МЕТОДИ І АЛГОРИТМИ СУЧАСНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

УДК 519.16

Р. Базилевич, Р. Кутельмах
Національний університет "Львівська політехніка",
кафедра програмного забезпечення

ДЕКОМПОЗИЦІЙНІ АЛГОРИТМИ ДЛЯ РОЗВ'ЯЗУВАННЯ ЗАДАЧІ КОМІВОЯЖЕРА

Описано алгоритми декомпозиції задачі комівояжера, які забезпечують знаходження розв'язків задачі із малими часовими затратами. Вхідна множина точок ділиться на підмножини, що істотно зменшує розмірність задачі. Одержані розв'язки потребують подальшої оптимізації.

The decomposition algorithms of solving Traveling Salesman Problem, that provide finding solution in the small time, are described. The whole input area is partitioned into subareas that substantially decrease the problem size. Achieved solutions need futher optimization.

Вступ

Задача комівояжера – одна із базових задач комбінаторної оптимізації, що має широке прикладне застосування [1, 2]. Задача є широко дослідженою, але простота її формулювання поєднується із великою складністю розв'язання. Сьогодні стають особливо актуальними задачі великих розмірностей. Це саме стосується і задачі комівояжера – істотне зростання її розмірності (до мільйонів точок), а також наявність часткових задач зі специфічними властивостями. Це – динамічні транспортні задачі, особливістю яких є поява нових точок обслуговування під час реалізації заданого маршруту; системи за викликом (швидка допомога, кур'єрська, пожежна служби, таксі), системи з часовими вікнами, системи постачання та інші. Вони потребують розроблення спеціальних алгоритмів, які забезпечують отримання якісних результатів у режимі реального часу. У зв'язку з цим актуальним є розроблення ефективних декомпозиційних алгоритмів для задачі комівояжера, які б забезпечили отримання якісних розв'язків для задач великих розмірностей.

Існує небагато алгоритмів, що забезпечують одержання якісних розв'язків задачі комівояжера, особливо за малих часових затрат [3]. Для розв'язування задачі комівояжера алгоритм Ліна–Кернігана є одним з найефективніших [4, 5]. Його обчислювальна складність – $O(n^2)$. Одержані результати – у межах 1–3% від оптимального [3]. Впродовж останніх років було одержано нову версію алгоритму Ліна–Кернігана – алгоритм Ліна–Кернігана–Гельсгауна [6]. Він забезпечує оптимальний розв'язок задачі для 7397 точок із бібліотеки тестів для транспортних задач – TSPLIB[7]. Як показали результати тестування методів розв'язування задачі комівояжера DIMACS TSP Challenge [3], він є найточнішим евристичним алгоритмом [3,8]. Обчислювальна складність алгоритму – $O(n^{2,2})$.

Групою вчених [9–12] розроблено пакет програмного забезпечення для точного розв'язування задачі комівояжера – Concodre TSP Solver [13]. Він забезпечив одержання оптимальних розв'язків для усіх тестів із бібліотеки TSPLIB, включаючи задачу розмірністю 33810 точок. Обчислення задачі потребувало значних часових витрат.

1. Формулювання задачі

У класичному формулюванні задача комівояжера подається як граф $G=(V,E)$, де V – множина вершин графа, а E – множина його ребер. Вага (або довжина) $c_{i,j}$ кожного ребра $e_{i,j}\in E$ вважається заданою. Задача вважається симетричною, якщо $c_{i,j} = c_{j,i,}\ \forall i,j \in V$. Задачу вважають евклідовою, якщо $c_{i,j} + c_{j,k} \geq c_{i,k}\ \forall i,j,k \in V$. Необхідно знайти гамільтонів цикл мінімальної ваги, який є закритим циклом у графі, що включає усі вершини та передбачає відвідування кожної вершини лише один раз.

