УДК 004.8

В.С. Ленько, Ю.М. Щербина,
Львівський національний університет імені Івана Франка

ЗАСТОСУВАННЯ МЕТОДІВ ШТУЧНОГО ІНТЕЛЕКТУ ДО СЕГМЕНТАЦІЇ ГРАФІЧНОГО ОБРАЗУ

Розглянуто ефективний графовий алгоритм сегментації графічних образів (ЕГАСЗ). Досліджено аспекти ефективної реалізації алгоритму, зокрема використання структури даних "об'єднання множин, які не перетинаються" з евристиками "скорочення шляху" та "об'єднання за рангом". Оцінено придатність алгоритму до використання в автоматизованих системах. Проаналізовано застосовність мір кластерної придатності для оцінювання якості сегментації, одержаної застосуванням ЕГАСЗ до графічного образу.

Ключові слова: сегментація графічного образу, ефективний графовий алгоритм сегментації зображення, об'єднання множин, які не перетинаються, скорочення шляху, об'єднання за рангом, оцінка придатності кластеризації.

An efficient graph-based image segmentation algorithm (EGBIS) is considered. The aspects of an efficient algorithm implementation, in particular the use of "Disjoint sets union" data structure with its heuristics "path compression" and "union be rank", are investigated. A suitability of the algorithm for the use in automated systems is reviewed. An applicability of cluster validity measures for image segmentation quality assessment is analyzed.

Key words: image segmentation, efficient graph-based image segmentation, disjoint sets union, path compression, union by rank, clustering validity assessment.

Вступ. Формулювання проблеми

Сегментація графічного образу – це процес поділу зображення на сегменти (області, кластери), за якого будь-які два елементи з одного сегмента мають схожі візуальні характеристики (колір, яскравість, текстуру), а будь-які два елементи з різних сегментів – значно відрізняються за ними. Метою сегментації є отримання компактного подання графічних даних, що дасть змогу спростити їх подальший аналіз.

З часів наукового відкриття Гештальта в психології відомим є той факт, що перцептивне групування має важливе значення у зоровому сприйнятті об'єктів людиною [4], тому пошук самоподібних областей на графічному образі насправді є спробою змоделювати механізм цього сприйняття.

Сьогодні під час групування та сегментування зображення залишаються серйозним викликом при реалізації систем комп'ютерного зору, оскільки ці системи вимагають швидкої та якісної сегментації. Наприклад, задача середнього рівня, така як визначення траєкторії руху об'єкта, вимагає певну "область підтримки" для виконання відповідних розрахунків. Дослідження шляхів підвищення швидкодії та покращення якості сегментації зображення дозволить зменшити час прийняття рішення інтелектуальною системою.

Аналіз останніх публікацій

Для вирішення проблеми отримання сегментації відома достатня кількість алгоритмів. Останніми роками значний прогрес спостерігається у розвитку графових алгоритмів сегментації. Так, у 2004 р. в роботі [4] було представлено "Ефективний графовий алгоритм сегментації зображення". А у 2009 р. в роботі [8] з'явився "Ефективний паралельний графовий алгоритм сегментації зображення", який за швидкістю виконання сегментації перевершив попередника. Проте, незважаючи на відмінності у деталях, принцип сегментації зображення у цих двох алгоритмах однаковий: сегментація отримується в результаті побудови мінімального каркаса графа, в якому вершини відповідають пікселям зображення, а ваги ребер відображають міру відмінності між двома суміжними вершинами.

Формулювання цілей статті

Нехай задано довільний графічний образ. Необхідно знайти сегментацію цього образу. Очевидно, що для цього необхідно використати один з алгоритмів сегментації зображення. Наразі здебільшого проблему сегментації найшвидше та найякісніше вирішує людина, тому логічним є вибір алгоритму з області штучного інтелекту.

Процес сегментації зображення передбачає виконання великої кількості операцій над значною кількістю об'єктів, тому доцільно дослідити алгоритм на предмет використання оптимальних структур даних для збереження та пошуку окремих елементів. Забезпечення швидкодії алгоритму уможливить його використання в інтелектуальних системах, що працюють у режимі реального часу.

Сегментація графічного образу виконується з метою спрощення подальшого аналізу графічних даних. Проте якщо результат сегментації є неякісним, тобто перцептивно важливі регіони не знайдено, то подальший аналіз даних навпаки може ускладнитись. Якісний результат сегментації дозволяє зменшити час прийняття рішення (спростити аналіз), тому дослідження аспекту забезпечення якості є не менш важливим, ніж оптимізація швидкодії алгоритму.

Для дослідження було обрано ефективний графовий алгоритм сегментації зображення. Оптимізація алгоритму, на предмет підвищення швидкодії та покращення якості результату сегментації, можлива лише після ґрунтовного дослідження його структури.

