СЕКЦІЯ «ПРИКЛАДНА МАТЕМАТИКА»

УДК 519.6:004.421

DOI 10.31651/2076-5886-2024-1-4-21

PACS 02.10.Ox, 02.70.Hm, 07.05.Tp

ГЛАДКА Людмила Іванівна
к.ф.-м.н., доцент кафедри автоматизації та комп'ютерно-інтегрованих технологій, Черкаський національний університет імені Б. Хмельницького
e-mail: l_i_gladka@vu.cdu.edu.ua
ORCID 0000-0002-7030-9666

ЧАЛИЙ Антон Анатолійович
розробник ПЗ, ТОВ «ЮніСтар», м. Сміла
e-mail: anton.chaliy@gmail.com

ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ АЛГОРИТМІВ ПОШУКУ МАРШРУТУ НА ЛАБІРИНТАХ ТА ДВОВИМІРНИХ СІТКОВИХ СТРУКТУРАХ

У статті проведено порівняльний аналіз швидкодії алгоритму Дейкстри, алгоритму A та алгоритму пошуку точок переходу (Jump Point Search, JPS) на двовимірних сіткових структурах. Для формування тестових середовищ реалізовано генерацію лабіринтів на основі алгоритму бінарного дерева та рандомізованого алгоритму Прима, а також побудову мап із різною щільністю перешкод шляхом модифікації згенерованих структур. Порівняння алгоритмів здійснювалось за сумарним часом виконання в ході серії з 1000 ітерацій пошуку між випадково обраними парами точок на сітках розміром 255×255 клітинок. Визначено залежність ефективності алгоритмів від топології простору та щільності перешкод, сформульовано рекомендації щодо вибору алгоритму залежно від характеристик середовища. Отримані результати можуть бути використані під час проєктування навігаційних систем, ігрових застосунків та робототехнічних комплексів.*

Ключові слова: *пошук маршруту; алгоритм Дейкстри; алгоритм A*; Jump Point Search; евристичний пошук; двовимірна сітка; оптимізація маршрутів; генерація лабіринтів; навігаційні системи.*

Вступ

У сучасних інформаційних системах задачі пошуку оптимальних маршрутів посідають важливе місце та знаходять широке застосування у різних прикладних сферах, зокрема в навігаційних системах, робототехніці, телекомунікаційних мережах і індустрії відеоігор. Ефективність розв’язання таких задач безпосередньо впливає на продуктивність програмного забезпечення, особливо в умовах обмежених обчислювальних ресурсів та необхідності обробки даних у реальному часі.

Теоретичною основою задач маршрутизації є теорія графів, у межах якої простір моделюється у вигляді множини вершин та зв’язків між ними. Такий підхід дозволяє формалізувати процес пошуку шляху та застосовувати ефективні алгоритмічні методи для знаходження оптимальних або близьких до оптимальних рішень.

Особливий інтерес становить застосування алгоритмів пошуку маршруту у відеоіграх, де вони використовуються для керування рухом персонажів, побудови траєкторій та моделювання поведінки об’єктів. На відміну від класичних задач маршрутизації, у цьому середовищі додатково накладаються обмеження, пов’язані з необхідністю швидкого прийняття рішень, динамічністю середовища та великою

кількістю одночасних запитів на пошук шляхів. У більшості випадків ігровий простір моделюється у вигляді двовимірної прямокутної сітки, що визначає специфіку застосовуваних алгоритмів.

Актуальність дослідження зумовлена необхідністю підвищення ефективності алгоритмів пошуку маршруту в умовах обмежених ресурсів і різноманітних структур середовища. Незважаючи на значну кількість відомих алгоритмів, їх практична ефективність суттєво залежить від характеристик простору пошуку, таких як структура перешкод, щільність середовища та топологія графа.

Метою даного дослідження є порівняльний аналіз швидкодії поширених алгоритмів пошуку маршруту та визначення їх ефективності залежно від типу середовища на двовимірній сітці.

Для досягнення поставленої мети сформульовано такі завдання:

– реалізувати алгоритми генерації лабіринтів різних типів;
– реалізувати методи побудови мап на основі згенерованих лабіринтів;
– обґрунтувати вибір технологій для представлення та обробки середовища;
– реалізувати алгоритми пошуку маршруту, зокрема алгоритм Дейкстри, A* та алгоритм пошуку точок переходу;
– розробити методику експериментального порівняння алгоритмів;
– провести експериментальні дослідження на різних типах середовищ;
– здійснити аналіз отриманих результатів і визначити області ефективного застосування кожного алгоритму.

Наукова новизна роботи полягає у проведенні системного порівняльного аналізу алгоритмів пошуку маршруту в умовах різних типів структурованих середовищ (лабіринтів і мап), з урахуванням впливу топології простору на їх швидкодію.

Практичне значення отриманих результатів полягає у можливості використання сформульованих рекомендацій для вибору ефективних алгоритмів пошуку маршруту в задачах розробки програмного забезпечення, зокрема в ігрових застосуваннях та системах, що працюють у реальному часі.

Виклад основного матеріалу

Для реалізації поставлених у роботі задач було визначено низку вимог до програмних засобів та технологій, зокрема: підтримка складних структур даних, можливість ефективної роботи з графовими моделями та двовимірними сітками, наявність засобів для обробки растрових зображень, кросплатформність, а також достатня гнучкість для реалізації алгоритмів пошуку маршруту.

З урахуванням зазначених вимог як основний інструмент розробки було обрано мову програмування Python [3]. Вибір Python зумовлений її широким використанням у сфері розробки алгоритмічних та дослідницьких програмних рішень, наявністю значної кількості спеціалізованих бібліотек, а також підтримкою об’єктно-орієнтованого та функціонального підходів до програмування. Крім того, Python забезпечує зручні засоби роботи зі структурами даних, що є важливим під час реалізації алгоритмів пошуку на графах і сіткових середовищах.

Для роботи із зображеннями використано бібліотеку Pillow, яка є розширенням бібліотеки Python Imaging Library (PIL). Дана бібліотека забезпечує можливість завантаження, створення, редагування та збереження растрових зображень у різних графічних форматах. Використання Pillow дозволило представляти лабіринти та мапи у вигляді двовимірних масивів пікселів, що спрощує реалізацію алгоритмів генерації середовища та візуалізації знайдених маршрутів.

Для збереження сформованих зображень було використано формат BMP із кольоровою моделлю RGB. Такий підхід забезпечує коректне та наочне відображення

побудованих маршрутів на фоні лабіринтів і мап, а також мінімізує втрати якості під час збереження графічних даних.

1. Алгоритми побудови

1.1. Побудова лабіринту на основі бінарного дерева

Одним із базових методів генерації лабіринтів є алгоритм побудови на основі бінарного дерева. Даний підхід характеризується простотою реалізації та забезпечує формування зв'язного лабіринту без циклів.

На початковому етапі формується прямокутна сітка, у якій прохідні клітинки та стіни розташовуються почергово. Прохідні області позначаються світлими клітинками, тоді як непрохідні ділянки – темними.

Подальша генерація лабіринту виконується шляхом послідовного обходу всіх прохідних клітинок. Для кожної клітинки випадковим чином обирається один із двох можливих напрямків – угору або праворуч. Якщо клітинка розташована на межі сітки та один із напрямків недоступний, вибір здійснюється лише серед допустимих варіантів. Після визначення напрямку відповідна стіна видаляється, що створює новий прохід між сусідніми клітинками.

У результаті виконання алгоритму формується однозв'язний лабіринт без замкнених циклів, у якому між будь-якими двома точками існує єдиний оптимальний маршрут. Характерною особливістю такого методу є наявність структурного зміщення у напрямку верхнього правого кута, що проявляється у переважній орієнтації проходів та відкритості верхньої і правої меж лабіринту (рис. 1).

