Міністерство освіти і науки україни

Національний авіаційний університет

Факультет кібербезпеки, комп’ютерної та програмної інженерії

Кафедра прикладної математики

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач кафедри

\_\_\_\_\_\_\_\_\_\_\_ П. О. Приставка

“\_\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_2019 р.

**ДИПЛОМНА РОБОТА**

**(пояснювальна записка)**

випускника освітньоГО СТУПЕНЯ

“Бакалавр”

**Тема:** Автоматизована система сегментації цифрових зображень на основі дискретних структур.

**Виконавець:** Ковдря В. Ю.

**Керівник:** К.ф.-м.н., доцент Юрчук І. А.

**Нормоконтролер:** к.ф.-м. н., доцент Кубайчук О. А.

**Київ 2019**

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп’ютерної та програмної інженерії

Кафедра Прикладної математики

Спеціальність 6.040301 “ Прикладна математика ”

ЗАТВЕРДЖУЮ

Завідувач кафедри

\_\_\_\_\_\_\_\_\_ П. О. Приставка

“\_\_\_\_” \_\_\_\_\_\_\_\_\_2019 р.

**ЗАВДАННЯ**

**на виконання дипломної роботи**

*Ковдрі Владислава Юрійовича*

1. **Тема дипломної роботи:** Автоматизована система сегментації цифрових зображень на основі дискретних структур, затверджена наказом ректора від "11'' квітня 2018 р. № 775/ст.
2. **Термін виконання роботи:** з « \_\_\_»\_\_\_\_\_\_\_\_\_\_\_ по «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_
3. **Вихідні дані до роботи:** цифрові зображення довільного розміру, середовище розробки Visual Studio 2017.
4. **Зміст пояснювальної записки:** огляд основних теоретичних положень для виконання дипломного проекту, тобто основних понять топологічного аналізу даних, поняття симпліціального комплексу та поняття сталих гомологій; аналіз існуючих технологій на основі методів топологічного аналізу даних, аналіз структури цифрового зображення; реалізація програмного забезпечення для крастеризації цифрових зображень; проведення прикладного дослідження; аналіз та порівняння отриманих результатів.

**Календарний план-графік**

|  |  |  |  |
| --- | --- | --- | --- |
| №  пор. | Завдання | Термін виконання | Відмітка про виконання |
| 1 | Отримання завдання та огляд літератури за темою дипломної роботи. Аналіз літературних джерел |  | Виконано |
| 2 | Підготовка та написання 1-го та 2-го розділу |  | Виконано |
| 3 | Реалізація програмного забезпечення |  | Виконано |
| 4 | Оформлення дипломної роботи |  | Виконано |
| 5 | Створення презентації |  | Виконано |
| 6 | Передзахист дипломного проекту. |  | Виконано |
| 7 | Захист дипломної роботи. |  |  |

1. **Дата видачі завдання:** «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2019 р.

Керівник дипломної роботи Юрчук І.А.

Завдання прийняв до виконання Ковдря В. Ю.

# РЕФЕРАТ

Пояснювальна записка до дипломної роботи на тему: «Автоматизована система сегментації цифрових зображень на основі дискретних структур»: 54 сторінки, 25 рисунка, 30 використаних джерел.

**Об’єкт дослідження:** обчислювальна система сегментації цифрових зображень.

**Предмет дослідження:** метод ефективної сегментації на сітковому графі.

**Програмне середовище,** що використовувалося в процесі виконання дипломної роботи Microsoft Visual Studio 2017, мова програмування - С#

**Мета роботи:** розробити інформаційну технологію для кластеризації цифрового зображення на основі методу ефективної сегментації на сітковому графі.

**Результатом** виконання дипломної роботи є інформаційна технологія для сегментації цифрових зображень методом ефективної сегментації на сітковому графі реалізована в програмному середовищі Microsoft Visual Studio 2017.

**Галузь застосування:** результати дипломного проекту рекомендуються для використання з ціллю обробки цифрових зображень.

**ЗМІСТ**

[РЕФЕРАТ 4](#_Toc10749350)

[ПОСТАНОВКА ЗАДАЧІ 6](#_Toc10749351)

[РОЗДІЛ 1 8](#_Toc10749352)

[ТЕОРЕТИЧНІ ВІДОМОСТІ 8](#_Toc10749353)

[1.1. Огляд методів сегментації 8](#_Toc10749354)

[1.1.1 Методи, засновані на кластеризації 8](#_Toc10749355)

[1.1.2. Методи, засновані на стисненні 9](#_Toc10749356)

[1.1.3. Методи з використанням гістограми 10](#_Toc10749357)

[1.1.4.Методи, засновані на диференціальних рівняннях з частинними похідними 11](#_Toc10749358)

[1.2. Мой метод 12](#_Toc10749359)

[1.2.1 Предикат для попарного порівняння областей 13](#_Toc10749360)

[1.2.3. Алгоритм і його властивості 15](#_Toc10749361)

[1.3. Обробка зображень 19](#_Toc10749362)

[1.3.1. Згортка 19](#_Toc10749363)

[1.3.2. Згортка зображень 19](#_Toc10749364)

[1.3.3. Обробка країв 20](#_Toc10749365)

[1.3.4. Розмивання Гауса 21](#_Toc10749366)

[РОЗДІЛ 1 24](#_Toc10749367)

[Практична Частина 24](#_Toc10749368)

[Опис програмного забезпечення 24](#_Toc10749369)

[Система неперетинних множин 24](#_Toc10749370)

[Наївна реалізація 25](#_Toc10749371)

[Евристика стиснення шляху 26](#_Toc10749372)

[Евристика об'єднання за рангом 27](#_Toc10749373)

[Об'єднання евристик: стиснення шляху плюс рангова евристика 28](#_Toc10749374)

[Зчитування зображень 29](#_Toc10749375)

[Список використаних джерел 32](#_Toc10749376)

# ПОСТАНОВКА ЗАДАЧІ

Завдання:

1. Вивчити методметод ефективної сегментації на сітковому графі.
2. Розробити алгоритм сегментації цифрового зображення методом ефективної сегментації на сітковому графі
3. Розробити обчислювальну схему для отримання сегментації цифрових зображень.
4. Створити інформаційну технологію на основі розробленого алгоритму.
5. Провести тестування інформаційної технології на реальних даних.
6. Зробити порівняльний аналіз даного методу кластеризації з іншими методами (метод К-середніх).

