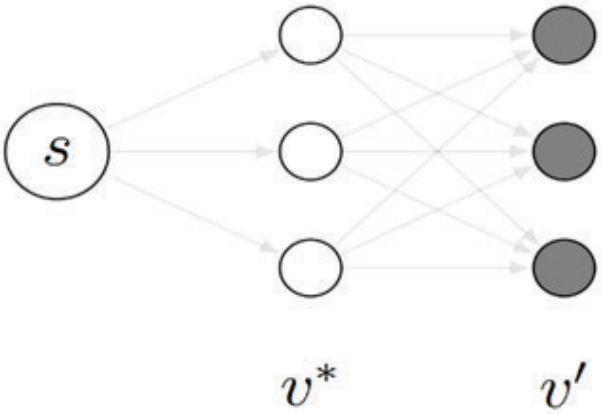


Рис. 13. До вилучення вершин

Вага нових ребер:

$$g_{new}\left(s, \sigma_{v'}(d')\right) = \min_{d^* \in D}\left(h(v^*, d^*) + g\left(\sigma_{v^*}(d^*), \sigma_{v'}(d')\right)\right), \quad (13)$$

$$d^* \in \underset{d^* \in D}{argmin} \Big(h(v^*, d^*) + g\big(\sigma_{v^*}(d^*), \sigma_{v'}(d')\big)\Big).$$

Описані вище дії виконуються в міру надходження інформації про сигнали $o_v \in \mathcal{O}$ на вузлах дерева T. Хоч алгоритм і має ітеративний характер, проте у випадку, коли одночасно надійшли дані про спостереження на декількох, не сусідніх між собою вузлах, їх оптимізація може бути проведена паралельно.

Після того, як всі вузли дерева T (рис. 15) стануть відкритими, а відповідні їх вершини і ребра графу G (рис. 16) будуть оптимізовані, ми отримаємо просту структуру - початкову вершину s, яка з'єднана з кінцевими вершинами $e_1, e_2, \ldots$ (рис. 17).

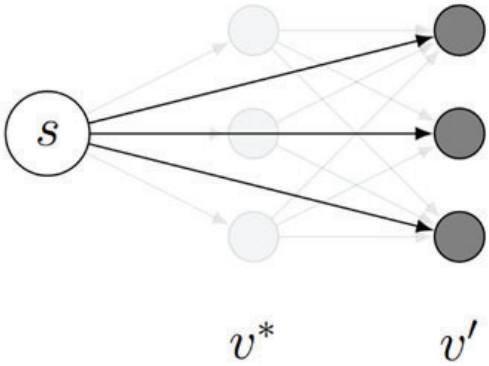


Рис. 14. Після вилучення вершин

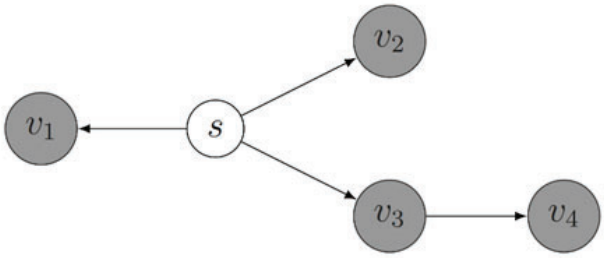


Рис. 15. Дерево T

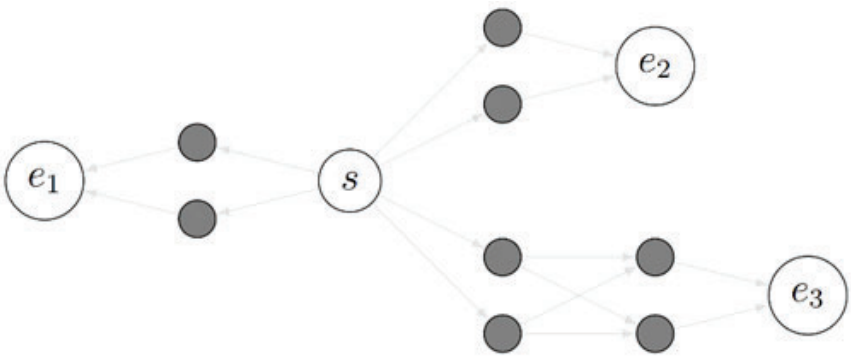


Рис. 16. Відповідний йому граф G до оптимізації (всі вузли закриті)