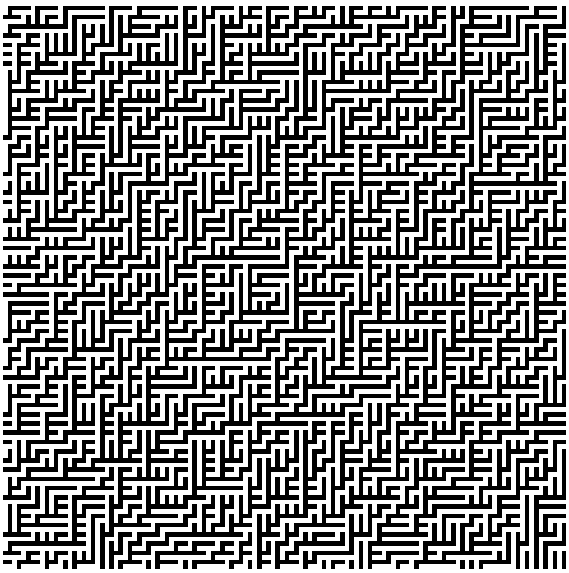

Рис. 1. Лабіринт, побудований на основі алгоритму бінарного дерева

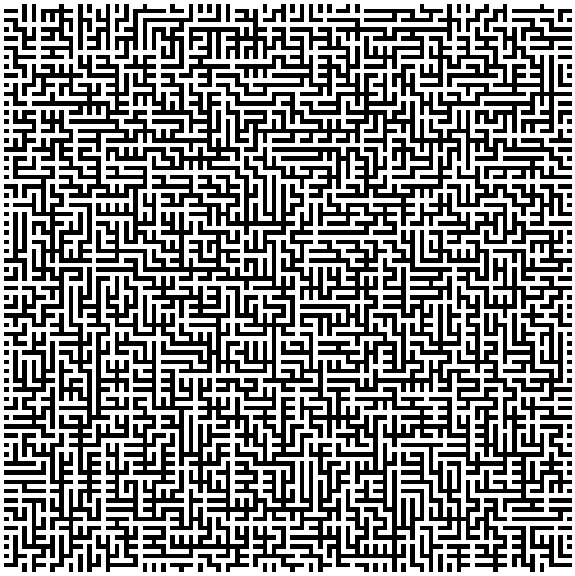

Рис. 2. Мапа на основі лабіринту з рис. 1

1.2. Побудова лабіринту на основі рандомізованого алгоритму Прима

Алгоритм Прима належить до жадібних алгоритмів та використовується для побудови мінімального кістякового дерева у зв'язному неорієнтованому графі. Для генерації лабіринтів застосовується його рандомізована модифікація, яка дозволяє формувати випадкові, але зв'язні структури проходів.

Побудова лабіринту починається із сітки, повністю заповненої стінами. На першому етапі випадковим чином обирається стартова клітинка, яка додається до

множини прохідних точок лабіринту. Після цього всі суміжні зі стартовою клітинкою стіни заносяться до окремого списку.

Далі алгоритм працює ітеративно. Із переліку стін випадковим чином вибирається одна стіна, після чого перевіряється можливість створення проходу через неї до нової клітинки. Якщо така клітинка ще не належить лабіринту, стіна видаляється, а нова клітинка додається до множини прохідних точок. Після цього до списку додаються всі прилеглі до нової клітинки стіни. Процес повторюється доти, доки список доступних стін не стане порожнім.

У результаті формується лабіринт із більш природною та менш регулярною структурою порівняно з лабіринтом, побудованим на основі бінарного дерева. Як показано на рис. 2, для такого лабіринту не характерне виражене зміщення у певному напрямку, а також відсутні порожні області вздовж меж сітки. Крім того, у структурі можуть виникати хрестоподібні перехрестя та складніші конфігурації проходів, що не зустрічаються при використанні бінарного алгоритму.

Важливою особливістю алгоритму є те, що сформований лабіринт залишається однозв'язним, тобто між будь-якими двома точками існує шлях. При цьому структура не містить ізольованих областей та циклів, що відповідає властивостям мінімального кістякового дерева.

Використання двох різних підходів до генерації лабіринтів – алгоритму бінарного дерева та рандомізованого алгоритму Прима – дозволяє отримати середовища з різною структурою проходів і характером перешкод. Це створює достатню основу для подальшого порівняльного аналізу алгоритмів пошуку маршруту, а також для побудови більш складних карт, що розглядаються в наступному підрозділі.

1.3. Побудова мап на основі лабіринтів

Лабіринти є зручним середовищем для тестування алгоритмів пошуку маршруту, однак вони не повністю відображають структуру реальних ігрових або навігаційних карт. У практичних задачах маршрутизації часто зустрічаються як вузькі проходи, так і відкриті області з перешкодами різної форми та густоти. Тому для формування більш універсальних карт у даній роботі використано підхід, заснований на модифікації згенерованих лабіринтів.

Метод побудови карт складається з декількох послідовних етапів та базується на поєднанні процедур генерації лабіринтів і подальшої трансформації їх структури.

На першому етапі формується базовий лабіринт. Для прикладу використовується лабіринт, побудований на основі бінарного дерева (див. рис. 1).

Другий етап передбачає видалення частини тупикових проходів з метою зменшення густоти лабіринту. Для цього виконується послідовний перегляд усіх прохідних клітинок. Якщо клітинка має лише одного сусіда, вона вважається тупиковою та додається до списку на видалення. Після завершення обходу всі знайдені тупикові точки видаляються. Процедура може повторюватися декілька разів залежно від бажаної структури карти. У наведеному прикладі операція виконувалась тричі.

Кількість ітерацій безпосередньо впливає на форму та ширину проходів у майбутній карті. Крім того, на рис. 3 можна помітити характерне зміщення структури у напрямку верхнього правого кута, властиве лабіринтам, побудованим за бінарним алгоритмом.

Наступним етапом є розширення проходів методом «нарощення». Для цього аналізуються всі клітинки поточного лабіринту та формується список прилеглих стін. Якщо стіна має три або більше сусідніх прохідних клітинок, вона перетворюється на прохід і додається до структури карти. Такий підхід дозволяє збільшити ширину коридорів та створити відкриті області.

Кількість повторень цього процесу визначає ступінь відкритості карти та густоту перешкод. Після завершення процедури, за необхідності, може повторно виконуватись видалення тупикових ділянок для згладжування форми проходів та усунення надлишкових вузьких областей.

На рис. 3 показано бінарний лабіринт (з рис. 1) після видалення кількох тупиків, що дозволяє зробити лабіринт менш щільним.

На рис. 4 представлена карта, отримана після нарощення проходів на основі бінарного лабіринту з рис. 1. Видно, що напрямок розширення коридорів виражається у продовгуватих перешкодах.

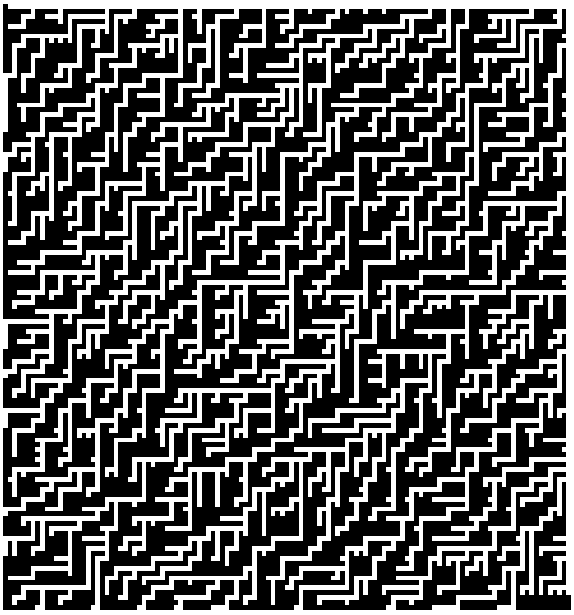

Рис. 3. Лабіринт з рис. 1 після трьох ітерацій видалення глухих кутів.