ВСТУП

**Актуальність**

Задачі сегментації і групування зображень залишаються складними для комп'ютерного зору. З часів гештальтпсихології було відомо, що сприйняття форми(перцептивне сприйняття) відіграє важливу роль у зоровому сприйнятті людини. Широкий спектр задач обчислювального бачення може використовувати сегментовані зображення, якщо такі надійно і ефективно обчислюються. Наприклад, задачі бачення середнього рівня складності, такі як стереобачення та оцінка руху потребують підтримки відповідних операцій. Просторово неоднорідні області можуть бути ідентифіковані за допомогою методів сегментації. Задачі бачення вищого рівня, такі як розпізнавання та індексація зображень, також можуть використовувати результати сегментації для розпізнавання по частинах.

# РОЗДІЛ 1

# ТЕОРЕТИЧНІ ВІДОМОСТІ

В комп'ютерному зорі сегментація зображення - це процес поділу цифрового зображення на кілька сегментів (наборів пікселів, також відомих як супер-пікселі). Мета сегментації-спростити і / або змінити подання зображення на щось більш значуще і більш легке для аналізу. Сегментація зображення зазвичай використовується для пошуку об'єктів і кордонів (ліній, кривих і т. д.) в образах. Точніше, сегментація зображення - це процес присвоєння мітки кожному пікселю зображення таким чином, щоб пікселі з однією і тією ж міткою мали певні характеристики.

Результатом сегментації зображення є набір сегментів, які в сукупності покривають все зображення, або набір контурів, витягнутих з зображення. Кожен з пікселів в області аналогічний за деякими характеристиками або обчислюваними властивостями, таких як колір, інтенсивність або текстура. Суміжні регіони істотно відрізняються один від одного по відношенню до однієї і тієї ж характеристики.

## 1.1. Огляд методів сегментації

### 1.1.1 Методи, засновані на кластеризації

k-середніх[1] — це ітераційний метод, який використовується для того, щоб розділити зображення на K кластерів. Базовий алгоритм наведений нижче:

* 1. Вибрати K центрів кластерів, випадково або на основі деякої евристики
  2. Помістити кожен піксель зображення в кластер, центр якого найближче до цього пікселя
  3. Знову визначити центри кластерів, усереднюючи всі пікселі в кластері
  4. Повторювати кроки 2 і 3 до збіжності (наприклад, коли пікселі будуть залишатися в тому ж кластері)

Тут за відстань зазвичай береться сума квадратів або абсолютних значень різниць між пікселем і центром кластера. Різниця зазвичай базується на кольорі, яскравості, текстурі і місце знаходження пікселя, або на зваженій сумі цих чинників. K може бути вибране вручну, випадково чи евристично.

Цей алгоритм гарантовано сходиться, але він може не привести до оптимального рішення. Якість рішення залежить від початкової множини кластерів і значення K.

### 1.1.2. Методи, засновані на стисненні

В методах заснованих на стисненні стверджується, що оптимальною сегментацією серед усіх можливих є та, котра використовує найменший об'єм даних для кодування результуючого зображення.[2][3] Зв'язок між сегментацією та стисненням пояснюється тим, що сегментація намагається знайти шаблони в зображенні, а будь-який взаємозв'язок в зображенні може бути використаний для його стиснення. Підхід описує кожен сегмент за його текстурою і формою контуру. Кожна складова сегменту моделюється формулою ймовірнісного розподілу, а об’єм даних для кодування обчислюється таким чином:

* 1. Кодування контурів ґрунтується на факті, що області на звичайних зображеннях стараються мати гладкий контур. Це припущення використовується в кодуванні Хаффмана для закодовування диференціального коду ланцюга контурів на зображенні. Так, наскільки контур гладкіший, на стільки ж він займає менше місця.
  2. Текстура кодується за допомогою стиснення з втратами, шляхом схожим до принципу мінімально допустимої довжини опису, але тут об'єм даних для моделі є апроксимованим кількістю вибірок ентропії моделі. Текстура в кожній області моделюється багатовимірним нормальним розподілом, ентропія котрого найбільш схожа з представленням. Цікавою властивістю даної моделі є те, що в межах оціночної ентропії міститься значення ентропії реальних даних текстури. Це тому, що серед усіх наявних функцій розподілу з відомими значеннями середнього арифметичного та кореляційного моменту, нормальний розподіл має найбільшу ентропію. З цього виходить, що реальний об'єм закодованих даних не може бути більшим ніж запропонований алгоритмом.

Для довільної сегментації дана схема обчислює кількість біт, потрібну для кодування зображення з використанням обраної сегментації. Так, серед всіх можливих сегментацій зображення, потрібно знайти сегментацію, яка представляється закодованими даними найменшої довжини. Це можна досягнути з використанням методів ієрархічної кластеризації. Спотворення в стисненні з втратами визначає похибку сегментування і її оптимальне значення може відрізнятись для окремих зображень. Цей параметр може бути оцінено евристично за показниками контрасту текстур на зображенні.

### 1.1.3. Методи з використанням гістограми

Методи з використанням гістограми дуже ефективні порівняно з іншими методами сегментації оскільки вони вимагають тільки один прохід по пікселях. У цьому методі гістограма обчислюється за всіма пікселям зображення і її мінімуми і максимуми використовуються, щоб знайти кластери на зображенні [4]. Колір або яскравість можуть бути використані при порівнянні.

Покращення цього методу — рекурсивно застосовувати його до кластерів на зображенні для того, щоб поділити їх на дрібніші кластери. Процес повторюється з усе меншими і меншими кластерами до тих пір, коли перестануть з'являтися нові кластери [4, 5].

Один недолік цього методу — те, що йому може бути важко знайти значні мінімуми і максимуми на зображенні. У цьому методі класифікації зображень схожі метрика відстаней і зіставлення інтегрованих регіонів.