Основний матеріал

ЕГАСЗ належить до класу алгоритмів, які виконують сегментацію зображення під час побудови мінімального каркаса графа. Для виявлення межі між компонентами використовується предикат порівняння пари областей [4], а для побудови мінімального каркаса графа – алгоритм Краскала. З метою контролю властивостей кінцевої сегментації в алгоритм подаються значення таких параметрів:

- графічний образ, що являє собою матрицю розміром $m \times n$, елементами якої є RGB значення кольорів відповідних пікселів;
- s – радіус гауссового згладжування, що застосовується до графічного образу перед початком процесу сегментації;
- k – параметр, що використовується для обчислення значення динамічного порогу в предикаті порівняння пари областей;
- min – параметр, що відповідає за мінімальну потужність результуючих сегментів.

Задля кращого контролю над процесом сегментації, в алгоритм варто додатково передавати значення таких параметрів:

- спосіб побудови множини ребер графа;
- метрика для обчислення відмінності між вершинами графа.

Після того як значення усіх параметрів задано, розпочинається процес сегментації графічного образу. Згідно з алгоритмом, у вказаній послідовності виконуються такі дії:

1) до графічного образу застосовується гауссове згладжування з радіусом s ;

2) будується множина вершин графа;

3) будується множина зважених ребер графа у заданий спосіб;

4) виконується побудова мінімального каркаса графа за допомогою алгоритму Краскала;

5) виконується усунення сегментів, потужність яких менша за значення параметра min;

6) якщо сегментацію необхідно відобразити, тоді переважно виконують розфарбування сегментів різними кольорами;

Розглянемо детальніше кожен з вищеописаних кроків, з погляду оптимальної програмної реалізації. Першим кроком в ЕГАСЗ є застосування гауссового згладжування до графічного образу з метою усунення шумів та зменшення деталізації. Радіусом гауссового згладжування є параметр s . Ця величина повинна бути вибрана так, щоб після виконання процедури згладжування межі об'єктів збереглися, тому значення для s переважно вибирають з проміжку [0;1]. Крім того, потрібно

пам’ятати, що чим більший радіус, тим довше процедура виконується. Логічно припустити, що для зображення, яке являє собою двовимірний об’єкт, варто використати двовимірне гауссове згладжування. Проте, як показують досліди [3], ефективнішим є виконання двох проходжень (по вертикалі та горизонталі) одновимірним гауссовим згладжуванням. Загалом, важливість цієї процедури полягає у тому, що вона дає змогу зменшити кількість дрібних сегментів та сприяє знаходженню самоподібних областей з неоднорідною структурою і, отже, покращує якість сегментації.

Наступним кроком є створення множини вершин графа. Для зображення, що має розмір $m \times n$ пікселів, кількість вершин дорівнює числу mn. Множина вершин є необхідною лише для створення ребер та побудови початкової сегментації в алгоритмі Краскала, тому для підвищення ефективності роботи алгоритму, замість того, щоб зберігати mn об’єктів в оперативну пам’ять, варто використовувати лише унікальний індекс вершини, який є цілим числом з проміжку $[0; mn-1]$. Так можна зекономити оперативну пам’ять, час на її виділення та запис об’єктів в неї.

Після того, як множину вершин створено, необхідно побудувати множину ребер для того, щоб сполучити вершини графа. Множину ребер можна створювати за будь-яким принципом, проте варто керуватися таким міркуванням, що немає сенсу сполучати всі mn вершин між собою. Оптимальною буде кількість ребер $O(mn)$. Одним з способів створення множини ребер є сполучення сусідніх пікселів у гратці зображення (матриці пікселів) між собою. Прикладами такого способу є сполучення вершини з чотирма (4-connected) та вісьмома (8-connected) найближчими сусідами (рис. 1).

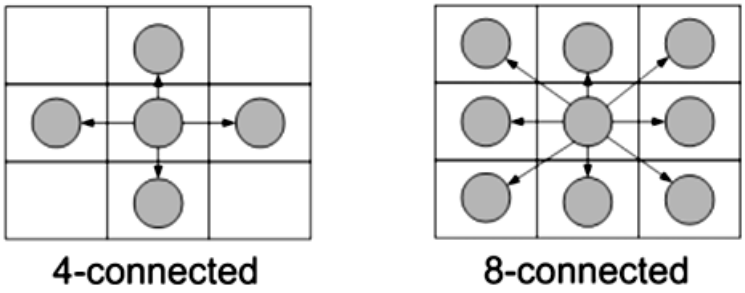

Рис. 1. Способи з’єднання вершин графа за допомогою ребер

Що стосується оптимального збереження ребер графа в оперативній пам’яті, то об’єкт “ребро” повинен містити такі поля: індекс першої вершини, індекс другої вершини, значення ваги ребра. Вибираючи спосіб з’єднання вершин, є можливість наперед дізнатися кількість ребер необхідних для цього. Наприклад, для випадку 4-connected кількість ребер дорівнює числу $2mn-(m+n)$, для 8-connected – числу $4mn-3(m+n)+2$. Отже, знаючи кількість об’єктів та об’єм пам’яті, яку займає об’єкт “ребро”, можна завчасно виділити необхідний об’єм оперативної пам’яті для списку ребер.