Розглядатиметься симетрична евклідова задача комівояжера, де заданими вважають множину N з n точок ($|N|=n$), які описані їхніми координатами (x_i, y_i). Необхідно найти маршрут S^*, що проходить по одному разу через кожну точку, довжина якого $L^*(S^*)$ є мінімальною:

$$L^*(S^*) = \Sigma_{ij}\ l_{ij}^* \rightarrow \min \Sigma_{ij}\ l_{ij}^*\ \ \forall\ l_{ij}^* \in l_{ij}',$$

де l_{ij}' – деяка з допустимих за заданими обмеженнями ділянка між двома суміжними точками i та j виділеного маршруту.

Запропоновані декомпозиційні алгоритми передбачають розділення вхідної множини точок на підмножини (N_0, N_1,..., $N_k \subset N$). Кожна підмножина $N_i \subset N$ описується у вигляді однієї умовної точки. Відстань між підмножинами визначається як довжина найкоротшого ребра, що з'єднує дві підмножини, або як відстань між умовними центрами підмножин. Формування підмножин здійснюється за допомогою декомпозиційних алгоритмів, що зменшує час обчислень. Задача з n точками замінюється задачею з k точками. Звичайно $k<<n$, що істотно спрощує задачу. Деякою вибраною базовою процедурою (алгоритмом) P_0 розв'язується класична задача комівояжера між підмножинами, яка визначає порядок їхнього обходу (макромаршрут). Маючи порядок обходу, визначаються граничні точки у підмножинах – точки, що з'єднують підмножини у макромаршруті. Наступним етапом є формування розв'язку, який складається з об'єднання часткових розв'язків задачі у кожній підмножині. Для задачі комівояжера з кластерним розподілом точок [14, 15] запропоновано декомпозиційні алгоритми, що розглядають кластери близько розміщених точок як підмножини, між якими проходить макромаршрут. Також запропоновано стратегії знаходження початкового маршруту та оптимізації [16–18].

Запропоновані декомпозиційні алгоритми передбачають такі етапи, як:

- поділ вхідної множини точок на підмножини (формування підмножин);
- створення макромаршруту;
- знаходження початкового розв'язку.

Розглянемо детальніше усі етапи та їхні особливості.

2. Поділ на підмножини

Етап формування підмножин передбачає розділення вхідної області точок на множини меншого розміру для зменшення розмірності задачі. Деякою базовою процедурою (алгоритмом) знаходять маршрут між утвореними підмножинами – макромаршрут. На етапі обчислення макромаршруту кожна підмножина розглядається як окрема точка.

2.1. Поділ за допомогою мінімальної сітки

Поділ вхідної області точок за допомогою мінімальної сітки – створення елементів неоднорідної геометричної форми, що являють собою групи взаємно близьких точок. Мінімальна сітка – множина найкоротших ребер, які належать кожній точці вхідної області.

Поділ за допомогою мінімальної сітки включає такі етапи:

- пошук множини найкоротших ребер до кожної точки (пошук мінімальної сітки);
- сортування ребер в мінімальній сітці у порядку зростання їхньої довжини;
- формування підмножин точок із утвореної мінімальної сітки (задається максимально допустима кількість точок у підмножині);

Пошук найкоротших ребер передбачає один параметр – кількість найближчих сусідів до кожної точки. Для ефективного пошуку найближчих сусідів застосовується декомпозиція Карпа [19]. За її допомогою при пошуку найближчих сусідів до кожної точки є змога розглядати не всі точки множини, а лише ті, які лежать у близькому околі. На рис. 1 показано множини найкоротших ребер – мінімальні сітки для різних розподілів точок (задана кількість найближчих сусідів – 5).