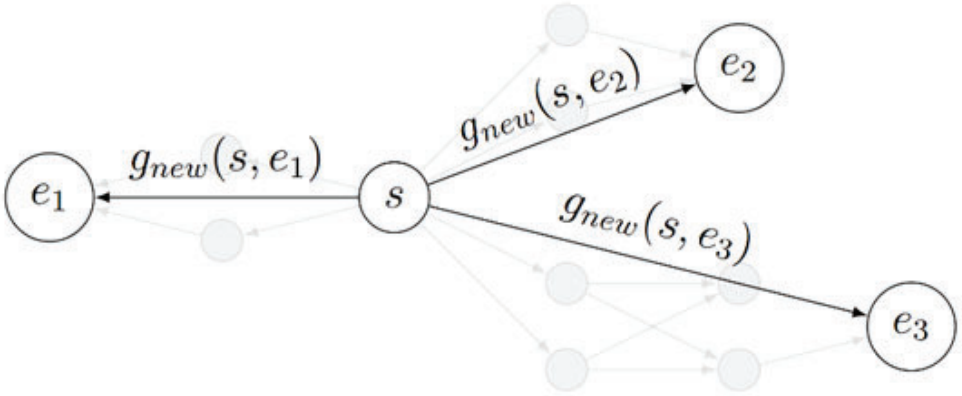


Рис. 17. Оптимізований граф G

Тоді мінімальна енергія для дерева T дорівнюватиме сумі ваг ребер в оптимізованому графі G (рис. 18), $E(\overline{d^*}) = \sum g_{new}(s, e_i)$. А сама послідовність $\overline{d^*}$ відновлюється по збереженим на кожному кроку оптимізації міткам кращого шляху.

Таким чином, на момент отримання всієї інформації про задачу, ми вже матимемо її рішення, у вигляді оптимальної розмітки $\overline{d^*}$, яку ми поступово знаходили безпосередньо під час надходження даних.

4.3. ЗАСТОСУВАННЯ ЗАПРОПОНОВАНОГО МЕТОДУ НА ПРИКЛАДІ ЗАДАЧІ СТЕРЕОЗОРУ

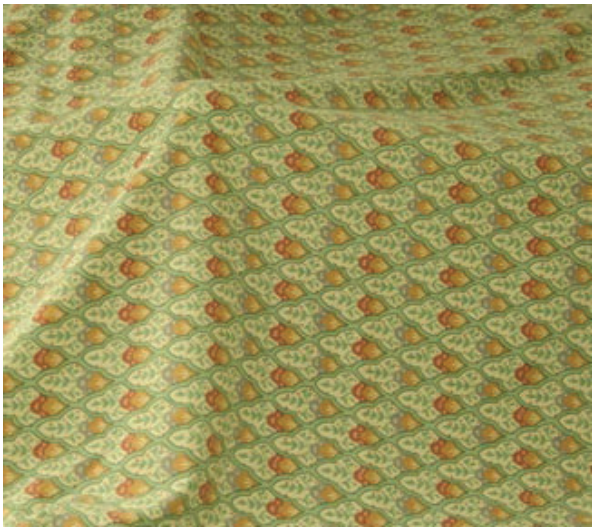
Розглянемо одне з можливих практичних застосувань запропонованого алгоритму пошуку розмітки на деревах для поступового надходження даних на прикладі задачі стереозору.

4.3.1. ЗАДАЧА СТЕРЕОЗОРУ

Задача стереозору полягає у відновлені інформації про тривимірну структуру сцени, маючи набір зображень цієї сцени (рис. 18), отриманих з різних ракурсів.

Підходів до пошуку карти глибини сцени на одному і тому ж наборі зображень існує декілька [6, 7], та для демонстрації принципу роботи запропонованого в цьому розділіті методу, ми зупинимось на одновимірному алгоритмі пошуку карти зсувів. Це означає, що кожен рядок зображення ми опрацьовуємо незалежно від інших. Такий підхід вимагає попередньої ректифікації зображення, проте є швидким (відносно інших алгоритмів пошуку

карт глибини), простим для розуміння, та дозволяє отримати непогані результати (рис. 19).