Кожне ребро має мати вагу. В ЕГАСЗ вага ребра повинна відображати міру відмінності між двома вершинами. Оскільки довільна вершина графа однозначно відповідає певному пікселю на зображенні, то вага ребра є певною мірою відмінності між двома суміжними пікселями. Кожен піксель має такі локальні властивості, як колір та позиція в матриці зображення (x, y). Очевидно, що обчислення міри відмінності повинне базуватися на порівнянні цих локальних характеристик. У цьому дослідженні за міру відмінності було вибрано різницю кольорів суміжних пікселів, оскільки позиція пікселів була врахована під час побудови множини ребер. Тепер постає питання: як обчислити різницю кольорів?

У машинній реалізації колір переважно представлений значенням з простору RGB, в якому будь-який колір означається комбінацією інтенсивностей червоного, зеленого та синього кольорів. У роботі [4] було запропоновано сегментувати зображення так:

1) для формування кінцевої сегментації необхідно виконати 3 проміжні сегментації, в яких ваги ребер представлені різницями інтенсивностей відповідно червоної, зеленої та синьої компоненти кольорів суміжних вершин;

2) якщо піксель не менше двох разів попадав у той самий сегмент, то він відбирався до кінцевого сегменту.

Проте вже в праці [8] було показано недоліки такого підходу. По-перше, формування кінцевої сегментації вимагає виконання трьох проміжних сегментацій, що призводить до зниження швидкодії алгоритму. По-друге, були обґрунтовані зауваження щодо принципу обчислення ваг ребер.

Первинний алгоритм вимагає наявності трьох проміжних сегментації у зв'язку з тим, що простір RGB є перцептивно неоднорідним. Відповідно використання евклідової відстані між кольорами в цьому просторі, в якості ваги ребра, є недоречним. Проте, якщо перейти від простору RGB до простору CIELAB, то ця сама метрика є цілком придатною для визначення різниці між кольорами. У просторі CIELAB евклідова відстань між кольорами означена як метрика CIE74. Ця метрика приваблює багатьох дослідників своєю "легкістю" в обрахунках, проте вона надає гіршу оцінку, ніж складніші метрики CIE94 та CIEDE2000. Вибір однієї з цих метрик залежить від вимог до процесу сегментації: якщо необхідно забезпечити високу швидкодію процесу, то варто обрати метрику CIE74, проте, якщо такої вимоги явно не вказано, то для забезпечення високої якості сегментації варто використати CIE94 або CIEDE2000.

Як тільки множина вершин та ребер графа сформовані, розпочинається побудова мінімального каркаса графа за допомогою алгоритму Краскала. Перший крок цього алгоритму полягає у сортуванні ребер за вагою у напрямку зростання. Для цього можна використати ефективний алгоритм сортування Quicksort, який завдяки своїй структурі, легко піддається розпаралеленню. Проте, якщо значення ваг усіх ребер є цілочисельними, то в такому разі краще використати алгоритм counting sort (також відомий як ultra sort чи math sort). Використання вищеописаних алгоритмів дозволяє покращити швидкодію алгоритму Краскала.

Наступним кроком в алгоритмі Краскала є побудова початкового набору множин. Кожна вершина поміщується в окрему множину і таким способом утворюється mn початкових множин. Після цього виконується ітерація по списку ребер: якщо вибране ребро не утворює циклу і предикат порівняння пари областей набуває значення істинності, то множини, в яких знаходяться вершини ребра, зливаються; в протилежному випадку злиття не відбувається. З метою оптимізації процесів пошуку множини за елементом та злиття двох множин набір множин необхідно реалізувати у вигляді структури даних "об'єднання множин, які не перетинаються", також відому як disjoint-sets union (DSU). З метою оптимізації обчислення значення предикату порівняння пари областей, в кожній множині з DSU додатково варто зберігати міру потужності множини та міру внутрішньої різниці множини [4]. У наступному підрозділі детально розглянуто структуру DSU та евристики, які дозволяють оптимізувати її функціонування.

Останнім кроком в роботі ЕГАСЗ є перевірка сегментів на мінімальну потужність, значення якої задається параметром min. Для цього виконується повторна ітерація по списку ребер. Виконується пошук множин, яким належать вершини ребра, і якщо кількість елементів в одній з множин є меншою, ніж значення параметра min, тоді виконується злиття цих множин. Ця процедура дозволяє усунути дрібні сегменти, причому об'єднаються вони з сегментами, які є "найближчими" до них. Такий поведінці сприяє порядок розташування ребер у списку.

Об'єднання множин, які не перетинаються (DSU)

Беззаперечно найвагомішою оптимізацією в програмній реалізації ЕГАСЗ є використання структури даних під назвою "Об'єднання множин, які не перетинаються (DSU)". Вона призначена для виконання операцій над набором множин, які не перетинаються. Зокрема, ця структура даних дає змогу ефективно виконувати операції пошуку множини за елементом та об'єднання двох множин. Характерною ознакою DSU є те, що кожна множина однозначно характеризується представником – одним зі свої елементів. Структура множини в DSU виглядає так: кожний елемент містить вказівник на власного представника (це не завжди представник множини), а представник

множини містить вказівник на самого себе. У термінах теорії графів DSU є лісом, оскільки будь-яка множина в цій структурі являє собою дерево. Корінь цього дерева – це представник множини.