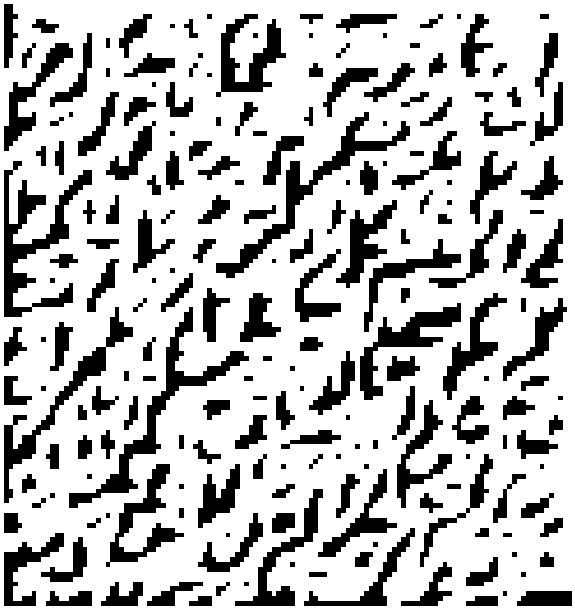

Рис. 4. Мапа на основі лабіринту з рис. 1

На рис. 5 представлено результат модифікації лабіринту, побудованого на основі рандомізованого алгоритму Прима (див. рис. 2), після кількох ітерацій видалення тупикових проходів. Застосування цієї процедури дозволяє суттєво зменшити густоту вузьких коридорів та сформувати більш відкриту структуру середовища, придатну для подальшого аналізу алгоритмів маршрутизації.

На рис. 6 наведено карту, сформовану на основі лабіринту Прима після виконання етапу розширення проходів методом «нарощення». У результаті такої трансформації структура карти набуває більш природного вигляду, характерного для реальних ігрових або навігаційних середовищ, де присутні як вузькі проходи, так і відкриті області різної конфігурації.

На рис. 4 помітно, що особливості вихідного бінарного лабіринту впливають на форму перешкод, які мають виражену орієнтацію в одному напрямку.

Аналогічні процедури були виконані для лабіринту, побудованого за рандомізованим алгоритмом Прима.

Порівняння рис. 4 та рис. 6 демонструє, що структура кінцевої карти значною мірою залежить від алгоритму генерації початкового лабіринту. Карти, побудовані на основі алгоритму Прима, мають більш рівномірний розподіл перешкод та менш виражену орієнтацію проходів, тоді як карти на основі бінарного дерева характеризуються певною напрямленістю структури. Це дозволяє використовувати

сформовані карти для дослідження ефективності алгоритмів пошуку маршруту в середовищах різної складності та конфігурації.

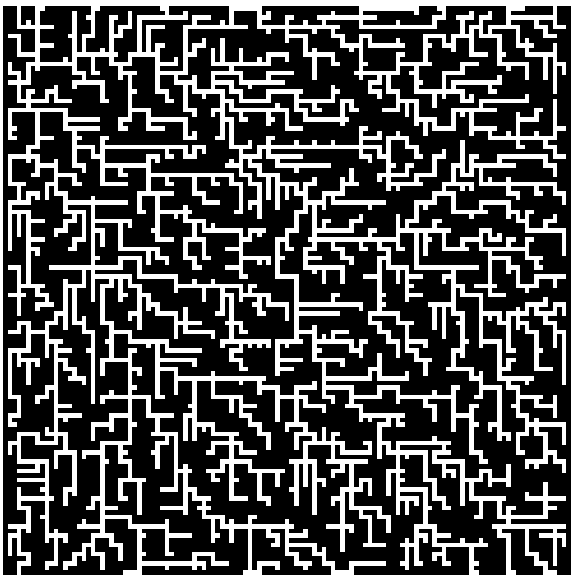

Рис. 5. Лабіринт на основі алгоритму Прима після трьох ітерацій видалення тупикових проходів.

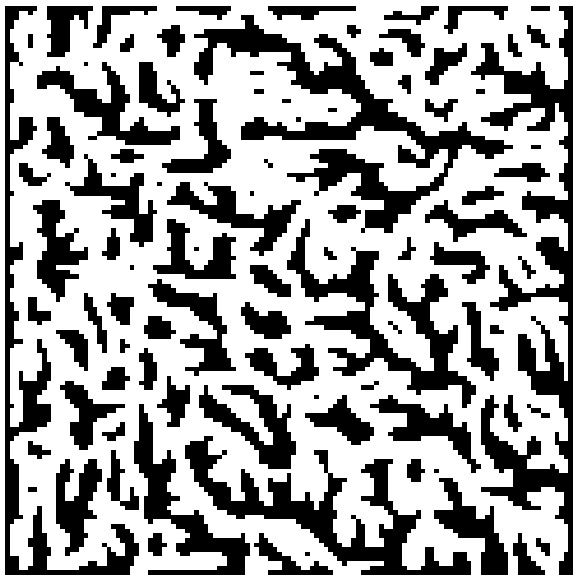

Рис. 6. Мапа на основі лабіринту з рис. 2

Порівняння карт, наведених на рис. 4 та рис. 6, демонструє суттєвий вплив початкового алгоритму генерації лабіринту на структуру кінцевого середовища. Карти, побудовані на основі бінарного дерева, характеризуються помітною напрямленістю та витягнутими перешкодами, тоді як карти, сформовані на основі алгоритму Прима, мають більш хаотичний та рівномірний розподіл прохідних зон і перешкод. Такі відмінності дозволяють формувати тестові середовища з різним рівнем складності та використовувати їх для комплексного дослідження ефективності алгоритмів пошуку маршруту.

2. Алгоритми пошуку маршруту

2.1. Алгоритм Дейкстри

Алгоритм Дейкстри є класичним алгоритмом теорії графів, призначеним для знаходження найкоротших шляхів від заданої початкової вершини до всіх інших вершин зваженого графа з невід'ємними вагами ребер [3].

Нехай граф задано як $G=(V,E)$, де V – множина вершин, а E – множина ребер. Алгоритм ґрунтується на ітеративному уточненні оцінок відстаней $d(v)$ від початкової вершини s до всіх інших вершин шляхом послідовної обробки вершин у порядку зростання поточної оцінки відстані.

У рамках даної роботи граф представляється у вигляді прямокутної сітки, де вершини відповідають клітинкам, а ребра – можливим переходам між ними. Вартість переходу між сусідніми клітинками задається як стала величина (для ортогональних і діагональних переміщень).

Узагальнений алгоритм Дейкстри можна подати у вигляді наступної послідовності кроків:

1. Ініціалізація: для всіх вершин $v \in V$ задається $d(v)=\infty$, для початкової вершини s – $d(s)=0$.
2. Початкова вершина додається до черги з пріоритетом, де пріоритет визначається значенням $d(v)$.
3. Поки черга не порожня:
   - обирається вершина з мінімальним значенням $d(v)$;
   - для кожної суміжної вершини виконується операція релаксації: $d(u)=min(d(u),d(v)+w(v,u))$, де $w(v,u)$ – вага ребра;
   - у разі оновлення значення $d(u)$ відповідна вершина додається до черги.

Після завершення обчислень маршрут до цільової вершини відновлюється за допомогою збережених попередників.

У даній реалізації алгоритм виконується без дострокового завершення після досягнення цільової вершини, що дозволяє отримати повну інформацію про найкоротші шляхи до всіх доступних вершин графа.

Основною перевагою алгоритму є його універсальність і гарантована оптимальність знайдених шляхів. Водночас його суттєвим недоліком є висока обчислювальна складність у задачах з великим простором пошуку, оскільки алгоритм не використовує інформацію про розташування цільової вершини.

На рис. 7 наведено приклад маршруту, знайденого алгоритмом Дейкстри.