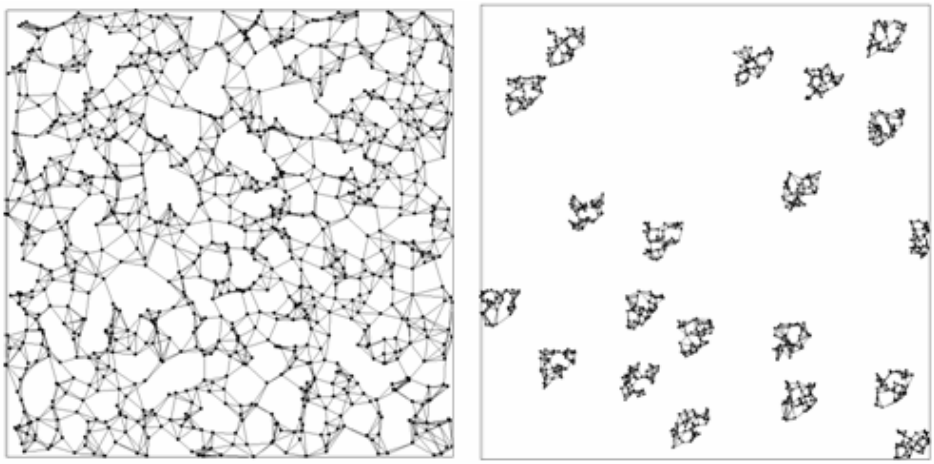

Рис. 1. Мінімальні сітки для довільного та кластерного розподілів точок

Наступним етапом є сортування ребер мінімальної сітки. Для задач великих розмірностей має велике значення, який саме вибрано метод сортування, тому рекомендовано сортувати ребра за допомогою алгоритмів, обчислювальна складність яких не більша за O($n$log(n)).

Після того, як мінімальну сітку знайдено, а ребра в ній відсортовано за зростанням довжини, настає етап формування підмножин – груп неоднорідної форми. Формування підмножин передбачає один параметр – максимально допустиму кількість точок у ній. Процес відбувається так: з мінімальної сітки вибирається ребро, і точки, що йому належать, утворюють підмножину, далі береться наступне ребро і точки, що йому належать, теж утворюють підмножину. Якщо вибране ребро містить точку, яка вже належить одній множині, то друга точка ребра теж належить до цієї підмножини. Якщо точки вибраного ребра вже належать різним підмножинам, то дві підмножини об’єднуються, проте лише тоді, коли кількість точок у новій підмножині не перевищить задану. Якщо точки вибраного ребра вже належать одній і тій самій підмножині, то таке ребро ігнорується.

Операція передбачає перегляд усіх ребер. Утворені підмножини міститимуть кількість точок, не більшу від заданої максимально допустимої. Це будуть елементи неоднорідної форми, структура та форма яких залежать від розподілу точок. Наприклад, в результаті поділу вхідної множини довільно розподілених точок отримується результат, показаний на рис. 2 (вхідна область – 1000 точок, утворено 14 підмножин, кожна з яких містить не більше ніж 100 точок):

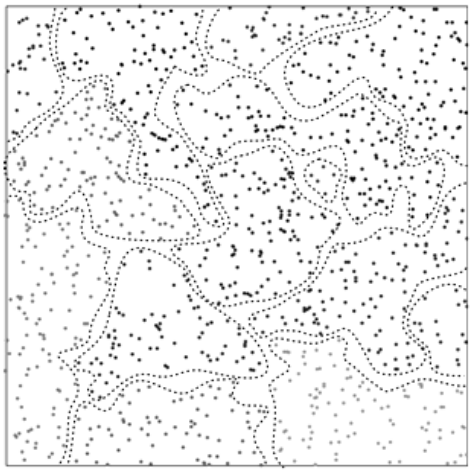

Рис. 2. Результат поділу за допомогою мінімальної сітки

При кластерному розподілі точок утворені підмножини матимуть вигляд, що показаний на рис. 3 (вхідна область – 1000 точок, утворено 20 підмножин, кожна з яких містить не більше ніж 100 точок).