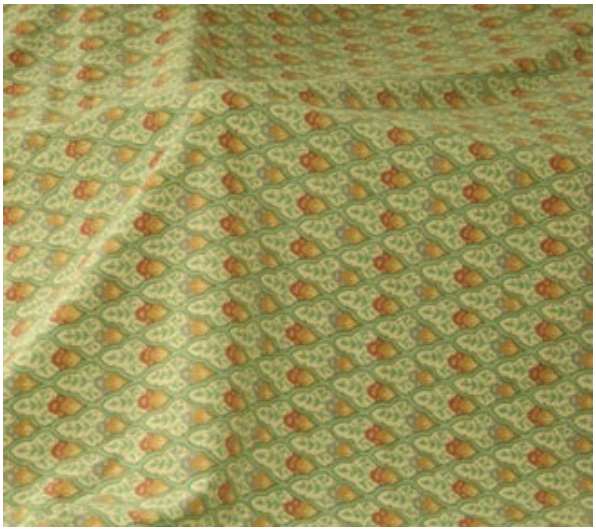


Рис. 18. Ліве та праве зображення сцени

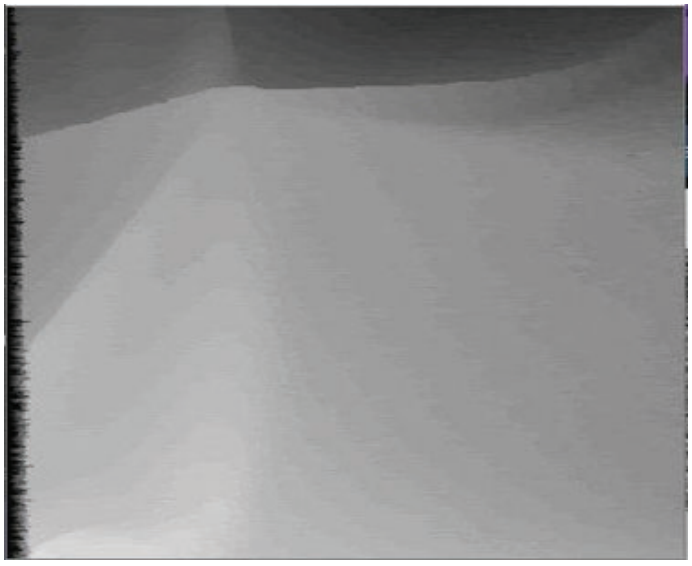


Рис. 19. Карта глибини сцени

4.3.2. ПОШУК КАРТИ ГЛИБИНИ ПРИ ПОВНИХ ДАНИХ

Далі під «зображенням» будемо розуміти деякий рядок вхідного зображення (рис. 20). Нехай n - довжина зображення, а $I = \{1, 2, .., n\}$ - множина координат пікселів. Ліве і праве зображення задамо як функції $\mathcal{L}: I \to R$ та $\mathcal{R}: I \to R$. Тобто $\mathcal{L}(i)$ - інтенсивність i-го пікселя на лівому зображені, а $\mathcal{R}(i)$ - інтенсивність і-го пікселя на правому зображені. Така постановка

природно узагальнюється на випадок кольорових зображень. Також введемо множину зсувів $D = \{0, \dots, D_{max}\}$, де D_{max} - значення максимального зсуву, та підбирається експериментально.

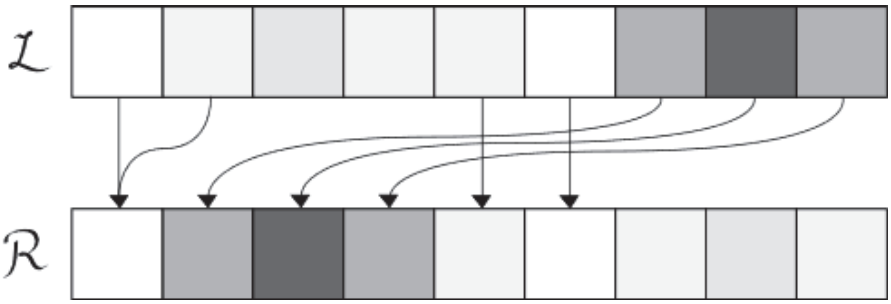


Рис. 20. Пошук відповідних пікселів між правим та лівим зображенням

Для розв'язання задачі для кожного пікселя лівого зображення потрібно знайти відповідний йому піксель на правому зображенні (рис. 20). Називатимемо пару пікселей «відповідною», якщо обидва пікселі містять інформацію про однакову область сцени.

Тобто для кожного пікселя з номером i лівого зображення знайти таке $d_i \in D$, щоб піксель правого зображення з номером $i - d_i$ відповідав йому.

Проте, не будь яка пара пікселей може знаходитися у відповідності, тобто пікселю з номером i на лівому зображені можуть відповідати тільки ті пікселі правого зображення з номером j, для яких $i \leq j$. Переконатися в цьому можна тримаючи перед собою олівець та по черзі закриваючи то праве, то ліве око. Для правого ока олівець буде знаходитися лівіше ніж для лівого.

Тож треба знайти таку послідовність $\overline{d} \in D^n$, яка б мінімізувала наступну штрафну функцію (14).

$$\omega(\overline{d}) = \sum_{i=1}^{n} h(i, d_i) + \sum_{i=1}^{n-1} g(d_i, d_{i+1}), \quad (14)$$

$$h(i, d_i) = | \mathcal{L}(i) - \mathcal{R}(i - d_i) |, \quad (15)$$

$$g(d\_i, d_{i+1}) = \alpha | d\_i - d_{i+1} |.$$

Тут $h(i, d_i)$ відповідає за схожість кольору пікселів, $g(d_i, d_{i+1})$ - за гладкість поля зсувів (15), а α - коефіцієнт згладжування.