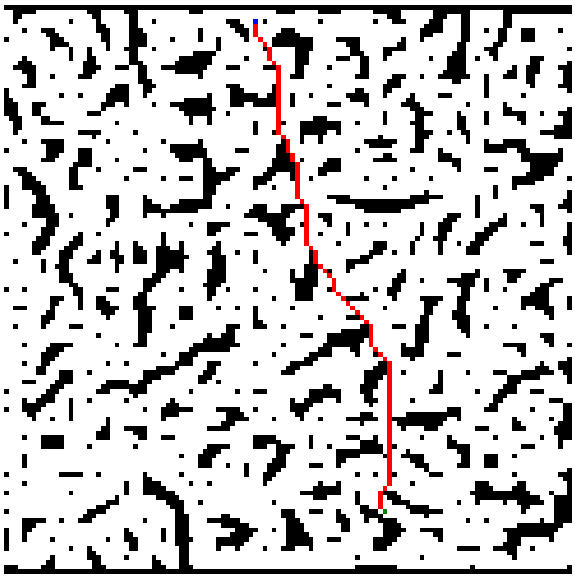

Рис. 7. Маршрут, побудований алгоритмом Дейкстри

2.2. Алгоритм A*

Алгоритм A* належить до класу евристичних алгоритмів пошуку та використовується для знаходження найкоротшого шляху між двома вершинами графа з невід'ємними вагами ребер [4]. Алгоритм було запропоновано у 1968 році та він поєднує властивості алгоритму Дейкстри та жадібного пошуку.

Основною особливістю алгоритму A* є використання оціночної функції:

$$f(n) = g(n) + h(n),$$

де $g(n)$ – вартість шляху від початкової вершини до поточної вершини n, а $h(n)$ – евристична оцінка вартості шляху від вершини n до цільової.

За умови допустимості евристичної функції (тобто $h(n)$ не переоцінює реальну

відстань), алгоритм гарантує знаходження оптимального маршруту.

Нами для евристичної оцінки використано евклідову відстань між вершинами, що є доцільним у випадку, коли дозволені діагональні переміщення на сітці.

Алгоритм A* функціонує аналогічно до алгоритму Дейкстри, однак вибір наступної вершини для обробки здійснюється з урахуванням значення функції $f(n)$, що дозволяє спрямовувати пошук у напрямку цільової вершини та зменшувати кількість розглянутих вершин.

На відміну від алгоритму Дейкстри, у даній реалізації A* пошук припиняється одразу після досягнення цільової вершини, що дозволяє додатково скоротити час виконання.

Основною перевагою алгоритму є значно вища ефективність порівняно з алгоритмом Дейкстри за рахунок використання евристики. Крім того, алгоритм є гнучким у налаштуванні, оскільки вибір евристичної функції дозволяє адаптувати поведінку пошуку до конкретних умов задачі.

На рис. 8 наведено приклад маршруту, знайденого алгоритмом A*.

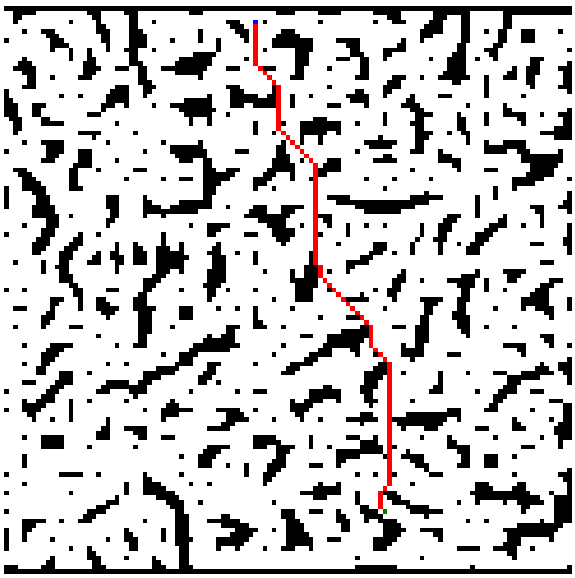

Рис. 8. Маршрут, побудований алгоритмом A*

Порівняння маршрутів, зображених на рис. 7 та рис. 8, показує, що, незважаючи на однакову довжину, отримані шляхи можуть відрізнятися за конфігурацією, що зумовлено різною стратегією обходу простору пошуку.

2.3. Алгоритм пошуку точок переходу

Алгоритм пошуку точок переходу (англ. Jump Point Search, JPS) є оптимізацією алгоритму A*, орієнтованою на роботу з прямокутними сітками з рівномірною вартістю переміщення. Запропонований у 2011 році, цей підхід спрямований на зменшення кількості розглянутих вершин шляхом усунення симетричних варіантів маршрутів та обмеження області пошуку лише ключовими точками.

Основна ідея алгоритму полягає у скороченні простору пошуку за рахунок розгляду не всіх сусідніх вершин, як у класичних алгоритмах, а лише так званих точок переходу (jump points), які визначаються відповідно до певних правил поширення. Це дозволяє уникнути надлишкових обчислень і значно підвищити ефективність пошуку.

Алгоритм базується на двох ключових механізмах: відсічення сусідів та виявлення вимушених сусідів.

Відсікання сусідів полягає у виключенні з розгляду тих сусідніх вершин, до яких

існує альтернативний шлях такої ж або меншої довжини через інші вершини. Таким чином, алгоритм уникає дослідження симетричних маршрутів, що не впливають на оптимальність результату.

Вимушені сусіди виникають у випадках, коли наявність перешкод змінює структуру допустимих шляхів. Якщо оптимальний маршрут до певної вершини може проходити лише через поточну вершину, така вершина повинна бути включена до розгляду незалежно від загальних правил відсічення.

Процес дослідження простору пошуку в алгоритмі JPS має спрямований характер. Як показано на рис. 9, розширення виконується переважно у напрямках, визначених попереднім кроком, причому переміщення по горизонталі та вертикалі мають пріоритет над діагональними. Діагональний рух здійснюється лише після вичерпання можливостей у відповідних ортогональних напрямках.

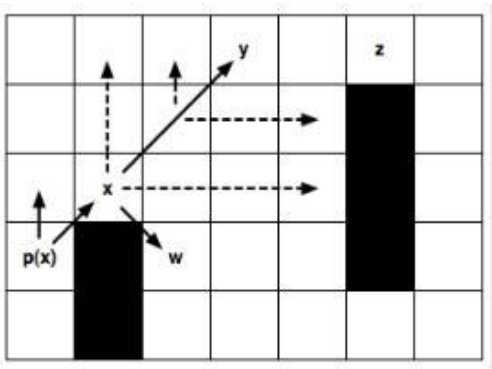

Рис. 9. Приклад дослідження точок

Точка переходу визначається як вершина, що:

– має хоча б одного вимушеного сусіда;
– є результатом діагонального переміщення, після якого можливе подальше розгалуження пошуку;
– є початковою або цільовою вершиною.

На відміну від алгоритму A*, у якому до черги з пріоритетом додаються всі потенційні вершини, в алгоритмі JPS до розгляду включаються лише точки переходу. Пріоритет визначається аналогічно до A*, як:

$$f(n) = g(n) + h(n).$$

Узагальнений алгоритм можна подати таким чином:

1. Ініціалізація: усім прохідним вершинам призначається початкове значення відстані (нескінченність), початковій вершині – нуль.
2. Початкова вершина додається до черги з пріоритетом.
3. Поки черга не порожня:
   – обирається вершина з мінімальним значенням f(n);
   – виконується пошук точок переходу відповідно до правил поширення;
   – знайдені точки переходу додаються до черги.

Після досягнення цільової вершини здійснюється відновлення маршруту.

Завдяки описаним механізмам алгоритм JPS забезпечує суттєве скорочення кількості досліджуваних вершин порівняно з алгоритмами Дейкстри та A*, що позитивно впливає на швидкодію. Водночас слід зазначити, що його застосування

обмежується середовищами, які можуть бути представлені у вигляді регулярної сітки з однаковою вартістю переміщення.

На рис. 10 наведено приклад маршруту, побудованого за допомогою алгоритму пошуку точок переходу. Як видно, структура маршруту може відрізнятися від результатів інших алгоритмів, однак довжина шляху залишається оптимальною. Зокрема, у випадку симетричних альтернатив алгоритм надає перевагу діагональним переміщенням, що впливає на геометрію отриманого маршруту.