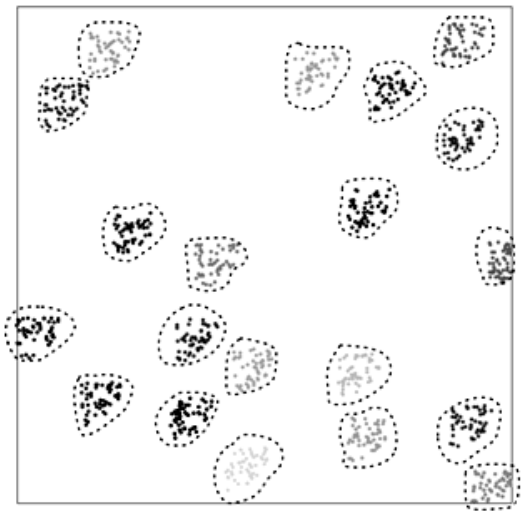

Рис. 3. Результат поділу за допомогою мінімальної сітки при кластерному розподілі точок

З рис. 3 видно, що утворені підмножини не об'єдналися, хоч містять набагато менше ніж 100 точок. Причина цього – мінімальна сітка не містила ребра, які б об'єднували різні кластери (рис. 1).

Описаний вище метод поділу вхідної області на підмножини забезпечує розділення задачі на підзадачі, зменшуючи розмірність.

Обчислювальна складність методу – O($n$log(n)) за умови, якщо ребра мінімальної сітки сортуються за час, менший або еквівалентний O($n$log(n)).

Сформовані підмножини використовуються на наступному етапі декомпозиційних алгоритмів – створенні макромаршруту.

3. Формування макромаршруту

Результат формування макромаршруту впливає на якість початкового розв'язання задачі, а отже, і на весь розв'язок задачі.

Знаходження макромаршруту – це розв'язок задачі комівояжера між підмножинами, що утворені на етапі поділу вхідної області точок. При знаходженні макромаршруту кожна підмножина розглядається як окрема точка. Для розв'язання цієї задачі спочатку необхідно визначити відстані між підмножинами. Запропоновано два підходи до визначення відстаней між підмножинами [16], згідно з якими відстанню між підмножинами є:

- відстань між умовними їхніми центрами (точкова математична модель);
- довжина найкоротшого ребра, що їх з'єднує (повна математична модель).

Після того, коли вибрано математичну модель підмножин, розв'язується класична задача комівояжера між підмножинами за допомогою добре відомої та ефективної базової процедури (алгоритму). У нашому випадку це може бути алгоритм *Ліна–Кернігана*, *Ліна–Кернігана–Гельсгауна*, *2-opt* чи *3-opt*.

4. Пошук початкового розв'язку

Знаходження початкового розв'язку задачі – етап декомпозиційних підходів, що передбачає розв'язання задачі для цілої вхідної області даних. Початковий розв'язок задачі, як правило, містить значно гіршу якість маршруту порівняно з оптимальним. Запропоновано спосіб побудови початкового маршруту на основі отриманого макромаршруту.

4.1 Знаходження початкового розв'язку на основі отриманого макромаршруту

Згідно з цим підходом початковий розв'язок задачі складається з об'єднання часткових розв'язків у підмножинах (сформованих за допомогою попередніх етапів). Для розв'язання задачі комівояжера у підмножинах використовується відомий базовий алгоритм, що був застосований для пошуку макромаршруту, проте також може бути використаний інший.

Для розв'язання задачі у кожній підмножині необхідно ідентифікувати так звані граничні точки підмножини, тобто точку, що буде відвідана першою з-поміж точок підмножини та точку, що буде відвідана останньою.

Обходячи послідовно усі підмножини, які є суміжними, та з'єднуючи їхні розв'язки, отримаємо повний маршрут. При побудові макромаршруту для повної математичної моделі початкова та кінцева точки відомі – це точки, що з'єднують дві суміжні підмножини. Для точкової математичної моделі граничні точки обчислюються додатково як взаємно найближчі точки між підмножинами.