4.3.3. ЗВЕДЕННЯ ЗАДАЧІ ДО ПОШУКУ НАЙКОРОТШОГО ШЛЯХУ НА ГРАФІ

Представимо задачу пошуку послідовності $\overline{d}$, що мінімізує штрафну функцію (формула 14), як пошук найкоротшого шляху через орієнтований зважений граф $G =< \mathcal{V}, \mathcal{E} >$ (рис. 21). Множина вершин якого:

$$\mathcal{V} = \sigma(i,d) | i \in I, d \in D \cup S, E. \tag{16}$$

Вершина $\sigma(i,d)$ має вагу $h(i,d),\ i \in I,\ d \in D$.

$$\mathcal{E} = \cup_{d \in D} < S, \sigma(1,d) > \cup \cup_{d \in D} < \sigma(n,d), E > \cup \cup_{\substack{i=1..n-1 \\ d \in D \\ d' \in D}} < \sigma(i,d), \sigma(d+1,d') >. \tag{17}$$

Ваги ребер:

- З S в $\sigma(1,d)$ – $0, d \in D$
- З $\sigma(n,d)$ в E – $0, d \in D$
- З $\sigma(i,d)$ в $\sigma(i+1,d')$ – $g(d,d'), d, d' \in D, i \in I$

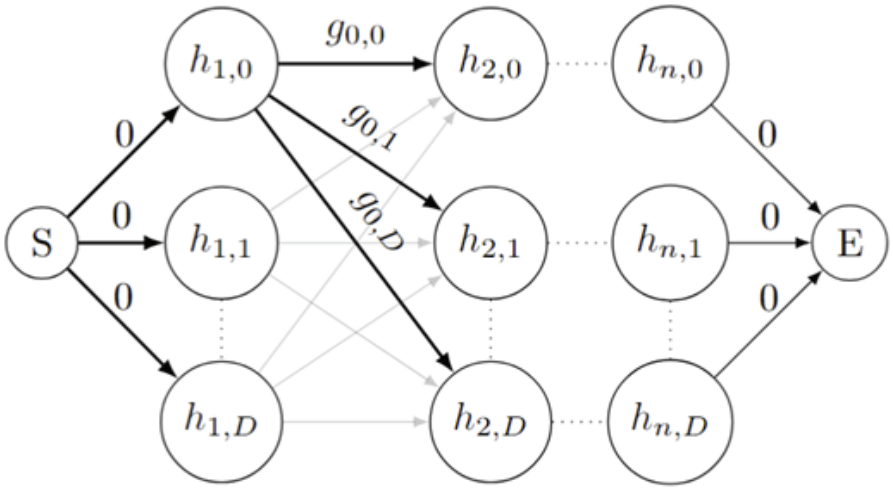


Рис. 21. Структура графу G

Тоді послідовність $\overline{d}$ - послідовність вершин, через які проходить найкоротший шлях з S в E, і буде мінімізувати штрафну функцію $\omega(\overline{d})$ (рис. 21).

Позначимо довжину найкоротшого шляху з вершини S в вершину $\sigma(i,d)$ як $f_i(d)$. Тоді $\forall d \in D$:

$$f_1(d) = h(1, d), \tag{18}$$

$$f_2(d) = \min_{d' \in D}\big(f_1(d) + g(d', d)\big) + h(2, d),$$

...

$$f_i(d) = \min_{d' \in D}\big(f_{i-1}(d) + g(d', d)\big) + h(i, d).$$

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

$$d_n = arg\min_{d' \in D}\big(f_n(d')\big), \tag{19}$$

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

4.3.4. ПОШУК КАРТИ ГЛИБИНИ ПРИ ПОСТУПОВОМУ НАДХОДЖЕННІ ДАНИХ

Метод, описаний в підрозділі 4.3.3, потребує наявності всіх даних до початку обчислень. В ситуації, коли дані надходять повільно, цей метод не є ефективними. Доводиться чекати надходження всіх даних і тільки потім починати обчислення, адже ми не можемо розрахувати ваги деяких вершин.

Замість цього можна проводити деякі розрахунки над частиною даних, яка вже надійшла, цим самим зменшивши час роботи алгоритму після отримання всіх даних.

Нехай на початку нам невідомі жодні пікселі (рис. 22), тож ми не можемо обчислити вагу жодної з вершин графа G.

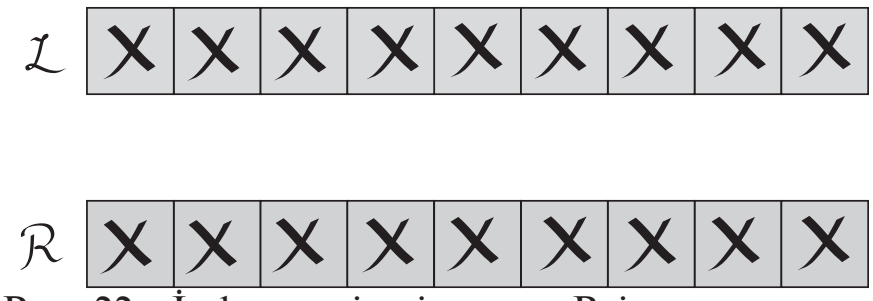


Рис. 22. Інформація відсутня. Всі вершини закриті

Будемо називати вершину "**закритою**" якщо ми не можемо обчислити вагу вершини, а якщо можемо - назвемо її "**відкритою**".

Тож на початку, всі вершини графу G, окрім початкової та кінцевої, будуть закритими (рис. 23).