Підходи, засновані на використанні гістограм можна також швидко адаптувати для кількох кадрів, зберігаючи їх переваги в швидкості за рахунок одного проходу. Гістограма може бути побудована кількома способами, коли розглядаються декілька кадрів. Той самий підхід, який використовується для одного кадру, також може застосовуватися для декількох. Тобто, після об'єднання результатів всі мінімуми і максимуми, які було складно виділити на окремих кадрах, стають помітніші. Гістограма також може бути застосована для кожного пікселя, де інформація використовується для визначення найчастішого кольору для даного положення пікселя. Цей підхід використовує сегментацію, засновану на рухомих об'єктах і нерухомому оточенні, що дає інший вид сегментації, корисний у відео трекінгу.

### 1.1.4.Методи, засновані на диференціальних рівняннях з частинними похідними

Використовуючи диференціальні рівняння з частинними похідними та їх розв'язки за певною числовою схемою можна одержати сегмент зображення.[6] Крива поширення є відомим підходом в цій галузі, з великими можливостями практичного застосування в виокремленні об'єктів, відслідковуванні об'єктів, відновленні просторового зображення, і т.д.. Головною ідеєю є еволюція початкової кривої відповідно найменшого потенціалу функції оцінки, її визначення ж буде відображати завдання котре повинно бути виконано. Щодо зворотної задачі, мінімізації функціоналу оцінки, то вона є важкою і накладає певні обмеження щодо гладкості функції при вирішенні, що може бути виражено як геометричні обмеження на еволюційну криву.

***Параметричні методи***

Техніки відновлення многочлена Ланранжа базуються на визначенні параметра контуру відповідно до деякої стратегії вибірки, і наступної еволюції кожного елемента відповідно до зображення та внутрішніх термів. Дані техніки є швидкими та ефективними, але початкове "чисто параметричне" формулювання (за Касом Віткіним та Терзополусом в 1987 відоме як "змійки") є піддане критиці через обмеження, які полягають в виборі стратегії вибірки, внутрішніх геометричних властивостей кривої, зміни топології (поділ та об’єднання кривих), проблеми адресації при вищих вимірах і т.д.. На сьогодні були розроблені ефективніші "дискретизовані" формулювання для подолання даних обмежень зберігаючи ефективність. В обох випадках оптимізація геометрії виконується методом градієнтного спуску а для обчислення похідних використовується метод скінченних різниць.

***Методи встановлення рівня***

Методи встановлення рівня з самого початку використовувались для відслідковування інтерфейсів, котрі переміщувались Ошером та Сетіаном в 1988 і увійшли в інші галузі обробки зображень наприкінці 90-х. Вони можуть бути використані для ефективного вирішення задач пошуку кривих, поверхонь і т. д. в неявній формі. Головною ідеєю є представлення еволюціонуючого контуру  використовуючи знакову функцію, нульове значення якої відповідає фактичному контуру. Тоді, згідно з рівнянням руху контуру можна легко вивести схожий підхід для неявної поверхні, котра на нульовому рівні буде відображати контур об'єкта. Методи встановлення рівня має багато переваг, він задається в неявній формі, не залежить від параметра, дозволяє просто оцінити геометричні властивості структури, яка еволюціонує, надає можливості зміни топології. Зао, Мерімен та Ошер в 1996 запропонували використовувати дані методи як основу в вирішенні задач оптимізації. З цього випливає, що дані методи можуть бути використані для вирішення багатьох задач в області машинного зору та аналізу медичних зображень [7]. Дослідження різноманітних структур даних для встановлення рівня дало можливість створити дуже ефективні методи вирішення цієї задачі.

## 1.2. Мой метод

Використаємо заснований на графі підхід до сегментації. Нехай G = (V, E) неорієнтований граф з вершинами vi ∈ V - набір елементів для сегментації, і ребра ∈ *E* відповідають парам сусідніх вершин. Кожне ребро має відповідну вагу , яка є невід'ємної мірою відмінності між сусідніми елементами vi і vj. У разі сегментації зображення елементи з V є пікселями, а вага ребра є деякою мірою відмінності між двома пікселями, пов'язаними цим ребром (наприклад, різниця в інтенсивності, кольорі, місцезнаходженні або якому-небудь іншому локальному атрибуті). У розділах 5 і 6 ми розглянемо конкретні набори ребер і вагових функції для сегментації зображень. Однак формулювання тут не залежить від цих визначень.

У графовому підході сегментація S являє собою розбиття V на компоненти таким чином, що кожен компонент (або область) С∈ S відповідає з’єднаним компонентом у графі G’ = (V, E’), де E’ Е. Іншими словами, будь-яка сегментація є підмножиною ребер в Е. Існують різні способи вимірювання якості але вцілому ми хочемо, щоб елементи в компоненті були схожі, а елементи в різних компонентах-несхожі. Це означає, що ребра між двома вершинами в одному компоненті повинні мати відносно низькі ваги, а ребра між вершинами в різних компонентах повинні мати більш високі ваги.

### 1.2.1 Предикат для попарного порівняння областей

У цьому розділі ми визначимо предикат D для оцінки необхідності розділення, двох компонентів в сегментації (двох області зображення). Цей предикат заснований на вимірюванні відмінностей між елементами уздовж кордону двох компонентів, щодо міри відмінностей між сусідніми елементами всередині кожного з двох компонентів. Отриманий предикат порівнює міжкомпонентні відмінності з відмінностями всередині компонента і тим самим адаптується до локальних характеристик даних.

Визначимо *внутрішню різницю* компонента C ⊆ V як найбільшу вагу в MST(C; E), тобто

 (1)

Ідея, що лежить в основі цієї міри полягає в тому, що компонент С залишається зв’язаним доти, доки ребра, що розглядаються мають вагу меншу за Int (C).