Для прикладу знаходження початкового розв'язку розглянемо сформований макромаршрут при вибраній повній математичній моделі. Підмножини утворені за допомогою мінімальної сітки. Позначимо повну множину точок через N, кожну утворену підмножину через $N_i \in N$, початкову точку підмножини N_i – α_i, а кінцеву точку – β_i. $TSP(N_i)$ – розв'язок задачі у підмножині N_i, а $edge(\alpha_{i+1}, \beta_i)$ – ребро, що з'єднує кінцеву точку множини N_i із початковою точкою множини N_{i+1}.Початковий маршрут $TSP(N)$ складатиметься із об'єднання розв'язків задачі комівояжера у кожній підмножині та ребер, що з'єднують підмножини (містять граничні точки). Для заданих підмножин $N_0, N_1,\ldots, N_i$ початковим розв'язком буде:

$$TSP(N) = TSP(N_0)* edge(\alpha_1, \beta_0)* TSP(N_1)* edge(\alpha_2, \beta_1)*\ldots* TSP(N_i)* edge(\alpha_0, \beta_i)$$

На рис. 4 показано утворені підмножини, порядок їхнього обходу та граничні точки.

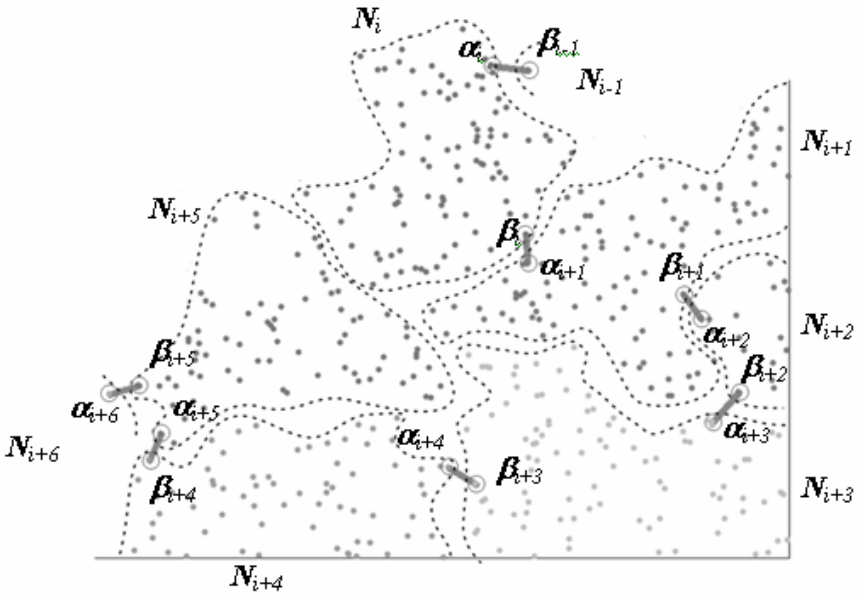

Рис. 4. Підмножини та їхні граничні точки

Розв'язання задачі комівояжера у кожній підмножині зводиться до розв'язання незамкнутої задачі комівояжера для точок, що містяться у підмножині. Незамкнута задача комівояжера – це тип задачі комівояжера, де відсутня умова повернення до початкової точки маршруту. Для знаходження розв'язку такої задачі застосовується довільний класичний базовий алгоритм, проте із дещо модифікованими вхідними даними. Модифікація являє собою штучне введення умовного ребра між

крайніми точками підмножини. Як вхідні дані для базового алгоритму подається множина точок підмножини та ребро між крайніми точками α_i та β_i, довжина якого – нескінченно мала величина. Описана вище модифікація являє собою метод “умовного ребра” [17,18].

За допомогою базового алгоритму розв’язуємо задачу комівояжера для точок, що входять у підмножину. Умовне ребро залишиться в результуючому маршруті і потрібно лише його вилучити. На рис. 5 показано підмножину з її граничними точками та результат розв’язання задачі комівояжера у ній.