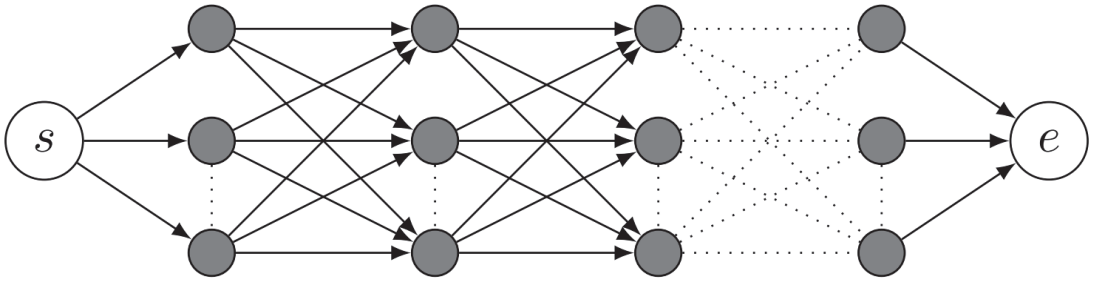


Рис. 23. Граф G

Далі, поступово, з надходженням нових частин інформації, якісь вершини будуть ставати відкритими. Ідея 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 (рис. 24).

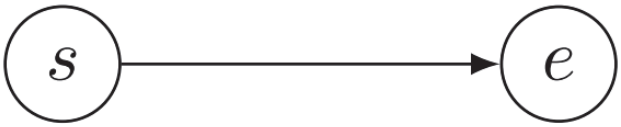


Рис. 24. Граф $G*$ після надходження всіх даних

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 (рис. 25-26).

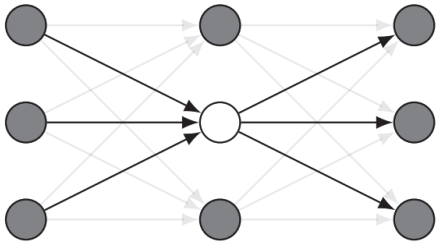


Рис. 25. До вилучення вершини

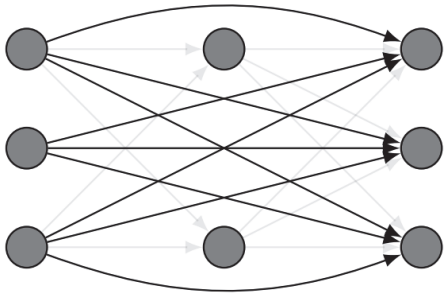


Рис. 26. Після вилучення вершини

Повертаючись до нашого конкретного прикладу: нехай нам поступово, по порядку надходить по одному пікселю кожного зображення. Коли нам відомий лише перший піксель кожного зображення то граф G матиме лише одну відкриту вершину $\sigma(1,0)$ (рис. 27).

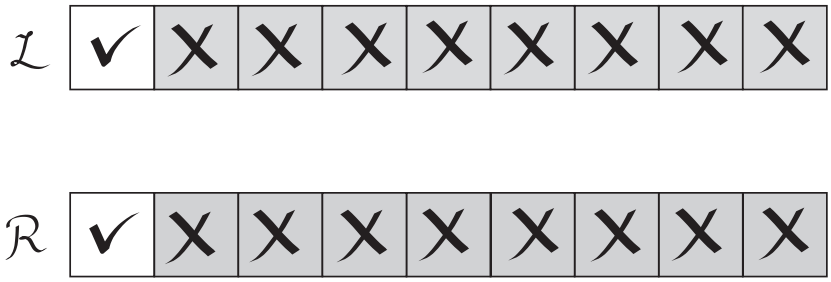


Рис. 27. Відомі перші пікселі обох зображень

Коли нам буде відомо два перші пікселі, то стануть відомі ще й вершини $\sigma(1,1)$ та $\sigma(2,0)$. Таким чином, при надходженні перших k пікселів обох зображень, в графі G будуть відкриті вершини $\sigma(i,di), \text{де} i \in I, i \leq k, adi \in D, 0 \leq di \leq min(k-i, Dmax)$. І тільки коли k буде більше за $Dmax$, всі вершини в першому стовпчику $\sigma(1,d) \forall d \in D$ будуть відкритими, і ми зможемо починати оптимізацію: нехай $G^0 =< \mathcal{V}^0, \mathcal{E}^0 \geq < \mathcal{V}, \mathcal{E} \geq G$. Нехай також $T = \{1,2,\dots, n - D_{max} - 1\}$ - множина ітерацій. А $s: T \times D \to R$. $s^0(d) = 0, \forall d \in D$. Для відновлення послідовності $\overline{d}$ введемо матрицю «попередніх оптимальних вершин» $\hat{\mathcal{P}}$ розмірності $n \times (D_{max} + 1)$.