Визначимо різницю між двома компонентами C1, C2 ⊆ V як мінімальну вагу з ребер, що з’єднує ці компоненти. Тобто:



Якщо немає ребра, що з'єднує C1 і C2, то нехай Dif(C1; C2) = . Ця міра різниці в принципі може бути проблематичною, оскільки вона відображає тільки найменшу вагу ребра між двома компонентами. На практиці ми виявили, що ця міра працює досить добре, незважаючи на це очевидне обмеження. Більше того, зміна визначення для використання медіанної ваги або якого-небудь іншого квантиля, щоб зробити його більш стійким до аномалій, робить проблему пошуку хорошої сегментації NP-складною, про це йде мова у додатку.

Предикат порівняння областей обчислює чи необхідно розділяти пару компонентів, перевіряючи, чи є різниця між компонентами Dif(C1; C2) великою у порівнянні до внутрішньої різниці принаймні в одному з компонентів, Int(C1) і Int(C2). Порогова функція використовується для керування ступенем, в якій різниця між компонентами повинна бути більше мінімальної внутрішньої різниці. Визначимо предикат попарного порівняння як,



де мінімальна внутрішня різниця, MInt, визначається як,



Порогова функція керує ступенем, в якому відмінність між двома компонентами повинна бути більше за їх внутрішні відмінності, щоб між ними була межа (D повертав істинне значення). Для невеликих компонентів Int (C) не є хорошою оцінкою локальних характеристик даних. В крайньому випадку, коли = 1, Int (C) = 0. Тому ми використовуємо порогову функцію, засновану на розмірі компонента



Де позначає розмір C, а k-деякий постійний параметр. Тобто для невеликих компонентів нам потрібні більш сильні докази кордону. На практиці k встановлює шкалу спостереження, в якій більше k викликає перевагу більших компонентів.

Будь-яка невід'ємна функція одного компонента може бути використана для без зміни алгоритму в розділі 4. Наприклад, метод сегментації може віддати перевагу компонентам певних геометричних форм, визначивши таку функцію , яка повертає великі значення для компонентів, які не відповідають деякій бажаній формі, і маленькі значення для тих, які відповідають. Це змусить алгоритм сегментації агресивно об'єднувати компоненти, які не мають бажаної форми. Такі переваги форми були б настільки слабкими, як переважні компоненти, які не є довгими і тонкими (наприклад, використовуючи відношення периметра до площі) або  сильні, як переважні компоненти, які відповідають конкретній моделі форми. Зверніть увагу, що результатом цього будуть не тільки компоненти бажаної форми, проте серед будь-яких двох сусідніх компонентів один завжди буде бажаної форми

### 1.2.3. Алгоритм і його властивості

У цьому розділі ми опишемо і проаналізуємо алгоритм отримання сегментації з використанням критерію прийняття рішення D, введеного вище. Ми покажемо, що сегментація, отримана за цим алгоритмом, не надто груба і не надто добра, згідно з такими визначеннями:

**Визначення 1** *Сегментація S називається «надто доброю», якшо існує пара областей C1; C2 ∈ S, для яких немає доказів існування границі між ними.*

Щоб визначити, що значить для сегментації бути занадто грубою (мати занадто мало компонентів), ми спочатку введемо поняття *уточнення сегментації*. Для двох заданих сегментацій S і T, утворених з одних початкових даних T називають уточненням S, коли кожен компонент з T міститься (або еквівалентний) в деякому компоненті з S. На додачу, ми кажемо, що T є *правильним уточненням* S коли T S. Зауважимо, що якщо T-правильне уточнення S, то T може бути отримано шляхом розщеплення однієї або декількох областей S. Коли T є правильним уточненням S ми кажемо, що T більш добра за S і S більш груба за T.

**Властивість 1.** Для будь-якого (скінченного) графу G = (V; E) існують деякі сегментації, які не є ні надто грубими, ні надто добрими.

Легко зрозуміти чому ця властивість виконується. Розглянемо сегментацію, де всі елементи знаходяться в одному компоненті. Очевидно, що ця сегментація не надто добра, тому що є тільки один компонент. Якщо сегментація також не надто груба, ми закінчили. В іншому випадку, за визначенням надто грубої сегментації, отримане правильне уточнення не є надто добрим. Обравши одне з цих удосконалень можна повторювати цю процедуру до тих пір, поки ми не отримаємо сегментацію, яка не буде надто грубою. Процедуру можна повторювати N-1 разів, тому що всякий раз, коли ми вибираємо правильне уточнення, ми збільшуємо кількість компонентів у сегментації принаймні на один. Тоді *найкраща* сегментація, яку ми можемо отримати та, кожен елемент якої міститься у своєму компоненті.

Тепер звернемося до алгоритму сегментації, який тісно пов'язаний з алгоритмом Краскала для побудови мінімального остовного дерева графа [6]. Його реалізація може працювати за час O(m log m), де m – кількість ребер у графі.

**Алгоритм 1** *Алгоритм сегментації.*

На вхід подається граф G = (V; E), з n вершинами і m ребрами. Результатом є сегментація V на компоненти S = (C1;…; Cr).

1. Сортуємо E π = (o1, . . . , om) у порядку зростання за вагами.
2. Починаємо з сегментації S0 , де кожна вершина vi знаходиться у своєму компоненті.
3. Повторюємо крок 3 для q = 1; … ; m.
4. Будуємо Sq за заданою Sq-1. Нехай vi і vj позначають вершини, з’єднані ребром q-й послідовності, тобто oq = (vi; vj). Якщо vi і vj знаходяться в системі неперетинних множин Sq-1 і w(oq) мала, в порівнянні з внутрішньою різницею обох цих компонентів, то необхідно об’єднати ці два компоненти. В іншому випадку – нічого не робити. Більш формально, Нехай є компонентом з Sq−1, що містить vi і  - компонент, що містить vj. Якщо  і w(oq) ≤ MInt(, ), тоді при утворенні сегментації Sq компоненти об’єднуються. В іншому випадку компоненти  і переходять до сегментації Sq без змін.
5. Повернути S = Sm.

Тепер покажемо, що сегментація S, отримана алгоритмом 1, підпорядковується глобальним властивостями не бути ні надто доброю, ні надто грубою при використанні предиката порівняння областей D, визначеного в (3). Тобто, хоча алгоритм і приймає тільки жадібні рішення, він породжує сегментацію, що задовольняє цим глобальним властивостям. Крім того, ми покажемо, що будь-які можливі сортування у порядку зростання, на 0 кроці алгоритму, виробляють ту ж сегментацію.