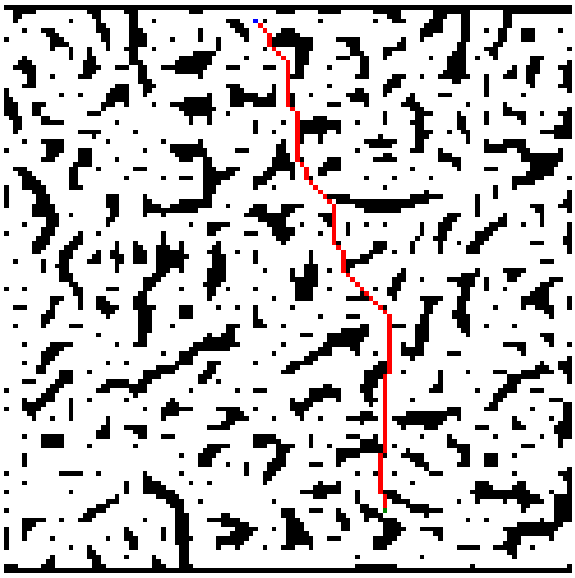

Рис. 10. Маршрут, знайдений алгоритмом пошуку точок переходу

3. Порівняння алгоритмів

Для оцінювання було використано два типи лабіринтів розміром 255×255 клітинок: лабіринт, згенерований на основі бінарного дерева, та лабіринт, побудований за допомогою рандомізованого алгоритму Прима. Основним критерієм порівняння обрано час виконання алгоритмів при пошуку маршруту між випадково обраними парами точок.

3.1. Методика порівняння на лабіринтах

Експериментальне дослідження проводилося за уніфікованою процедурою, що забезпечує коректність порівняння алгоритмів. Для кожного типу лабіринту виконувалася наступна послідовність дій:

1. Генерація лабіринту відповідного типу.
2. Ініціалізація змінних для накопичення часу виконання кожного алгоритму.
3. Встановлення кількості ітерацій експерименту (1000).
4. Для кожної ітерації:
   - випадковим чином обиралися стартова та цільова вершини;
   - виконувалося знаходження маршруту за допомогою алгоритмів Дейкстри, A* та пошуку точок переходу (ПТП);
   - фіксувався час виконання кожного алгоритму з подальшим накопиченням.

Після завершення серії ітерацій обчислювався сумарний час роботи кожного алгоритму.

Такий підхід дозволяє мінімізувати вплив випадкових факторів і отримати узагальнену оцінку продуктивності.

3.2. Лабіринт на основі бінарного дерева

Перший експеримент було проведено на лабіринті, сформованому за алгоритмом бінарного дерева (рис. 11), який характеризується переважно лінійною структурою проходів та наявністю вираженого напрямку.

За результатами експерименту отримано такі значення сумарного часу виконання:

- алгоритм Дейкстри – 0:20:25.041311;
- алгоритм A* – 0:11:16.526667;
- алгоритм ПТП – 0:07:06.049409.

Отримані результати демонструють, що алгоритм Дейкстри має найнижчу продуктивність, що зумовлено повним обходом доступної області графа без урахування цільової вершини. Використання евристичної функції в алгоритмі A* дозволяє суттєво обмежити область пошуку, що забезпечує приблизно дворазове скорочення часу виконання.

Найкращий результат продемонстрував алгоритм пошуку точок переходу. Це пояснюється тим, що в умовах переважно прямолінійних проходів він ефективно усуває симетричні варіанти маршрутів і зменшує кількість розглянутих вершин.

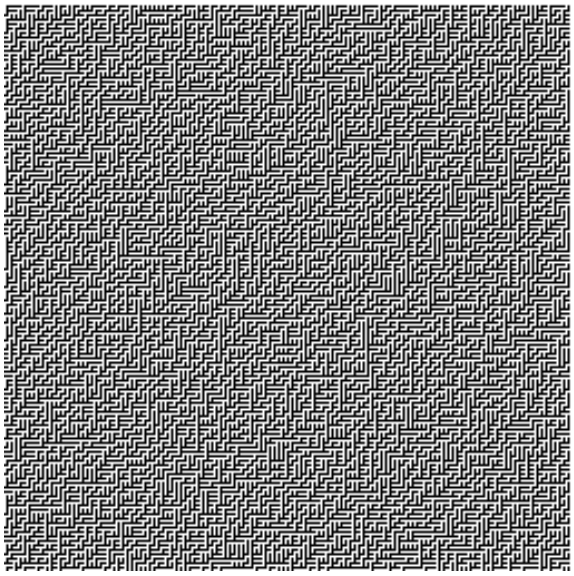

Рис. 11. Лабіринт на основі бінарного дерева

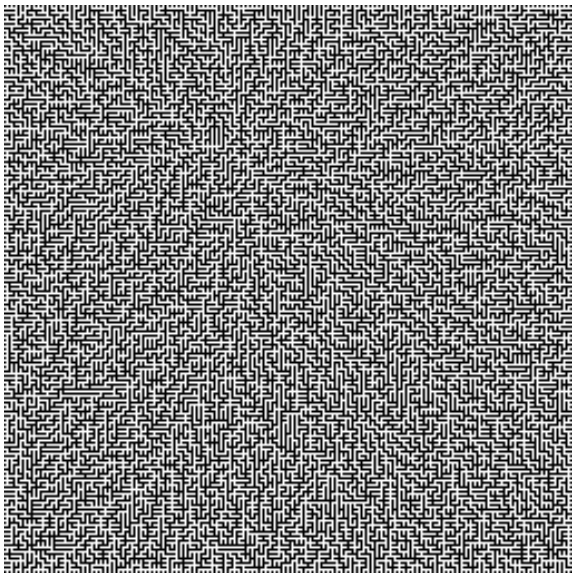

Рис. 12. Лабіринт на основі алгоритму Прима

3.3. Лабіринт на основі алгоритму Прима

Другий експеримент було проведено на лабіринті, згенерованому за допомогою алгоритму Прима (рис. 12), який відзначається більш складною топологією, наявністю численних перехресть і відсутністю домінуючого напрямку.

Результати вимірювань мають наступний вигляд:

- алгоритм Дейкстри – 0:20:13.978304;
- алгоритм A* – 0:11:02.435265;
- алгоритм ПТП – 0:04:35.152273.

Аналогічно до попереднього випадку, алгоритм Дейкстри продемонстрував найменшу ефективність, що підтверджує його обмеженість у задачах з великим простором пошуку. Алгоритм A* знову показав значно кращі результати завдяки використанню евристичної оцінки.

Водночас у даному типі лабіринту спостерігається ще більш виражена перевага алгоритму ПТП. Це можна пояснити тим, що складна структура середовища з великою кількістю перешкод створює умови, у яких механізми відсічення та виявлення вимушених сусідів працюють найбільш ефективно, суттєво скорочуючи кількість досліджуваних вершин.

Результати експерименту наведено в табл. 1.

Таблиця 1

Порівняння часу виконання алгоритмів на лабіринтах

Тип лабіринту	**Алгоритм Дейкстри**	**A***	**ПТП**
Бінарне дерево	0:20:25	0:11:16	0:07:06
Алгоритм Прима	0:20:13	0:11:02	0:04:35

Порівняльний аналіз показує, що продуктивність алгоритмів пошуку маршруту істотно залежить від структури середовища. Алгоритм Дейкстри демонструє стабільно низьку швидкодію через відсутність спрямованості пошуку. Алгоритм A* забезпечує значне покращення завдяки використанню евристики, проте його ефективність обмежується необхідністю розгляду значної кількості вершин у складних середовищах.

Алгоритм пошуку точок переходу виявився найбільш ефективним у всіх досліджених випадках. Його перевага зумовлена зменшенням симетрії пошуку та концентрацією обчислень лише на ключових вершинах. Отримані результати підтверджують доцільність використання цього алгоритму для задач пошуку маршруту на регулярних сітках з рівномірною вартістю переміщення.