$$p_{1,d} = 0, \forall d \in D. \tag{20}$$

Як тільки перші $D_{max} + t$, $(t \in T)$ стереопікселей стають нам відомі, шукаємо граф G^t:

- $\forall d \in D$:
$$s^t(d) = \min_{d' \in D}\left(s^{(t-1)}(d') + h(t,d') + g(d',d)\right).$$
$$p_{1+t,d} = \arg\min_{d' \in D}\left(s^t(d)\right)$$
- $\mathcal{V}^t = \mathcal{V}^{t-1} \setminus \{\, \sigma(t,d) \mid d \in D \,\}$.
- Позначимо множину $\bigcup_{d\in D} < S, \sigma(t,d) >$ як $\mathcal{A}$, множину $\bigcup_{\substack{d \in D \\ d' \in D}} < \sigma(t,d), \sigma(t+1,d') >$ як $\mathcal{B}$, а множину $\bigcup_{d\in D} < S, \sigma(t+1,d) >$ як $\mathcal{C}$. Тоді $\mathcal{E}^t = \left(\mathcal{E}^{t-1} \setminus (\mathcal{A} \cup \mathcal{B})\right) \cup \mathcal{C}$. А вага ребра $< S, \sigma(t+1,d) \succ s^t(d)$.
- $G^t =< \mathcal{V}^t, \mathcal{E}^t >$

Кожен новий граф G^t матиме на $(D_{max})^2$ менше ребер, ніж граф G^{t-1}. Таким чином, на момент приходу останніх пікселів, нам треба буде опрацювати лише D_{max} ребер. А якщо б ми спочатку чекали приходу всіх даних, а тільки потім починали обчислення, нам треба було опрацювати $(n-1)(D_{max})^2$ ребер.

Коли ж нам стануть відомі всі n пікселей обох зображень, нам залишиться тільки знайти

$$d_n = arg\min_{d' \in D}\left(s^{(n-D_{max}-1)}(d') + h(n,d')\right), \tag{21}$$

та відновити послідовність $\overline{d}$ через матрицю $\hat{\mathcal{P}}$:

$$d_i = p_{i+1,d_{i+1}}, i = \overline{n-1,1}. \tag{22}$$

4.3.5. НЕУПОРЯДКОВАНЕ НАДХОДЖЕННЯ ДАНИХ. НАЙГІРШИЙ ВИПАДОК

Розглянемо найгірший можливий випадок для запропонованого алгоритму у випадку неупорядкованого надходження даних на тому ж прикладі задачі стереозору.

Найгіршим вважатимемо випадок, при якому на етапі оптимізації t, проміжний граф G^t матиме найбільшу кількість ребер. Для заданої структури дерева, така ситуація складається у випадку, коли у кожному "стовпчику" відкрито по одній вершині (рис. 28).

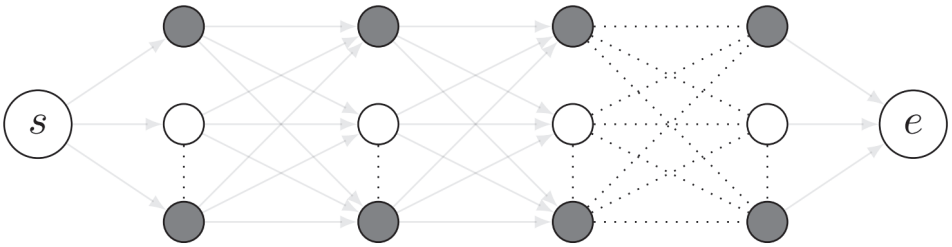


Рис. 28. Відомі перші пікселі обох зображень

Адже тоді, кожна закрита вершина матиме додаткове ребро до кожної закритої вершини у наступних стовпчиках (рис. 29).

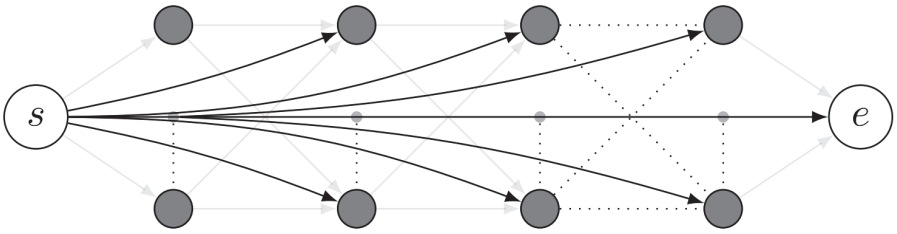


Рис. 29. Відомі перші пікселі обох зображень

У такій ситуації загальна кількість ребер не буде перевищувати $D_{max}^2 n^2$. До того ж будь-яка нова відкрита вершина буде тільки зменшувати загальну кількість ребер у графі.