**Лемма 1.** *Якщо на 3 кроці алгоритму, при розгляді ребра oq, компоненти, що розглядаються, не об’єднуються, то один з них обов’язково буде в кінцевій сегментації. Нехай  і  позначають два компоненти, що з’єднані ребром oq = (vi; vj), коли це ребро розглядається алгоритмом. Тоді, або , де - компонентом, що містить vi і  - компонент, що містить vj. укінцевій сегментації.*

**Доведення.** Є два випадки, які призведуть до того, що злиття не відбудеться. Розглянемо випадок . Оскільки ребра розглядаються в неспадаючому порядку за вагою , для всіх . Таким чином, ніяких додаткових злиттів з цим компонентом не відбудеться, тобто, . Аналогічно для .

Зверніть увагу, що у Леммі 1 мається на увазі, що ребро, яке є причиною злиття двох компонентів є саме ребром мінімальної ваги між компонентами. Таким чином, ребра, що викликають злиття, - це саме ті ребра, які будуть обрані алгоритмом Краскала для побудови мінімального остовного дерева (MST) кожного компонента.

**Теорема 1.** Сегментація, отримана за алгоритмом 1, яка використовує предикат порівняння областей D, визначений в (3), не є надто доброю грубою, відповідно до визначення 1.

**Доведення.** За визначенням, для того, щоб S була надто доброю, повинна існувати деяка пара компонентів, для яких D не виконується. Між такою парою компонентів повинно бути хоча б одне ребро, яке розглядалося на кроці 3 і не викликало злиття. Нехай  перше таке ребро в послідовності. У цьому випадку алгоритм вирішить не об’єднувати з , що . За леммою 1 відомо, * або  У будь-якому випадку ми бачимо, що* . Мається на увазі, що D має місце для і , що є протиріччям.

**Теорема 2.** Сегментація, отримана за алгоритмом 1, яка використовує предикат порівняння областей D, визначений в (3), не є надто грубою, відповідно до визначення 2.

**Доведення.** Для того, щоб S була надто грубою, має бути якесь правильне уточнення, T, яке не надто добре. Розглянемо мінімальну вагу ребра e, яке є внутрішнім для компонента C S, але з'єднує різні компоненти A, B T. Відзначимо, що за визначенням уточнення A C і B C.

Оскільки T не надто добра  або . Не втрачаючи загальності, скажемо, що модель вірна. За конструкцією, будь яке ребро, що з’єднує А з іншим підкомпонентом С має масу щонайменше таку ж велику, як w (e), яка, у свою чергу, перевищує максимальну вагу ребра в MST(A; E) тому що . Таким чином, алгоритм, який розглядає ребра в неспадаючому порядку за вагами, повинен врахувати всі ребра з MST(A; E) перед іншими частинами з C. Таким чином, алгоритм повинен сформувати A перед формуванням C, і при формуванні C він повинен об'єднати A з деяким іншим підкомпонентом C. Вага ребра, що викликала це злиття, повинна бути не менше w (e). Однак алгоритм не об'єднав би A в цьому випадку, оскільки . Протиріччя.

**Теорема 3.** *Сегментація, отримана за алгоритмом 1, не залежить від того, який неспадний порядок ребер за вагами використовується.*

***Доведення.*** Будь-який порядок може бути змінений на інший шляхом заміни сусідніх елементів з однаковою вагою. Таким чином, достатньо показати, що заміна двох сусідніх ребер однакової ваги в неспадаючому ваговому порядку не змінює результат, отриманий алгоритмом 1.

Нехай та – сусідні ребра в неспадній послідовності з однаковою вагою. Ясно, що якщо вони з'єднують непересічні пари компонентів, то й порядок, в якому вони розглядаються, не має значення. Єдиний випадок, який нам треба перевірити, - це коли знаходиться між двома компонентами A і B, а - між одним з цих компонентів, скажімо B, та деяким іншим компонентом C.

Тепер ми покажемо, що викличе злиття, коли розглядається після так само як і при розгляді перед . Спершу, припустимо, що викликає злиття при розгляді до . Це означає, що . Якби розглядався перед , або не викликав би злиття, і тривіально все одно викликав би злиття, або викликав би злиття, і в цьому випадку новий компонент B С мав би  Таким чином, ми знаємо , що означає, що все одно викличе злиття. З іншого боку, припустимо, що не викликає злиття, якщо розглядати його до . Це означає, що. Тоді або , і в цьому випадку це все одно було б вірно, якби спочатку розглядався (бо не стосується A), або . У другому випадку, якщо розглядався першим, він не міг викликати злиття, оскільки  і тому . При розгляді після у нас все ще є  і не викликає злиття.

## 1.3. Обробка зображень

### 1.3.1. Згортка

Згортка - це математична операція, що застосовується до двох функцій f і g і породжує третю функцію, яка іноді може розглядатися як модифікована версія однієї з початкових. По суті, це особливий вид інтегрального перетворення.

Операцію згортки можна інтерпретувати як «схожість» однієї функції з відбитою і зрушеною копією іншої. Також згортка може бути описана як вага однієї функції в разі, якщо інша функція, будучи відображеною і зрушеною, є ваговою.

Нехай  — дві функції, інтегровані щодо міри Лебега на просторі . Тоді їх згорткою називається функція , визначена формулою:



Або для дискретного випадку:



### 1.3.2. Згортка зображень

Згортка - це операція обчислення нового значення обраного пікселя, що враховує значення оточуючих його пікселів. Для обчислення значення використовується матриця, яка називається ядром згортки. Зазвичай ядро ​​згортки є квадратною матрицею n \* n, де n - непарне, однак це не обов’язкова вимога. Під час обчислення нового значення обраного пікселя ядро ​​згортки як би «прикладається» своїм центром (саме тут важлива непарність розміру матриці) до даного пікселя. Навколишні пікселі так само накриваються ядром. Далі вираховується сума, де складовими є добутки значень пікселів на значення комірки ядра, що накрила даний піксель. Отримана сума ділиться на суму всіх елементів ядра згортки. Частка як раз і є новим значенням обраного пікселя. Якщо застосувати згортку до кожного пікселя зображення, то в результаті вийде якийсь ефект, що залежить від обраного ядра згортки.