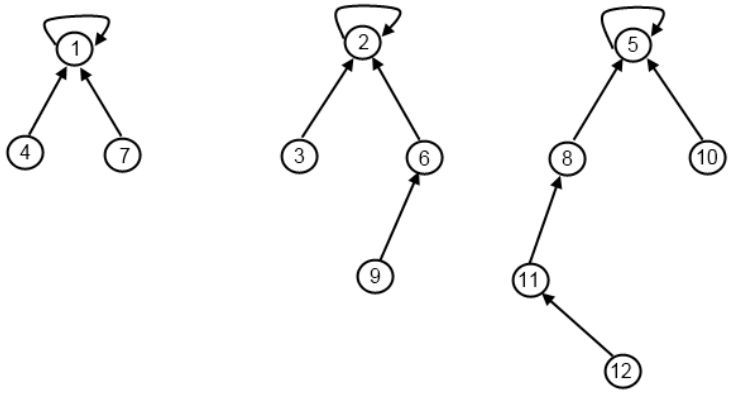

Рис. 2. DSU з трьома множинами

DSU передбачає можливість виконання таких операцій:

1) створення множини з одним елементом, $MakeSet(x)$;

2) пошук множини за елементом, $Find(x)$;

3) об'єднання двох множин, $Union(x, y)$, де *x* та *y* елементи двох різних множин.

Операція створення множини дозволяє породити множину з єдиним елементом. На рис. 3 проілюстровано результат виконання цієї операції для семи різних елементів: утворено DSU, що складається з семи множин. У кожній множині лише по одному елементу, який відповідно виконує функцію представника множини.

Рис. 3. DSU з трьома множинами

Пошук множини за елементом x, $Find(x)$, полягає у знаходженні представника цієї множини. Оскільки кожна множина в DSU має структуру дерева, то для відшукання представника необхідно підніматися догори по дереву, аж поки не буде досягнуто його корінь.

Об'єднання двох множин в структурі даних DSU виконується, присвоюючи представнику однієї з множин представника іншої, у значенні батька.

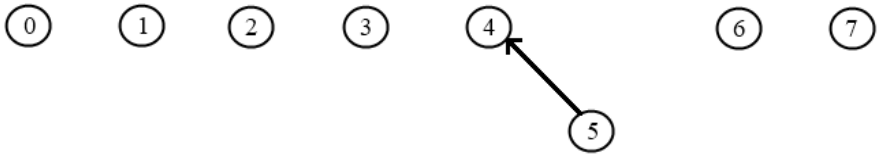

Рис. 4. Результат об'єднання множин з представниками "4" та "5"

Евристика "path compression"

Під час роботи алгоритму Краскала, для побудови сегментації, необхідно виконати велику кількість звернень до функції $Find(x)$. Час виконання цієї операції напряму залежить від висоти дерева, яким є множина в DSU. Зі структури функції видно, що чим більшою є висота дерева, тим довше триває пошук представника множини. Тому логічно проводити топологічну зміну дерева, з метою зменшення його висоти.

Одним з засобів, який дає змогу виконати цю топологічну зміну, є евристика path compression (скорочення шляху). Для того, щоб знайти множину за елементом, достатньо піднятись догори по дереву до його кореня. Основна ідея евристики "path compression" полягає в такому: як тільки представника множини знайдено, необхідно, шляхом використання засобів рекурсії, повернутися

назад до елемента, від якого розпочався пошук; крім того на зворотному шляху кожній проміжній вершині потрібно присвоїти представника множини за батька. Таким чином під час наступного пошуку множини, наприклад, за елементом c, пошук представника множини виконається швидше. Результат застосування евристики “path compression” проілюстровано на рис. 5.

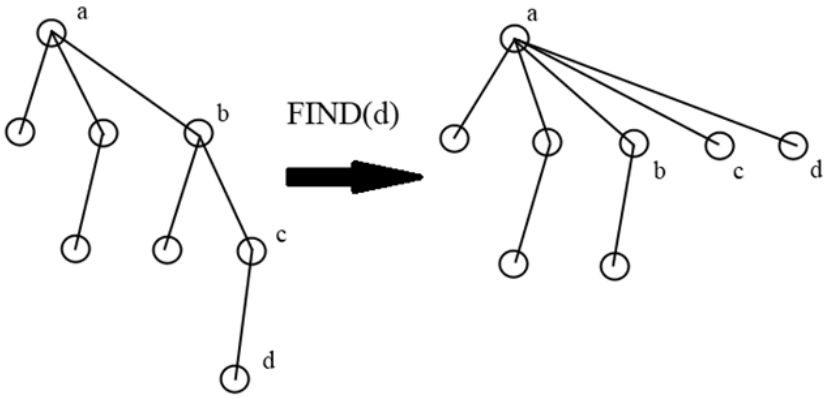

Рис. 5. Скорочення шляху під час пошуку множини за елементом “d”

Евристика “union by rank”

Ця евристика також використовується з метою зменшення висоти дерева. Об’єднання двох множин в DSU відбувається присвоєнням представнику однієї з множин представника іншої за батька. Суть евристики “union by rank” полягає в тому, що під час операції об’єднання двох множин, представнику множини-дерева з ймовірно меншою висотою (рангом) необхідно присвоїти за батька представника множини-дерева з ймовірно більшою висотою. Такий підхід дозволяє не збільшувати висоту результуючого дерева без потреби. Якщо ж множини, що підлягають об’єднанню, рівні за рангом, тоді будь-яка з них приєднується до іншої, а ранг результуючої множини збільшується на 1. Приклад застосування цієї евристики проілюстровано на рис. 6.