4.3.6. ПРАКТИЧНИЙ АНАЛІЗ РОБОТИ ЗАПРОПОНОВАНОГО МЕТОДУ НА ПРИКЛАДІ ЗАДАЧІ СТЕРЕОЗОРУ

В цьому розділі порівнюється час отримання карти глибини сцени у результаті роботи алгоритму (наведеного у розділі 4.3.1), що є адаптацією запропонованого методу пошуку розмітки на деревах до задачі одновимірного стереозору τ_{online}, з часом роботи звичайного алгоритму $\tau_{offline}$.

Час порівнюється як з, так і без затримки між надходженням даних, для того, щоб мати уяву про вплив ітеративного процесу оптимізації графу на загальний час роботи, та оцінити доцільність такого ускладнення.

Тестування проводилось на зображеннях з Middlebury Stereo Datasets [10], з максимальним диспаритетом Dmax = 100. Розмір зображення зафіксований як 1000x1000 пікселів (рис. 30).

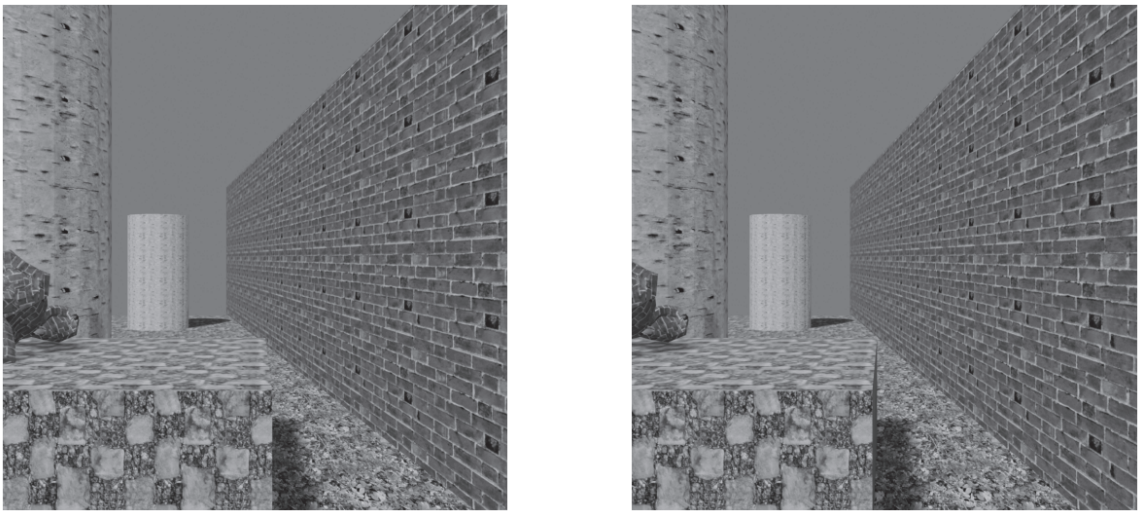


Рис. 30. Тестові зображення (ліве та праве)

На кожній ітерації $t \in \{1, \ldots, 1000\}$ відкриваються пікселі лівого та правого зображення з координатою *t*. Рузультат роботи зображено на рис 31.

Рис. 31. Результати роботи

Таблиця 1 Порівняння часу роботи

Метод	Затримка 0с	Затримка 10^{-5}с
$\tau_{offline}$	5.6с	5.6с
τ_{online}	6.9с	0.04с

Час роботи offline алгоритму після надходження всіх даних $\tau_{offline} = 5.6$ секунд і при відсутній затримці, і при затримці в 10^{-5} секунди. Час роботи online алгоритму після надходження всіх

даних $\tau_{online} = 6.9$ секунд при відсутній затримці, і $\tau_{online} = 0.04$ при затримці в 10^{-5} секунди. Тобто запропонований алгоритм не дає жодних покращень для ситуації, коли всі дані доступні до початку обрахунків. Проте у ситуаціях, коли інформація надходить поступово, запропонований алгоритм потребує значно менше часу для отримання результату, після надходження всіх даних.

4.4. РЕЗУЛЬТАТИ

У цій праці було запропоновано модифікацію методу знаходження оптимальної розмітки на деревах для випадку поступового надходження даних. Основною перевагою запропонованого методу є відсутність необхідності наявності всіх даних для початку роботи; алгоритм дозволяє опрацьовувати вершини дерева у міру надходження інформації про них.

Ефективність і доцільність використання запропонованого алгоритму залежать від комплексу взаємопов'язаних факторів, та продиктовані і природою задачі, і такими факторами, як структура конкретного дерева та характер функцій ваги вершин h. Однак, найважливішим фактором, який визначає доцільність використання запропонованого алгоритму, є співвідношення між часом, необхідним для обробки однієї вершини, і середнім інтервалом часу між надходженням нової інформації про вершини. Якщо алгоритм встигає обробити інформацію про одну вершину до того, як надійде нова інформація про наступну вершину, то такий алгоритм вважається ефективним і його застосування є доцільним. Іншими словами, якщо час обробки однієї вершини менший або дорівнює середньому інтервалу між надходженням нової інформації, то алгоритм працює в режимі реального часу і не утворює затримки в обробці даних.