Загальним виразом згортки є:



де — відфільтроване зображення,  — вхідне зображення, —ядро згортки. Розглядається кожен елемент ядра фільтра by .

Приклад згортки для пікселя з координатами [2;2] (без нормування)



### 1.3.3. Обробка країв

При використанні операції згорки на краях зображення, необхідні значення пікселів за його межами. Звідси з’являються різні варіанти обробки країв:

* **Розширення**

Найблищі до країв пікселі продовжуються настільки далеко, наскільки це необхідно для згортки. Кутові пікселі утворюють квадрати, а інші пікселі продовжуються у лінію.



* **Обгортання**

Зображення загорнуто саме в себе і значення беруться з протилежного краю або кута.



* Віддзеркалення по краю

При зверненні до пікселів за межами оригінального зображення беруться їх дзеркальні відображення відносно границь або кутів.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| e | d | d | e | f |
| b | a | a | b | c |
| b | a | a | b | c |
| e | d | d | e | f |
| h | g | g | h | i |

* Зріз ядра

Елементи, що виходять за межі зображення не враховуються. Коефіцієнт нормування коректується для компенсації.

### 1.3.4. Розмивання Гауса

Розмивання Гауса — це метод фільтрації зображення за допомогою функції Гауса, який призводить до розмивання зображення. Даний ефект широко використовується в графічних програмах, як правило, для зменшення зашумленості зображенні та зниження деталізації. Візуальний ефект цієї фільтрації розмивання аналогічний погляду на зображення крізь напівпрозорий екран.

**Принцип роботи**

Розмивання Гауса це тип фільтру розмивання зображення, що використовує функцію Гауса для розрахунку трансформації кожного пікселя у зображенні. Рівняння функції Гауса в одному вимірі:



Для двовимірного випадку:



де x — це відстань від початку координат в осі абсцис, y — це відстань від початку координат у осі ординат, а σ — стандартне відхилення розподілу Гауса.

Коли метод застосовується у двох вимірах, отримується поверхня, контури якої є концентричні кола розподілу Гауса з центральної точки (Рис 1). Значення з цього розподілу використовуються для створення матриці згортки(початок координат у центрі матриці). Для кожного нового значення пікселя визначається середнє зважене в околі пікселя. Значення поточного оригінального пікселя має більшу вагу (найвище значення розподілу Гауса), а сусідні пікселі отримують все меншу вагу в залежності від того наскільки далеко вони знаходяться від поточного оригінального пікселя. Це надає ефект розмитості, яка зберігає кордони та краї краще, ніж інші, аналогічні фільтри розмиття.



Рис 1. концентричні кола розподілу Гауса

Теоретично, в кожній точці зображення буде відмінним від нуля результат функції Гауса, це означає, що для розрахунку для кожного пікселя необхідно брати значення усіх пікселів у зображенні. На практиці, при обчисленні дискретного спрощення функції Гауса, вага пікселів на відстані більш ніж 3σ достатньо мала, щоб мати якийсь вплив на середнє зважене значення і вважається нулем. Значення пікселів, що розташовані за межами цього діапазону можуть бути проігноровані. Як правило, програмі обробки зображень необхідно тільки розрахувати матрицю з розмірами , щоб забезпечити результат, який є досить близьким до отриманого від усього розподілу Гауса.

Застосування розмивання Гауса призводить до розмивання на зображення і має ті ж наслідки, що застосування єдиного розмивання Гауса, радіус якого рівний квадратному кореню з суми квадратів радіусу розмиття, що застосовується. Наприклад, застосовуючи послідовного розмивання Гауса з радіусом 6 і 8 дає ті ж результати, як застосування єдиного розмивання радіусом 10, оскільки . Згідно з цим відношенням, час обробки не може бути зменшений шляхом імітації розмивання Гауса з послідовними процесом.

# РОЗДІЛ 2

# Практична Частина

## Опис програмного забезпечення

### Система неперетинних множин

Система неперетинних множин (англ. *disjoint-set-union* або **DSU**, також використовують назви англ. *union–find data structure*, англ. *merge–find set*) — структура даних, яка дозволяє відстежувати множину елементів, розбиту на неперетинні підмножини. При цьому кожній підмножині призначається її представник — елемент цієї підмножини.

Спочатку є кілька елементів, кожен з яких знаходиться в окремій (своїй власній) множині. За одну операцію можна об'єднати дві будь-які множини, а також можна дізнатися, в якій множині зараз знаходиться зазначений елемент. У класичному варіанті вводиться ще одна операція — створення нового елемента, який поміщається в окрему множину.

Таким чином, базовий інтерфейс даної структури даних складається всього з трьох операцій:

* *MakeSet* — додання нового елемента; розміщення його в нову множину, що складається з одного нього;
* *Union* — об'єднання двох зазначених множин;
* *Find* — повернення значення, в якій множині знаходиться зазначений елемент. Насправді при цьому повертається один з елементів множини званий представником (англ. *representative*) або лідером (*leader*). Цей представник вибирається в кожній множині самою структурою даних (і може змінюватися з плином часу). Наприклад, якщо виклик для якихось двох елементів повернув одне і те ж значення, то це означає, що ці елементи знаходяться в одній і тій же множині, а в іншому випадку — в різних множинах.

**Побудова ефективної структури даних**

Множина елементів буде зберігатися у вигляді дерев: одне дерево відповідає одній множині. Корінь дерева — це представник (лідер) множини.

При реалізації це означає, що ми заводимо масив, в якому для кожного елемента зберігаємо посилання на його предка в дереві. Для коренів дерев будемо вважати, що їх предок — вони самі (тобто посилання зациклюється в цьому місці).

#### Наївна реалізація

Наведеної вище інформації вже достатньо, щоб написати першу реалізацію системи неперетинних множин. Вона буде досить неефективною, але потім буде покращена за допомогою двох прийомів, що дадуть змогу отримати в результаті майже константний час роботи.