4. Порівняння на мапах

Наступним етапом експериментального дослідження стало порівняння ефективності алгоритмів пошуку маршруту в середовищах, що моделюють більш реалістичні умови порівняно з лабіринтами. Для цього було використано чотири мапи з різною щільністю перешкод, побудовані на основі двох типів структур: лабіринту Прима та бінарного лабіринту. Для кожного типу розглядалися два варіанти: з меншою та більшою щільністю перешкод.

Метою дослідження є оцінка впливу структури середовища та ступеня його заповненості на продуктивність алгоритмів Дейкстри, A* та пошуку точок переходу (ПТП).

4.1. Методика експерименту

Методика проведення експерименту є аналогічною до описаної в підрозділі 3.1. Для кожної мапи виконувалася серія з 1000 ітерацій, у межах яких:

1) випадковим чином обиралися стартова та цільова вершини;
2) кожен алгоритм виконував пошук маршруту;
3) фіксувався час виконання з подальшим накопиченням.

Після завершення серії ітерацій обчислювався сумарний час роботи кожного алгоритму, що використовувався як інтегральна характеристика продуктивності.

4.2. Отримані результати

Результати експерименту для ***мапи з відносно низькою щільністю перешкод***, побудованої на основі алгоритму Прима (рис. 13), наведено нижче:

- алгоритм Дейкстри – 0:33:21.470127;
- алгоритм A* – 0:18:31.745551;
- алгоритм ПТП – 0:03:58.323529.

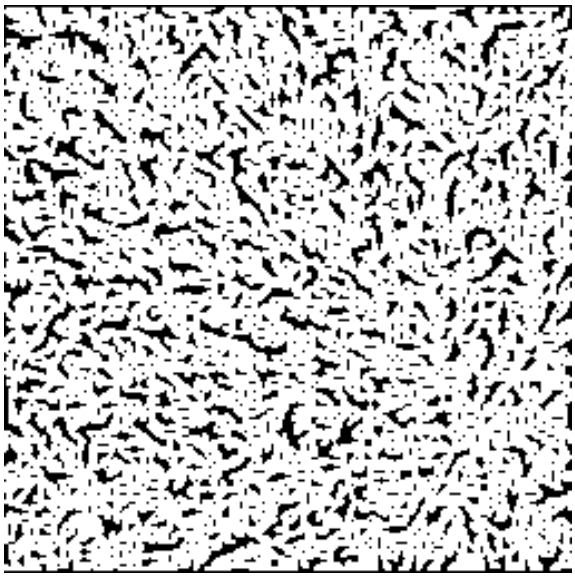

Рис. 13. Мапа на основі лабіринту Прима з меншою щільністю перешкод

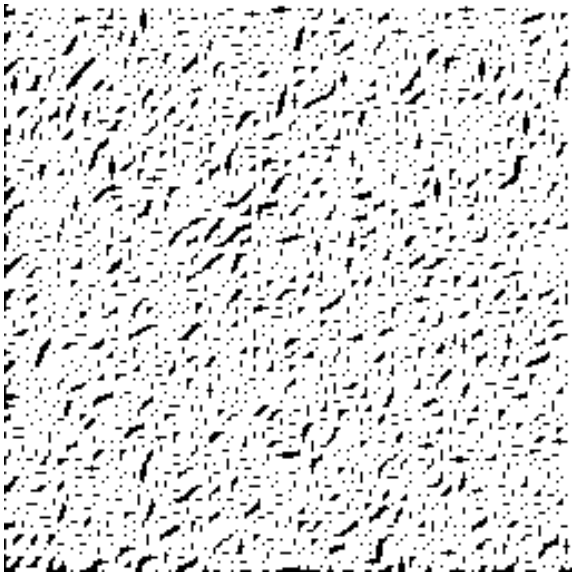

Рис. 14. Мапа на основі бінарного лабіринту з меншою щільністю перешкод

Отримані результати свідчать про суттєву перевагу алгоритму ПТП у середовищах із великою кількістю відкритих ділянок. У таких умовах алгоритм ефективно скорочує простір пошуку за рахунок руху між ключовими точками, ігноруючи проміжні вершини. Алгоритм Дейкстри, навпаки, демонструє найгірші результати через необхідність повного обходу доступної області.

Для ***мапи, сформованої на основі бінарного лабіринту з меншою щільністю перешкод*** (рис. 14), отримано такі результати:

- алгоритм Дейкстри – 0:37:19.253132;
- алгоритм A* – 0:21:08.208494;
- алгоритм ПТП – 0:04:17.588614.

Аналогічно до попереднього випадку, алгоритм ПТП демонструє найвищу ефективність. При цьому варто зазначити, що структура мапи, успадкована від бінарного лабіринту, зумовлює часткову впорядкованість перешкод, що дещо знижує ефективність евристичного спрямування алгоритму A* порівняно з більш хаотичними структурами.

Для ***мапи з підвищеною щільністю перешкод, побудованої на основі алгоритму Прима*** (рис. 15), отримано такі значення:

- алгоритм Дейкстри – 0:26:59.961727;
- алгоритм A* – 0:15:29.628627;
- алгоритм ПТП – 0:04:06.620957.

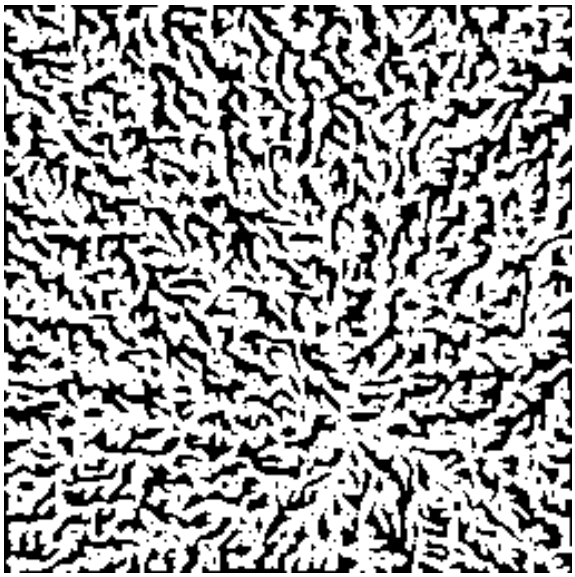

Рис. 15. Мапа на основі лабіринту Прима з більшою щільністю перешкод

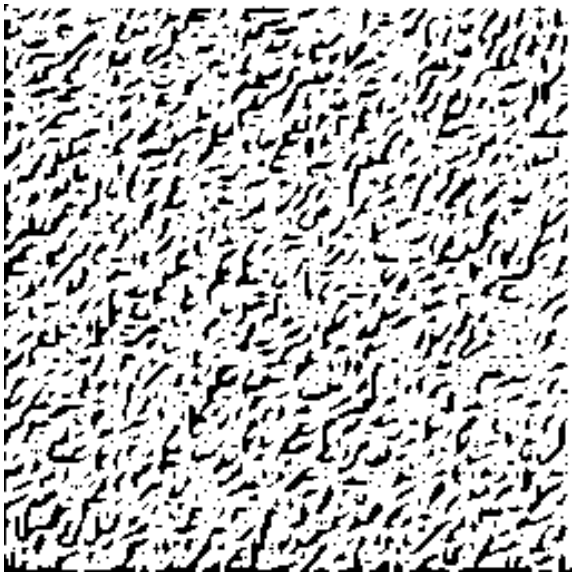

Рис. 16. Мапа на основі бінарного лабіринту з більшою щільністю перешкод

Збільшення щільності перешкод призводить до зменшення області доступного простору, що частково знижує обчислювальне навантаження для всіх алгоритмів. У цих умовах алгоритм A* демонструє помітне покращення ефективності порівняно з алгоритмом Дейкстри, оскільки евристична функція дозволяє більш точно спрямовувати пошук. Алгоритм ПТП зберігає найкращі показники завдяки ефективному скороченню кількості розглянутих вершин.

Останній експеримент проведено на ***мапі, побудованій на основі бінарного лабіринту з високою щільністю перешкод*** (рис. 16):

- алгоритм Дейкстри – 0:31:03.161842;
- алгоритм A* – 0:17:09.005322;
- алгоритм ПТП – 0:04:12.425184.