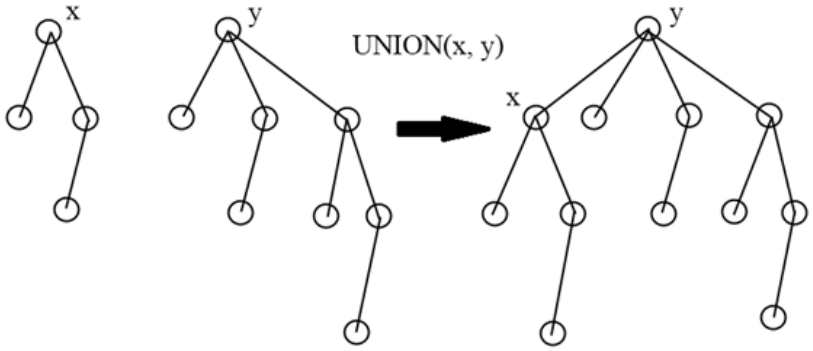

Рис. 6. Об’єднання за рангом множин з представниками x та y

Результати програмного експерименту

Одним з ефективних інструментів дослідження ЕГАСЗ стала його програмна реалізація, яка дозволила наочно оцінити вплив значень вхідних параметрів алгоритму на якість сегментації графічного образу. Розроблене під час дослідження ПЗ працює за таким алгоритмом:

1) завантажується графічний образ;
2) задаються значення вхідних параметрів алгоритму: s , k , min , спосіб побудови ребер, метрика оцінки колірної різниці;
3) виконується побудова сегментації графічного образу за заданими параметрами;
4) проводиться оцінка якості отриманої сегментації за допомогою метрик кластерної придатності $\overline{r}$ та l , які означені в роботі [7];
5) виконується візуалізація сегментації. Кожен сегмент відображається унікальним кольором.

Для зручності у порівнянні значення заданих параметрів відображаються під сегментацією, а значення мір придатності над нею. Грунтуючись на програмних результатах, оцінимо окремо вплив значень кожного параметра на якість сегментації.

Першим розглянемо параметр s. Для переважної більшості зображень застосування гауссового згладжування покращує якість сегментації, оскільки зазвичай зображення є зашумленими. На рис. 7 проілюстровано сегментацію зображення при значеннях s з проміжку $[0;2]$. З рисунка видно, що відсутність гауссового згладжування або занадто велике значення s погіршує якість сегментації.