Отже, вся інформація про множини елементів зберігається за допомогою масиву **parrent**.

***Опис операцій***

* Щоб створити новий елемент (операція make\_set(v)), ми просто створюємо дерево з коренем у вершині, зазначаючи, що його предок — це він сам.
* Щоб об'єднати дві множини (операція union(a, b)), ми спочатку знайдемо лідерів першої і другої множини. Якщо лідери збіглися, то нічого не робимо — це означає, що множини і так вже були об'єднані. В іншому випадку можна просто вказати, що предок першої вершини дорівнює другій (або навпаки) — тим самим приєднавши одне дерево до іншого.
* Реалізація операції пошуку лідера (find(v)) проста: ми піднімаємося по предкам від вершини, поки не дійдемо до кореня. Цю операцію зручніше реалізувати рекурсивно.

void make\_set (int v) {

parent[v] = v;

}

int find (int v) {

**if** (v == parent[v])

**return** v;

**return** find\_set (parent[v]);

}

void union (int a, int b) {

a = find\_set (a);

b = find\_set (b);

**if** (a != b)

parent[b] = a;

}

Утім, така реалізація системи неперетинних множин дуже неефективна. Легко побудувати приклад, коли після кількох об'єднань множин вийде ситуація, що множина — це дерево, звиродніле в довгий ланцюжок.

Для уникнення подібних ситуацій вводяться евристики: «Евристика стиснення шляху» та «Евристика об'єднання за рангом».

#### Евристика стиснення шляху

Ця евристика призначена для прискорення роботи операції ***Find***.

Вона полягає в тому, що коли після виклику ми знайдемо шуканого лідера множини, то запам'ятаємо, що у вершині *v* і всіх пройдених по шляху вершин — саме цей лідер. Найпростіше це зробити, перенаправивши їх parrent-покажчик на цю вершину.

Таким чином, ідеологія масиву parrent дещо змінюється: тепер це стислий масив предків, тобто для кожної вершини там може зберігатися не безпосередній предок, а предок предка, предок предка предка, і т. д.

З іншого боку, зрозуміло, що не можна зробити, щоб ці покажчики parrent завжди вказували на лідера: інакше при виконанні операції довелося б оновлювати лідерів у елементів.

Таким чином, до масиву слід підходити саме як до масиву предків, можливо, частково стиснутого.

Нова реалізація операції має такий вигляд:

int find (int v) {

**if** (v == parent[v])

**return** v;

**return** parent[v] = find\_set (parent[v]);

}

Така проста реалізація робить все, що задумувалося: спочатку шляхом рекурсивних викликів знаходиться лідера множини, а потім, в процесі розкрутки стека, цей лідер присвоюється parrent посиланнями для всіх пройдених елементів.

Реалізувати цю операцію можна і не рекурсивно, але тоді доведеться здійснювати два проходи по дереву: перший знайде шуканого лідера, другий — проставить його всім вершин шляху. Втім, на практиці нерекурсивна реалізація не дає істотного виграшу.

#### Евристика об'єднання за рангом

Розглянемо іншу евристику, яка сама по собі здатна прискорити час роботи алгоритму, а в поєднанні з евристикою стиснення шляхів і зовсім здатна досягти практично константного часу роботи на один запит в середньому.

Ця евристика полягає в невеликій зміні роботи операції Union: якщо в наївній реалізації те, яке дерево буде приєднано до якого, визначається випадково, то тепер ми будемо це робити на основі рангів.

Є два варіанти рангової евристики: в одному варіанті рангом дерева називається кількість вершин в ньому, в іншому — глибина дерева (точніше, верхня межа на глибину дерева, оскільки при одночасному застосуванні евристики стиснення шляхів реальна глибина дерева може зменшуватися).

В обох варіантах суть евристики одна й та ж: при виконанні Union будемо приєднувати дерево з меншим рангом до дерева з більшим рангом.

Реалізація **рангової евристики на основі глибини дерев**:

void make\_set (int v) {

parent[v] = v;

rank[v] = 0;

}

void union(int a, int b) {

a = find\_set (a);

b = find\_set (b);

**if** (a != b) {

**if** (rank[a] < rank[b])

swap (a, b);

parent[b] = a;

**if** (rank[a] == rank[b])

++rank[a];

}

}

#### Об'єднання евристик: стиснення шляху плюс рангова евристика

При спільному застосуванні евристик: стиснення шляху та об'єднання за рангом — час роботи на один запит виходить O(A(n)) в середньому, де A(n)— обернена функція Акермана, яка зростає дуже повільно, настільки повільно, що для всіх розумних обмежень вона не перевершує 4 (приблизно для n<=10600).

***підсумкова реалізацію системи неперетинних множин:***

void make\_set (int v) {

parent[v] = v;

rank[v] = 0;

}

int find (int v) {

**if** (v == parent[v])

**return** v;

**return** parent[v] = find\_set (parent[v]);

}

void union (int a, int b) {

a = find\_set (a);

b = find\_set (b);

**if** (a != b) {

**if** (rank[a] < rank[b])

swap (a, b);

parent[b] = a;

**if** (rank[a] == rank[b])

++rank[a];

}

}

### Зчитування зображень

Перш ніж реалізовувати алгоритм сегментації на мові програмування високого рівня c#, необхідно було дослідити різні методи зчитування зображення, тобто його перетворення до матриці структур Color.

В усіх методах використовується класичний клас Bitmap (System.Drawing.Bitmap). Даний клас зручний тим, що приховує від нас деталі кодування растрових форматів, підтримуючи всі поширені формати, типу BMP, GIF, JPEG, PNG.