У цьому випадку спостерігається аналогічна тенденція: алгоритм ПТП демонструє найкращу продуктивність, тоді як алгоритм Дейкстри залишається найменш ефективним. Висока щільність перешкод обмежує варіативність маршрутів, що частково підвищує ефективність алгоритму A*, однак не дозволяє йому досягти рівня ПТП.

Аналіз результатів експерименту дозволяє зробити такі ***узагальнення***:

1) продуктивність алгоритмів суттєво залежить від структури середовища та щільності перешкод;
2) алгоритм Дейкстри демонструє найнижчу ефективність у всіх досліджених випадках через відсутність спрямованості пошуку;
3) алгоритм A* забезпечує стабільне покращення продуктивності завдяки використанню евристичної функції, причому його ефективність зростає зі збільшенням щільності перешкод;
4) алгоритм пошуку точок переходу є найбільш ефективним у всіх експериментальних сценаріях, особливо у середовищах із великими відкритими просторами.

Отримані результати підтверджують доцільність використання алгоритму ПТП для задач пошуку маршруту на регулярних сітках, а також демонструють важливість

врахування характеристик середовища при виборі алгоритму.

Таблиця 2

Порівняння сумарного часу виконання алгоритмів та відносного прискорення

Тип середовища	Дейкстра	A*	ПТП	Прискорення A* відносно Дейкстри	Прискорення ПТП відносно A*	Прискорення ПТП відносно Дейкстри
Бінарний лабіринт	20:25	11:16	7:06	44,8%	36,9%	65,2%
Лабіринт Прима	20:13	11:02	4:35	45,4%	58,5%	77,3%
Мапа (Прима, мало перешкод)	33:21	18:31	3:58	44,5%	78,6%	88,1%
Мапа (бінарна, мало перешкод)	37:19	21:08	4:17	43,4%	79,7%	88,5%
Мапа (Прима, багато перешкод)	26:59	15:29	4:06	42,6%	73,5%	84,8%
Мапа (бінарна, багато перешкод)	31:03	17:09	4:12	44,8%	75,5%	86,5%

Висновки

У роботі здійснено реалізацію та комплексне дослідження алгоритмів генерації лабіринтів і мап, а також алгоритмів пошуку маршрутів, зокрема алгоритму Дейкстри, алгоритму A* та алгоритму пошуку точок переходу (Jump Point Search, JPS). Проведене експериментальне дослідження дозволило оцінити їхню швидкодію в умовах різних типів середовищ, представлених у вигляді двовимірних сіток із різною структурою та щільністю перешкод.

На основі отриманих результатів можна сформулювати такі узагальнення:

1. Порівняльна ефективність алгоритмів.

Встановлено, що алгоритм A* демонструє стабільне покращення швидкодії порівняно з алгоритмом Дейкстри, забезпечуючи скорочення часу виконання в середньому на 42-47%. Це підтверджує доцільність використання евристичних підходів у задачах маршрутизації.

2. Переваги алгоритму пошуку точок переходу.

Алгоритм JPS показав найвищу ефективність серед розглянутих методів, забезпечуючи скорочення часу пошуку до 65–88% порівняно з алгоритмом Дейкстри та суттєво випереджаючи алгоритм A*. Найбільший виграш у швидкодії спостерігається у середовищах зі складною структурою та високою щільністю перешкод, що пояснюється зменшенням кількості симетричних обчислень.

3. Вплив структури середовища на продуктивність.

Експериментально підтверджено, що ефективність алгоритмів суттєво залежить від топології простору. Для середовищ із меншою щільністю перешкод різниця між алгоритмами зменшується, тоді як у більш складних середовищах переваги евристичних і оптимізованих підходів стають більш вираженими.

4. Рекомендації щодо застосування алгоритмів.

- алгоритм JPS доцільно використовувати у задачах із високими вимогами до швидкодії та частою зміною стартових і цільових позицій;
- алгоритм A* є ефективним компромісним рішенням у випадках, коли застосування JPS обмежене умовами задачі або структурою середовища;
- алгоритм Дейкстри доцільно застосовувати у задачах, що передбачають багаторазове використання результатів пошуку від однієї фіксованої вершини або за відсутності коректної евристичної функції.

5. Обмеження дослідження.

У роботі розглядалися виключно алгоритми, що забезпечують знаходження оптимального маршруту. Це зумовлює додаткові обчислювальні витрати, які можуть бути надмірними для задач, де допускається наближене розв'язання.

6. Практичне значення та перспективи подальших досліджень.

Отримані результати можуть бути використані при виборі алгоритмів пошуку маршруту в прикладних системах, зокрема у відеоіграх, робототехніці та навігаційних застосуваннях. Перспективним напрямом подальших досліджень є аналіз алгоритмів, що знаходять неоптимальні, але швидші маршрути, а також адаптація розглянутих методів до динамічних середовищ і тривимірних просторів.

Список використаної літератури

1. Severance, C. R. Python for Everybody: Exploring Data Using Python 3. CreateSpace Independent Publishing Platform, 2016. 244 p.
2. Miller, B. N., Ranum, D. L., and Anderson, J. Python Programming in Context. Jones & Bartlett Learning, 2019. 530 p.
3. Joshi, P. Artificial Intelligence with Python. Packt Publishing, 2017. 446 p.
4. Ni, Y.; Zhuo, Q.; Li, N.; Yu, K.; He, M.; Gao, X. Characteristics and Optimization Strategies of A* Algorithm and Ant Colony Optimization in Global Path Planning Algorithm. Int. J. Pattern Recognit. 2023, 37, 2351006.
5. Wang, P.; Liu, Y.; Yao, W.; Yu, Y. Improved A-star algorithm based on multivariate fusion heuristic function for autonomous driving path planning. Proc. Inst. Mech. Eng. Part D-J. Automob. Eng. 2022, 237, 1527-1542.
6. Zhang, Y.; Li, L.; Lin, H.; Ma, Z.; Zhao, J. Development of Path Planning Approach Using Improved A-star Algorithm in AGV System. J. Internet Technol. 2019, 20, 915-924.
7. Iram, N.; Amna, K.; Khurshid, A.; Zulfiqar, H. A Path-Planning Performance Comparison of RRT*-AB with MEA* in a 2-Dimensional Environment. Symmetry 2019, 11, 945.
8. Liu, L.; Lin, J.; Yao, J.; He, D.; Zheng, J.; Huang, J.; Shi, P. Path Planning for Smart Car Based on Dijkstra Algorithm and Dynamic Window Approach. Wirel. Commun. Mob. Comput. 2021, 2021, 356-368.
9. Banerjee, N.; Chakraborty, S.; Raman, V.; Satti, S.R. Space Efficient Linear Time Algorithms for BFS, DFS and Applications. Theor. Comput. Syst. 2018, 62, 1736-1762.
10. Lai, W.K.; Shieh, C.S.; Yang, C.P. A D2D Group Communication Scheme Using Bidirectional and InCremental A-Star Search to Configure Paths. Mathematics 2022, 10, 3321.
11. Wang, H.; Qi, X.; Lou, S.; Jing, J.; He, H.; Liu, W. An Efficient and Robust Improved A* Algorithm for Path Planning. Symmetry 2021, 13, 2213.
12. Chen Yifu, Lu Wei, Ding Haojie. Research on Optimization Strategy of Dijkstra Algorithm // Computer Technology and Development. – 2006. – Vol. 16, No. 9. – pp. 73-75.
13. Zhang Yonglong. Optimization of Dijkstra Optimal Path Algorithm // Journal of Nanchang Institute of Technology. – 2006. – Vol. 25, No. 3. – pp. 30-33.
14. Using Prim's Algorithm to Generate Continuous Cave Maps [Електронний ресурс] // Режим доступу: https://kairumagames.com/blog/cavetutorial

References:

1. Severance, C. R. Python for Everybody: Exploring Data Using Python 3. CreateSpace Independent Publishing Platform, 2016. 244 p.
2. Miller, B. N., Ranum, D. L., and Anderson, J. Python Programming in Context. Jones & Bartlett Learning, 2019. 530 p.
3. Joshi, P. Artificial Intelligence with Python. Packt Publishing, 2017. 446 p.