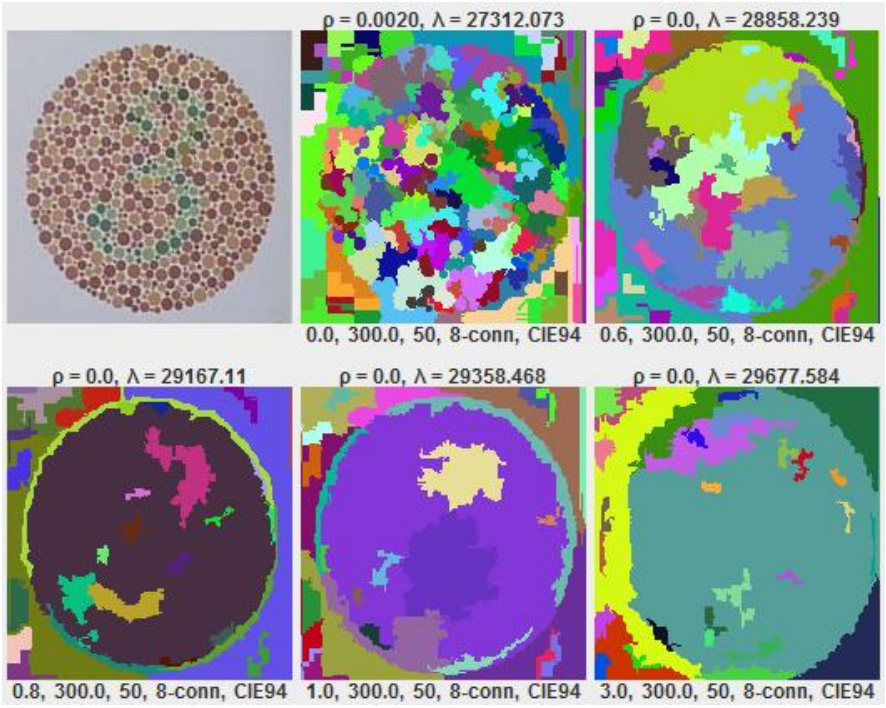

Рис. 7. Сегментація зображення при різних значеннях s

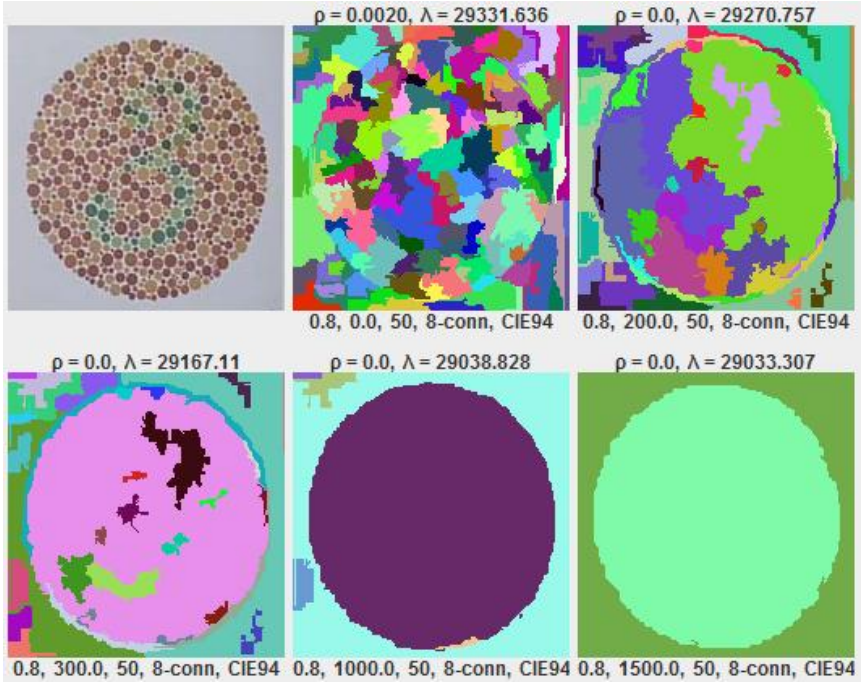

Рис. 8. Сегментація зображення при різних значеннях k

Параметр k відповідає за масштаб об'єктів спостереження [4]. На рис. 8 показано, що більші значення цього параметра сприяють формуванню більших сегментів. Для прикладу, при $k = 300$ частини числа "3" є помітні, в той час як при $k = 1500$ їх вже не видно.

Роль параметра min великою мірою подібна до ролі параметра k. Відмінність полягає в тому, що min використовується при об'єднанні "найближчих" областей на підставі їхнього розміру, в той час як k – в предикаті порівняння областей. Зауважимо, що максимальне значення параметра min для зображення розміром $m \times n$ не повинно перевищувати $\left\lfloor \frac{mn}{2} \right\rfloor$.

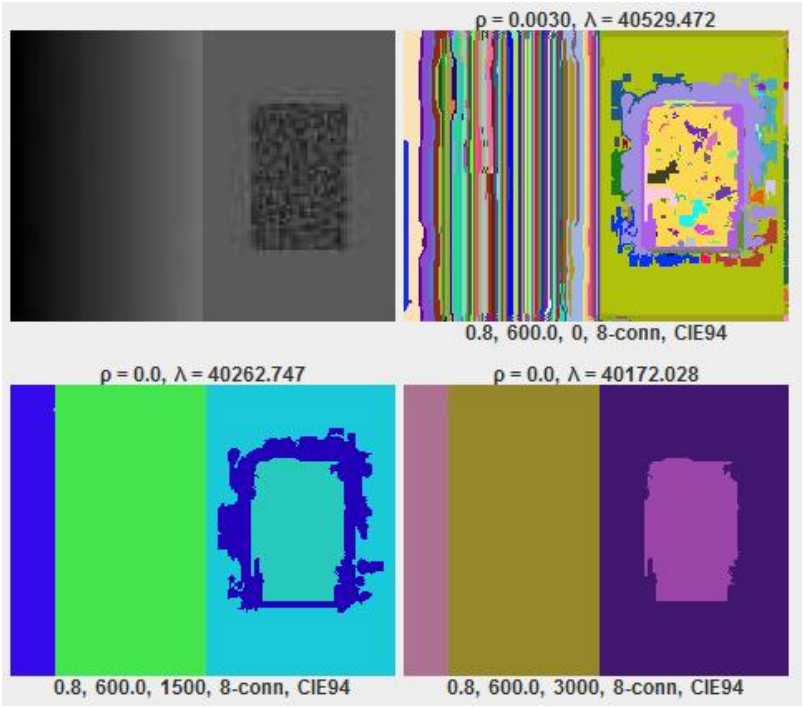

Рис. 9. Сегментація зображення при різних значеннях min

Єдиний параметр, який однозначно впливає на процес сегментації будь-якого зображення – це спосіб зв'язування вершин графа. Його поведінку можна охарактеризувати так: чим вищий ступінь кожної вершини графа, тим якіснішим є результат сегментації графічного образу.