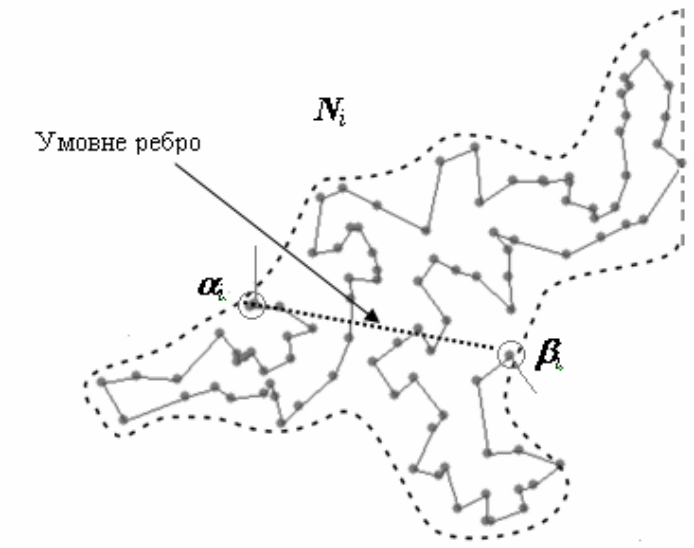

Рис. 5. Розв’язок задачі комівояжера у підмножині $\mathbf{N_i}$

Маючи розв’язки у кожній підмножині та макромаршрут, можна створити результуючий початковий маршрут. Для об’єднання розв’язків останню точку поточної підмножини з’єднуємо з першою точкою наступної і так для всього маршруту. На рис. 6 зображено об’єднання всіх розв’язків у підмножинах у єдиний початковий розв’язок задачі комівояжера.

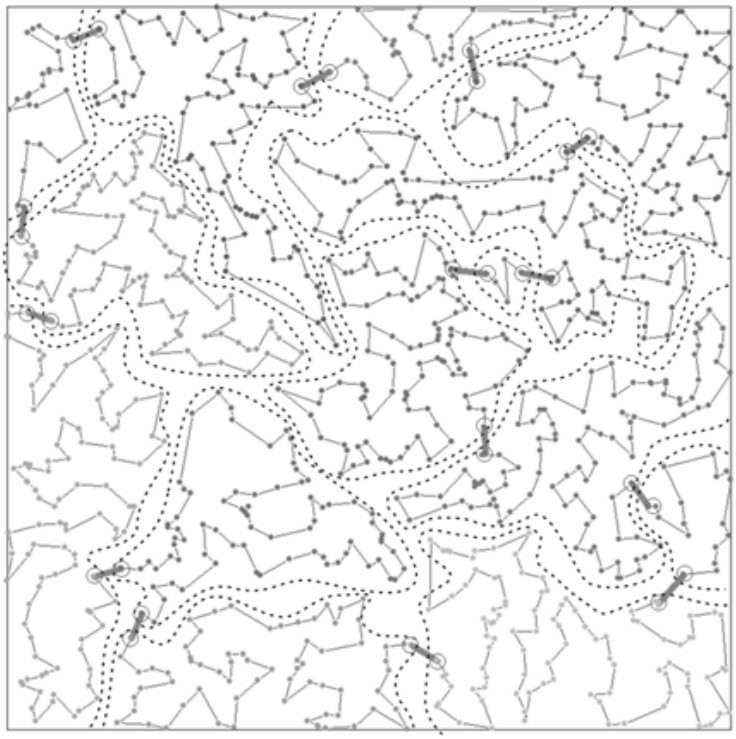

Рис. 6. Початковий розв’язок задачі комівояжера

Початковий розв'язок задачі, як правило, істотно гірший від оптимального. Результати експериментів показують, що залежно від вибраного базового алгоритму (*Ліна-Кернігана*, *2-opt* чи *3-opt*) на всіх стадіях обчислення початкового розв'язку довжина маршруту перевищує оптимальну на 2 – 20% (залежно від вибраного базового алгоритму та розмірності задачі). Для кластерного розподілу точок ця різниця в довжині ще більша. Основна причина цього – обмеження, накладені під час обчислення маршруту. Ними є:

- розв'язання задачі частково (в кожній підмножині окремо), а не усієї загалом;
- примусове встановлення в результуючому маршруті найкоротших ребер на стику підобластей.

Із рис. 6 видно, що існує багато ділянок із неоптимальними частинами маршруту. Ці ділянки виникають в результаті згаданих обмежень та є причиною істотного зростання довжини маршруту. На рис. 7 показано такі ділянки.