4. Ni, Y.; Zhuo, Q.; Li, N.; Yu, K.; He, M.; Gao, X. Characteristics and Optimization Strategies of A* Algorithm and Ant Colony Optimization in Global Path Planning Algorithm. Int. J. Pattern Recognit. 2023, 37, 2351006.
5. Wang, P.; Liu, Y.; Yao, W.; Yu, Y. Improved A-star algorithm based on multivariate fusion heuristic function for autonomous driving path planning. Proc. Inst. Mech. Eng. Part D-J. Automob. Eng. 2022, 237, 1527-1542.
6. Zhang, Y.; Li, L.; Lin, H.; Ma, Z.; Zhao, J. Development of Path Planning Approach Using Improved A-star Algorithm in AGV System. J. Internet Technol. 2019, 20, 915–924.
7. Iram, N.; Amna, K.; Khurshid, A.; Zulfiqar, H. A Path-Planning Performance Comparison of RRT*-AB with MEA* in a 2-Dimensional Environment. Symmetry 2019, 11, 945.
8. Liu, L.; Lin, J.; Yao, J.; He, D.; Zheng, J.; Huang, J.; Shi, P. Path Planning for Smart Car Based on Dijkstra Algorithm and Dynamic Window Approach. Wirel. Commun. Mob. Comput. 2021, 2021, 356–368.
9. Banerjee, N.; Chakraborty, S.; Raman, V.; Satti, S.R. Space Efficient Linear Time Algorithms for BFS, DFS and Applications. Theor. Comput. Syst. 2018, 62, 1736–1762.
10. Lai, W.K.; Shieh, C.S.; Yang, C.P. A D2D Group Communication Scheme Using Bidirectional and InCremental A-Star Search to Configure Paths. Mathematics 2022, 10, 3321.
11. Wang, H.; Qi, X.; Lou, S.; Jing, J.; He, H.; Liu, W. An Efficient and Robust Improved A* Algorithm for Path Planning. Symmetry 2021, 13, 2213.
12. Chen Yifu, Lu Wei, Ding Haojie. Research on Optimization Strategy of Dijkstra Algorithm // Computer Technology and Development. – 2006. – Vol. 16, No. 9. – pp. 73-75.
13. Zhang Yonglong. Optimization of Dijkstra Optimal Path Algorithm // Journal of Nanchang Institute of Technology. – 2006. – Vol. 25, No. 3. – pp. 30-33.
14. Using Prim's Algorithm to Generate Continuous Cave Maps // URL: https://kairumagames.com/blog/cavetutorial

HLADKA Liudmyla,
PhD in Physical and Mathematical Sciences, Associate Professor of the Department of Automation and Computer-Integrated Technologies, The Bohdan Khmelnytsky National University of Cherkasy, Ukraine
CHALYI Anton,
Software Developer, UniStar LLC, Smila, Ukraine

COMPARATIVE STUDY OF PATHFINDING ALGORITHM PERFORMANCE ON MAZE AND GRID-BASED ENVIRONMENTS

Summary. Introduction. *Pathfinding tasks on two-dimensional grid structures arise in a wide range of applied domains, including navigation systems, robotics, telecommunications, and video game development. The computational efficiency of pathfinding algorithms directly affects software performance, particularly under constrained resources and real-time processing requirements. Despite the availability of numerous well-known algorithms, their practical efficiency varies substantially depending on the structural properties of the search space, including obstacle topology, obstacle density, and graph connectivity. The selection of an appropriate algorithm for a given environment type remains an open practical problem.*

Purpose. *The purpose of this work is to conduct a comparative performance analysis of widely used pathfinding algorithms – Dijkstra's algorithm, the A* algorithm, and the Jump Point Search (JPS) algorithm – on two-dimensional grid environments of varying structure and obstacle density, and to formulate evidence-based recommendations for algorithm selection.*

Results. *The study was implemented in Python using the Pillow library for grid representation and BMP image generation. Two maze generation methods were employed: the Binary Tree algorithm, which produces mazes with predominantly linear corridor structures, and the randomized Prim's algorithm, which yields more complex, uniformly distributed topologies. Map environments of varying obstacle density were derived from both maze types through iterative dead-end removal and corridor expansion procedures, producing four distinct map configurations. Experimental evaluation was performed on 255×255 grids using 1000 randomized start–goal pairs per environment. Total accumulated execution time served as the primary performance metric. The results demonstrate that A* reduces execution time by 42-47% relative to Dijkstra's algorithm across all tested environments, owing to heuristic-guided search. The Jump Point Search algorithm achieved the highest performance in all scenarios, reducing total execution time by 65-88% relative to Dijkstra's algorithm and by 37-*

79% relative to A. The advantage of JPS was most pronounced on map environments with large open areas and low obstacle density, where symmetry pruning and jump point identification most effectively reduce the number of explored nodes. In high-density obstacle environments, the performance gain of JPS remained substantial, with A* also showing improved relative efficiency due to more focused heuristic guidance.*

Conclusion. *The experimental study confirms that heuristic and symmetry-based approaches substantially outperform classical graph traversal in grid pathfinding tasks. JPS is recommended for applications with high performance requirements and frequent re-planning between arbitrary node pairs. A* provides an effective compromise when JPS applicability is constrained by environmental conditions. Dijkstra's algorithm remains appropriate for scenarios requiring full shortest-path trees from a fixed source node or in the absence of a valid heuristic function. Future research directions include analysis of suboptimal but faster algorithms, extension to dynamic environments, and adaptation to three-dimensional search spaces.*

Keywords: *pathfinding algorithms; Dijkstra's algorithm; A* algorithm; Jump Point Search; heuristic search; two-dimensional grid; route optimization; maze generation; navigation systems; path planning.*

Одержано редакцією 07.06.2024 р.
Прийнято до публікації 28.08.2024 р.

УДК 004.932

DOI 10.31651/2076-5886-2024-1-21-33

PACS 07.05.Pj, 42.30.Va

ГАВРЮШЕНКО Анна Миколаївна
учитель математики та інформатики
Черкаської загальноосвітньої школи І-ІІІ ступенів № 32 Черкаської міської ради
e-mail: 91gavryushrenko@gmail.com
ORCID 0009-0000-0188-8505

БОРОЗДИХ Катерина Сергіївна
учениця Черкаської загальноосвітньої школи І-ІІІ ступенів № 32 Черкаської міської ради

ПОРІВНЯЛЬНИЙ АНАЛІЗ МЕТОДІВ ВИЯВЛЕННЯ КЛЮЧОВИХ ТОЧОК НА ЗОБРАЖЕННЯХ

У статті розглянуто математичне підґрунтя методу SIFT як одного з перших та найбільш теоретично обґрунтованих підходів до розв'язання зазначеної задачі, а також проведено порівняльний аналіз шести сучасних методів: SIFT, SURF, BRISK, ORB, KAZE та AKAZE. Дослідження виконане на чотирьох синтетичних зображеннях із різними геометричними властивостями та на двох реальних зображеннях. За результатами обчислювальних експериментів здійснено порівняння методів за кількістю виявлених ключових точок, якістю їх конфігурації та часом пошуку однієї ключової точки. Встановлено, що жоден із розглянутих методів не є абсолютним переможцем за сукупністю показників, а вибір оптимального методу суттєво залежить від характеристик зображення та вимог конкретної задачі.

Ключові слова: *ключові точки, дескриптор зображення, SIFT, SURF, BRISK, ORB, KAZE, AKAZE, комп'ютерний зір, масштабно-інваріантне перетворення.*

Вступ

Задача пошуку одного зображення на іншому лежить в основі багатьох сучасних систем комп'ютерного зору. Вона є невід'ємною частиною розпізнавання об'єктів і сцен, реконструкції тривимірних структур за серією знімків, побудови стереоскопічних