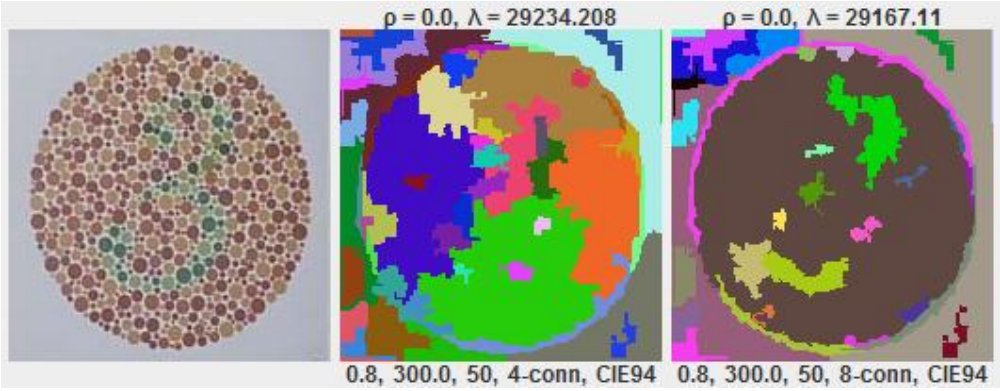

Рис. 10. Сегментація зображення за різних способів "зв'язування"

Вибір метрики для визначення різниці між пікселями має значний вплив на результуючу сегментацію. На рис. 11 проілюстровано результат використання метрик CIE94 та CIE76. Оскільки метрика CIE94 є покращенням CIE76, то відповідно результат сегментації є якіснішим.

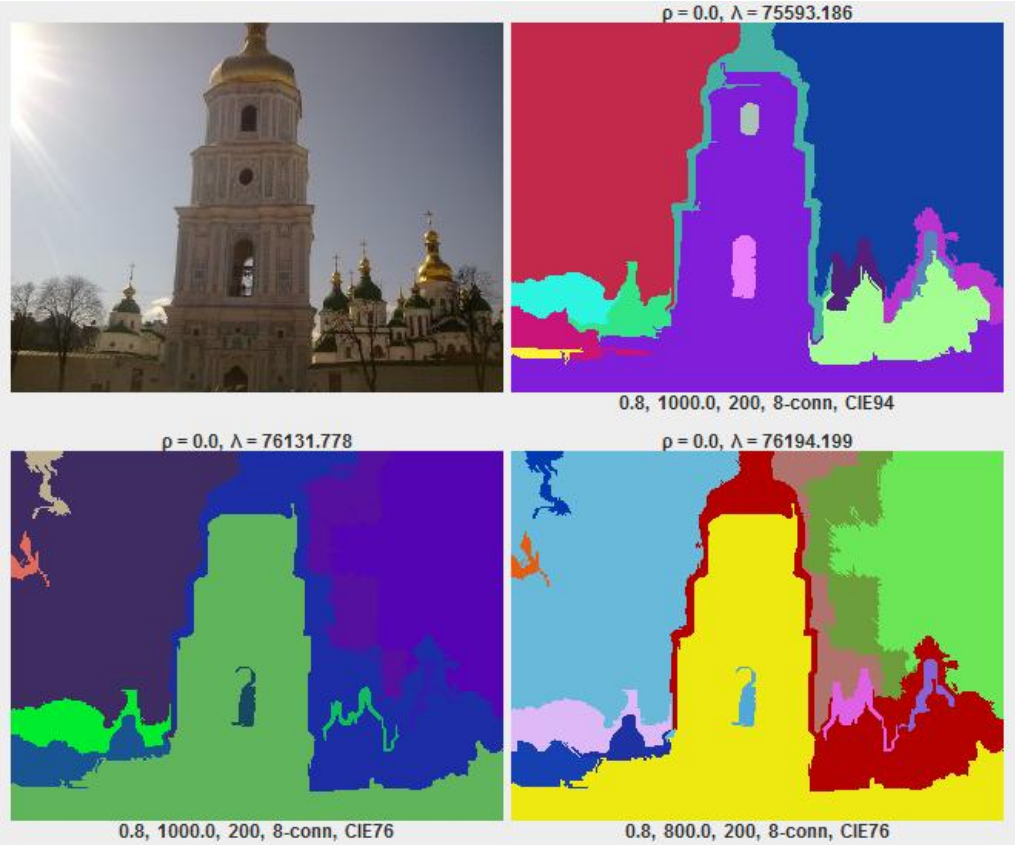

Рис. 11. Сегментація зображення з використанням CIE94 та CIE76

Висновки

З метою оптимізації швидкодії та підвищення якості сегментації графічного образу проведено дослідження структури ЕГСАЗ, під час якого кожен крок алгоритму був підданий ретельному аналізу. Найвагомішою оптимізацією швидкодії ЕГАСЗ є безумовно використання структури DSU з евристиками “path compression” та “union by rank”. Програмна реалізація алгоритму дозволила оцінити залежність якості сегментації від значень вхідних параметрів. Завдяки ПЗ отримано такі наукові результати:

1) виявлено значну залежність результатів сегментації від вхідних параметрів. Звідси зроблено такий висновок: ключовим напрямом розвитку алгоритму є розв’язання задачі підбору оптимальних вхідних параметрів алгоритму;

2) оцінено залежність якості результату сегментації. Згідно з оцінкою, для кожного графічного образу існує оптимальний набір значень параметрів алгоритму s, k, min, за якого результуюча сегментація має найкращу якість;

3) виявлено неможливість прямого застосування мір оцінки якості кластеризації $\overline{r}$ та l. Проте не варто відкидати можливість розроблення спеціальних евристик, які будуть використовувати значення цих метрик для оцінювання якості результатів сегментації.