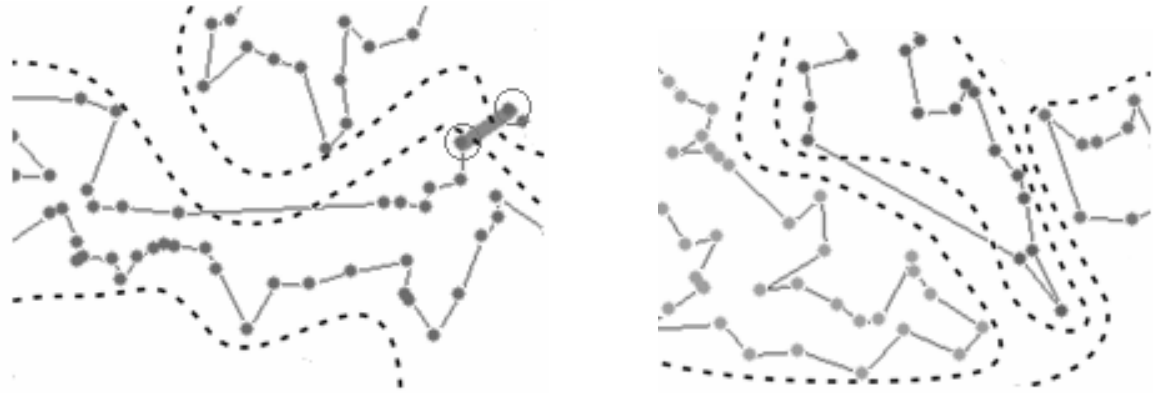

Рис. 7. Ділянки із поганою якістю проходження маршруту

Запропоновані методи знаходження початкового розв'язку забезпечують швидке формування початкового маршруту. Обчислювальна складність методів – лінійна, адже вони передбачають розв'язування задачі лише частинами. Довжина маршруту – на 2 – 20% більша від довжини оптимального. Для покращання якості отриманого початкового маршруту пропонується застосування різних методів оптимізації.

4. Експерименти

Здійснено тести для задач із різними розмірностями – 1000, 2000, 5000, 10000 та 20000 точок. В усіх тестах задавалися максимальні розміри підмножин – 50, 100, 150, 200, 300, 400. Поділ на підмножини здійснювався за допомогою мінімальної сітки. Задавалася різна кількість найближчих сусідів – 3, 5, 10. Розглядалася повна математична модель підмножин. Усі тести виконували на ПК з процесором Pentium IV – 3 МГц та об'ємом оперативної пам'яті 512 Мб. Як базові було взято два алгоритми: *2-opt* та алгоритм *Ліна–Кернігана–Гельсгауна* (взято із офіційного сайту Кельда Гельсгауна [20]).

Усі вхідні дані (множини точок) були згенеровані у випадковий спосіб.

Основні результати тестів наведені у таблиці.