Навіть якщо запропонований метод не встигає обробити інформацію про кожну вершину до того, як надійде нова інформація (тобто час обробки вершини трохи перевищує інтервал між надходженням даних), його використання все одно може бути виправданим. Це можливо за умови, що загальний час, необхідний для обробки всіх даних за допомогою цього методу, буде меншим, ніж час, який витрачається на обробку тих самих даних за допомогою традиційних методів. Іншими словами, невелика затримка в обробці окремих вершин може бути допустимою, якщо завдяки цьому ми отримуємо прискорення обчислень в цілому.

4.5. ПОДАЛЬШІ ДОСЛІДЖЕННЯ

Запропонований алгоритм має високий потенціал для подальших досліджень та оптимізації. Для більш практичного аналізу запропонованого алгоритму, було б цікаво розглянути його застосування на конкретних задач. Одним із перспективних напрямків є задача бінокулярного стереозору, тобто відновлення глибини сцени за двома зображеннями, отриманими з різних точок зору. Якщо ми сформулюємо задачу бінокулярного стереозору як послідовне знаходження карт диспаритету для кожного рядка зображення, то структура дерева T в нашому алгоритмі буде мати вигляд ланцюжка. Така проста структура дерева може відкрити додаткові можливості для оптимізації алгоритму.

Також можливі подальші модифікації запропонованого методу для конкретних характерів надходження даних. Зокрема, особливий інтерес представляють сценарії, в яких характер надходження даних не хаотичний, а має певну структуру. Наприклад якщо інформація надходить упорядковано, вершина за вершиною. Або дані надходять з деякою симетрією. Також цікавий випадок, коли характер надходження інформації не має певної структури, але відомий наперед. Для кожного такого специфічного випадку можна розробити індивідуальні оптимізації алгоритму, що дозволить підвищити його ефективність та точність.

ВИСНОВКИ

У даному розділі було в загальному вигляді розглянуто задачу пошуку оптимальної розмітки на деревах, та метод зведення такої задачі, до задачі пошуку найкращого шляху на графі спеціального вигляду. Була розглянута модифікація задачі пошуку оптимальної розмітки для випадку поступового надходження даних, та запропонований алгоритм розв'язання таких проблем, для початку роботи якого не обов'язкова наявність повної інформації про задачу.

Також, було наведено приклад практичного застосування запропонованого алгоритму для задачі построкового стереозору у випадку поступового надходження даних. В рамках цього прикладу було проведено експериментальне порівняння часу роботи, необхідного для отримання результату. Ефективність запропонованого алгоритму залежить від характеру даних, а саме,

у випадку наявності повного набору даних до початку обчислень, він не приносить покращень, але в умовах, коли інформація надходить з затримкою, метод є більш ефективним.

Ефективність запропонованого алгоритму сильно залежить від природи задачі, та факторів, таких як структура конкретного дерева T, характер функцій ваги вершин h. Але найважливішим фактором, що визначить чи підійде запропонований в цій роботі алгоритм, у якості ефективного методу вирішення конкретної задачі, є відношення часу надходження частини інформації до часу, необхідного для її обрахунку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Yu Chai and Fei Yang. Semi-global stereo matching algorithm based on minimum spanning tree. pages 2181–2185, 05 2018.

2. Heiko HirschmuËller. Semi-global matching: Motivation, development and applications. pages 173–184, 09 2011.

3. Dorit Hochbaum. An efficient algorithm for image segmentation, markov random fields and related problems. J. ACM, 48:686–701, 07 2001.

4. Heiko HirschmuËller and Daniel Scharstein. Evaluation of cost functions for stereo matching. IEEE Conference on Computer Vision and Pattern Recognition, CVPR'07, 06 2007.

5. Richard Szeliski, Ramin Zabih, Daniel Scharstein, Olga Veksler, Vladimir Kolmogorov, Aseem Agarwala, Marshall Tappen, and Carsten Rother. A comparative study of energy minimization methods for markov random fields. pp. 16–29, 01 1970.

6. Hirschmьller, Heiko. (2011). Semi-Global Matching: Motivation, Development and Applications. 173-184.

7. Facciolo, Gabriele & de Franchis, Carlo & Meinhardt, Enric. (2015). MGM: A Significantly More Global Matching for Stereovision. 10.5244/C.29.90.

8. D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. International Journal of Computer Vision, 47(1/2/3):7-42, April-June 2002.

9. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2003), volume 1, pp. 195-202, Madison, WI, June 2003.

10. Richard Szeliski, Ramin Zabih, Daniel Scharstein, Olga Veksler, Vladimir Kolmogorov, Aseem Agarwala, Marshall Tappen, and Carsten Rother. A comparative study of energy minimization methods for markov random fields, 01 1970.

11. H. Hirschmller and D. Scharstein. Evaluation of cost functions for stereo matching. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007), Minneapolis, MN, June 2007.

12. Middlebury Stereo Datasets. http://vision.middlebury.edu/stereo/data/.