1. Вихарев Д. Эффективная сегментация изображений на графах. – [Електронний ресурс]: http://habrahabr.ru/blogs/algorithm/81279/. 2. Нікольський Ю.В., Пасічник В. В., Щербина Ю.М. Дискретна математика. – К.: Видавнича група BHV, 2007. – 368 с. 3. Daniel Rakos. Efficient Gaussian blur with linear sampling. – [Електронний ресурс]: http://rastergrid.com/blog/2010/09/efficient-gaussian-blur-with-linear-sampling/. 4. Felzenszwalb P, Huttenlocher D. Efficient Graph-Based Image Segmentation // International Journal of Computer Vision. – 2004. – Vol. 59, no. 2. – P. 167–181. 5. Mitzenmacher M. Lecture 6: Disjoint set (Union-Find). – [Електронний ресурс]: http://www.fas.harvard.edu/~libcs124/CS/lec6.pdf. 6. Stein B., Niggemann O. On the Nature of Structure and its Identification // P. Widmayer, G. Neyer, S. Eidenbenz (eds.). Graph-Theoretic Concepts in Computer

Science. LNCS 1665. – Springer-Verlag, 1999. – P. 122–134. 7. Stein B., Meyer zu Eissen S., Wißbrock F. On Cluster Validity and the Information Need of Users // Proceedings of the 3rd IASTED International Conference on Artificial Intelligence and Applications (AIA'03). – Benalmadena, Spain: September 8-10, 2003. – P. 216–221. 8. Wassenberg *J.,* Middelmann *W., Sanders P. An Efficient Parallel Algorithm for Graph-Based Image Segmentation // Proceedings of the 13th International Conference on Computer Analysis of Images and Patterns (CAIP'09). – Münster, Germany: September 2–4, 2009. – P. 1003–1010.*

УДК 004.89

В.В. Литвин
Національний університет "Львівська політехніка",
кафедра інформаційних систем та мереж

ЗАДАЧІ ОПТИМІЗАЦІЇ СТРУКТУРИ ТА ЗМІСТУ ОНТОЛОГІЇ ТА МЕТОДИ ЇХ РОЗВ'ЯЗУВАННЯ

Розглянуто оптимізаційні задачі, які виникають під час автоматичної розбудови онтології щодо її стурутури та змісту. Сформовано низку таких задач. Розглянуто методи та алгоритми їх розв'язування.

Ключові слова: онтологія, концепт, інтелектуальний агент, структура онтології, зміст онтології, оптимізаційна задача.

In the paper the evaluation of the optimization problems that arise during the automatic ontology building on its sturutury and content. Formed a series of such problems. The methods and algorithms for their solution.

Key words: ontology, concept, intelligent agent, ontology structure, ontology content, optimization task.

Вступ. Постановка проблеми у загальному вигляді

Автоматична розбудова онтології призводить до виникнення недоліків у її структурі та змісті, невідповідності її наповнення інформаційним потребам користувача [1, 2]. Тому подібні системи необхідно "укомплектовувати" відповідним набором процедур оптимізації онтології, тобто оптимізаційною компонентою. Актуальність задачі оптимізації внутрішнього подання баз знань (БЗ), що дає змогу істотно підвищити ефективність функціонування інтелектуальних систем, також зазначається в роботі Пузанкова [3]. Нижче розглянуто методику динамічної побудови онтології, а також процедури її оптимізації.

Складна структура взаємозв'язків між поняттями, відображеними в онтології, а також її динамічне наповнення під час експлуатації вимагає застосування певних оптимізаційних процедур з метою мінімізації часу реакції на запити; неперевищення місця, відведеного для розміщення онтології; вирішення конфліктів між даними, внесеними з різних джерел, а також задоволення інших вимог та критеріїв, які належить визначити. Оптимізація онтології також виконується з метою адаптації її змісту до інформаційних потреб користувачів, вилучаючи елементи, які рідко використовуються або ж не використовуються взагалі, тобто не належать до певної предметної області (ПО).

У роботах [4, 5] обґрунтовано доцільність подання онтології у вигляді концептуального графу, вершинам якого присвоєно певні семантичні та числові характеристики. Він є зваженим графом, в якому на етапі формування допускається існування паралельних ребер, циклів, петель, дублювання вершин з аналогічними параметрами та інших особливостей. Виявлення й усунення таких особливостей з метою нормалізації графу онтології, а також оптимізація графу за визначени-