При тестах послідовно виконуються 100 зчитувань зображення розмірами 1000\*1000 пікселів. Процесор – Intel Core i5-8600k. При інтервальному оцінюванні, ймовірність похибки першого роду

**«Стандартний» метод**

Суть методу полягає у використанні методу Bitmap.GetPixel(x, y). Наведемо повністю код методу, який конвертує вміст Bitmap в тривимірний масив. При цьому перша розмірність - це колірна компонента (від 0 до 2), друга - позиція y, третя - позиція x.

public static double[,,] BitmapToDoubleRgbNaive(Bitmap bmp)

{

int width = bmp.Width,

height = bmp.Height;

double [,,] res = new double[3, height, width];

for (int y = 0; y < height; ++y)

{

for (int x = 0; x < width; ++x)

{

Color color = bmp.GetPixel(x, y);

res[0, y, x] = color.R;

res[1, y, x] = color.G;

res[2, y, x] = color.B;

}

}

return res;

}

Результати

**Загальний час тесту: 72,4922 секунди.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Нижнє значення** | **Значення** | **Верхнє значення** | **Sigma** |
| **Мат. Сподівання** | 0.7247 | 0.7262 | 0.7278 | 0.0008 |
| **Середньоквадратичне відхилення** | 0.0070 | 0.0081 | 0.0092 | 0.0005 |

**Пряма робота з даними Bitmap за допомогою вказівників**

Клас Bitmap надає більш швидкий спосіб звернутися до своїх даних. Для цього нам необхідно скористатися посиланнями, наданими класом BitmapData та арифметикою вказівників:

public unsafe static double[,,] BitmapToDoubleRgb(Bitmap bmp)

{

int width = bmp.Width,

height = bmp.Height;

double[,,] res = new double[3, height, width];

BitmapData bd = bmp.LockBits(new Rectangle(0, 0, width, height), ImageLockMode.ReadOnly, PixelFormat.Format24bppRgb);

try

{

byte\* curpos;

fixed (double\* \_res = res)

{

double\* \_r = \_res,

\_g = \_res + width \* height,

\_b = \_res + 2 \* width \* height;

for (int h = 0; h < height; h++)

{

curpos = ((byte\*)bd.Scan0) + h \* bd.Stride;

for (int w = 0; w < width; w++)

{

\*\_b = \*(curpos++); ++\_b;

\*\_g = \*(curpos++); ++\_g;

\*\_r = \*(curpos++); ++\_r;

}

}

}

}

finally

{

bmp.UnlockBits(bd);

}

return res;

}

Результати

**Загальний час тесту:** 1,7823 **секунди.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Нижнє значення** | **Значення** | **Верхнє значення** | **Sigma** |
| **Мат. Сподівання** | 0.0168 | 0.0178 | 0.0189 | 0.0006 |
| **Середньоквадратичне відхилення** | 0.0048 | 0.0056 | 0.0063 | 0.0004 |

**Незважаючи на те, що перший метод є дуже простим у застосуванні, використовувати його як модуль у сегментації зображень немає сенсу, через його низьку ефективність порівняно з другим методом.**

# Список використаних джерел

* 1. Kanungo, T.; [*Mount, D. M.*](https://en.wikipedia.org/wiki/David_Mount); [*Netanyahu, N. S.*](https://en.wikipedia.org/wiki/Nathan_Netanyahu); Piatko, C. D.; Silverman, R.; Wu, A. Y. (2002). [*"An efficient k-means clustering algorithm: Analysis and implementation"*](http://www.cs.umd.edu/~mount/Papers/pami02.pdf) *(PDF)*. IEEE Transactions on Pattern Analysis and Machine Intelligence. **24** (7): 881–892.
  2. Mobahi, Hossein; Rao, Shankar R.; Yang, Allen Y.; Sastry, Shankar S.; Ma, Yi (2011-04-08). [Segmentation of Natural Images by Texture and Boundary Compression](http://link.springer.com/article/10.1007/s11263-011-0444-0). *International Journal of Computer Vision* (en) **95** (1). с. 86–98. [ISSN](https://uk.wikipedia.org/wiki/ISSN) [0920-5691](https://www.worldcat.org/issn/0920-5691). [doi](https://uk.wikipedia.org/wiki/%D0%A6%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B8%D0%B9_%D1%96%D0%B4%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D1%96%D0%BA%D0%B0%D1%82%D0%BE%D1%80_%D0%BE%D0%B1%27%D1%94%D0%BA%D1%82%D0%B0):[10.1007/s11263-011-0444-0](http://dx.doi.org/10.1007%2Fs11263-011-0444-0)
  3. Shankar Rao, Hossein Mobahi, Allen Yang, Shankar Sastry and Yi Ma [Natural Image Segmentation with Adaptive Texture and Boundary Encoding](http://perception.csl.illinois.edu/coding/papers/RaoS2009-ACCV.pdf), Proceedings of the Asian Conference on Computer Vision (ACCV) 2009, H. Zha, R.-i. Taniguchi, and S. Maybank (Eds.), Part I, LNCS 5994, pp. 135--146, Springer.
  4. Linda G. Shapiro and George C. Stockman (2001): «Computer Vision», pp 279–325, New Jersey, Prentice-Hall, [ISBN 0-13-030796-3](https://uk.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B5%D1%86%D1%96%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0:%D0%94%D0%B6%D0%B5%D1%80%D0%B5%D0%BB%D0%B0_%D0%BA%D0%BD%D0%B8%D0%B3/0130307963)
  5. Ron Ohlander, Keith Price, and D. Raj Reddy (1978): «Picture Segmentation Using a Recursive Region Splitting Method», *Computer Graphics and Image Processing*, volume 8, pp 313–333
  6. Caselles, V.; Kimmel, R.; Sapiro, G. (1997). [Geodesic active contours](http://www.cs.technion.ac.il/~ron/PAPERS/CasKimSap_IJCV1997.pdf) (PDF). *International Journal of Computer Vision* **22** (1): 61–79.
  7. S. Osher and N. Paragios. [Geometric Level Set Methods in Imaging Vision and Graphics](http://www.mas.ecp.fr/vision/Personnel/nikos/osher-paragios/), Springer Verlag, [ISBN 0-387-95488-0](https://uk.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B5%D1%86%D1%96%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0:%D0%94%D0%B6%D0%B5%D1%80%D0%B5%D0%BB%D0%B0_%D0%BA%D0%BD%D0%B8%D0%B3/0387954880), 2003.