Результати тестування задач розмірностями 1000–20000 точок

Розмірність задачі – 1000	
Базовий алгоритм – *2-opt*	
Час роботи базового алгоритму, секунди	1,83
Час роботи декомпозиційного алгоритму, секунди	1,39
Довжина маршруту відносно базового алгоритму	-0,08%
Базовий алгоритм – *LKH*	
Час роботи базового алгоритму, секунди	81,47
Час роботи декомпозиційного алгоритму, секунди	17,44
Довжина маршруту відносно базового алгоритму	1,35%
Розмірність задачі – 2000	
Базовий алгоритм – *2-opt*	
Час роботи базового алгоритму, секунди	7,88
Час роботи декомпозиційного алгоритму, секунди	2,98
Довжина маршруту відносно базового алгоритму	2,96%
Базовий алгоритм – *LKH*	
Час роботи базового алгоритму, секунди	198
Час роботи декомпозиційного алгоритму, секунди	16,4
Довжина маршруту відносно базового алгоритму	3,49%
Розмірність задачі – 5000	
Базовий алгоритм – *2-opt*	
Час роботи базового алгоритму, секунди	51
Час роботи декомпозиційного алгоритму, секунди	8,92
Довжина маршруту відносно базового алгоритму	2,68%
Базовий алгоритм – *LKH*	
Час роботи базового алгоритму, секунди	3829
Час роботи декомпозиційного алгоритму, секунди	100,7
Довжина маршруту відносно базового алгоритму	3,26%
Розмірність задачі – 10000	
Базовий алгоритм – *2-opt*	
Час роботи базового алгоритму, секунди	244
Час роботи декомпозиційного алгоритму, секунди	16,08
Довжина маршруту відносно базового алгоритму	3,36%
Базовий алгоритм – *LKH*	
Час роботи базового алгоритму, секунди	16646
Час роботи декомпозиційного алгоритму, секунди	135
Довжина маршруту відносно базового алгоритму	3,91%
Розмірність задачі – 20000	
Базовий алгоритм – *2-opt*	
Час роботи базового алгоритму, секунди	1112
Час роботи декомпозиційного алгоритму, секунди	25,7
Довжина маршруту відносно базового алгоритму	3,80%
Базовий алгоритм – *LKH*	
Час роботи базового алгоритму, секунди	124563
Час роботи декомпозиційного алгоритму, секунди	330
Довжина маршруту відносно базового алгоритму	3,97%

На рис. 8 показано графік залежності часу обчислень від розмірності задачі для алгоритму *Lin–Kernighan–Helsgaun* (без декомпозиції).

На наступних рисунках показано графіки залежностей часу обчислень від розмірності задачі для декомпозиційних алгоритмів та алгоритму *2-opt* (рис. 9 та рис. 10). У дужках вказано базовий алгоритм.

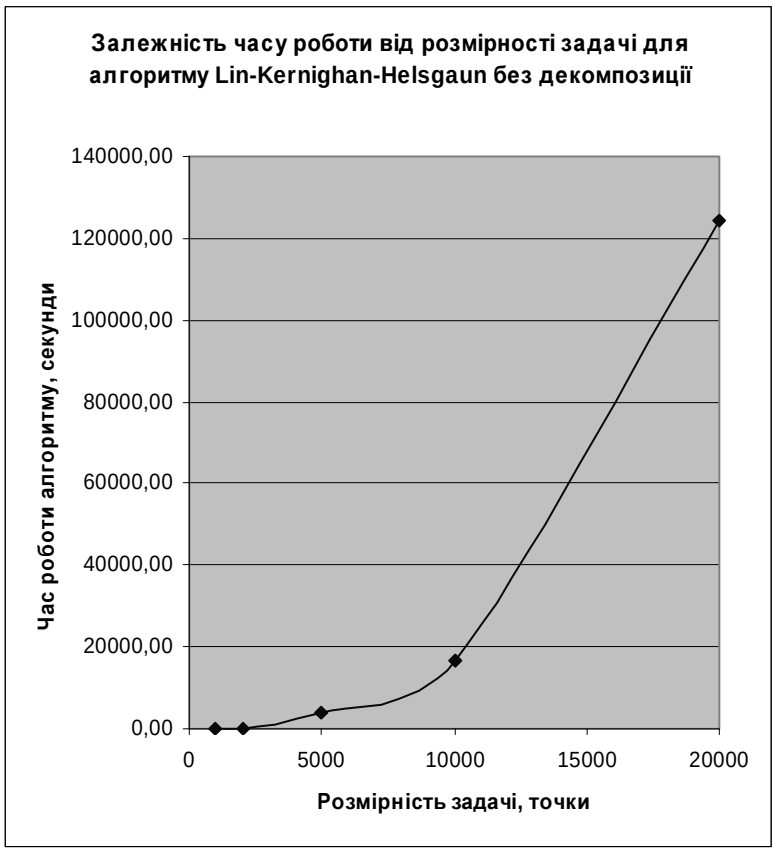

Рис. 8. Залежність часу обчислень від розмірності задачі для алгоритму Lin–Kernighan–Helsgaun

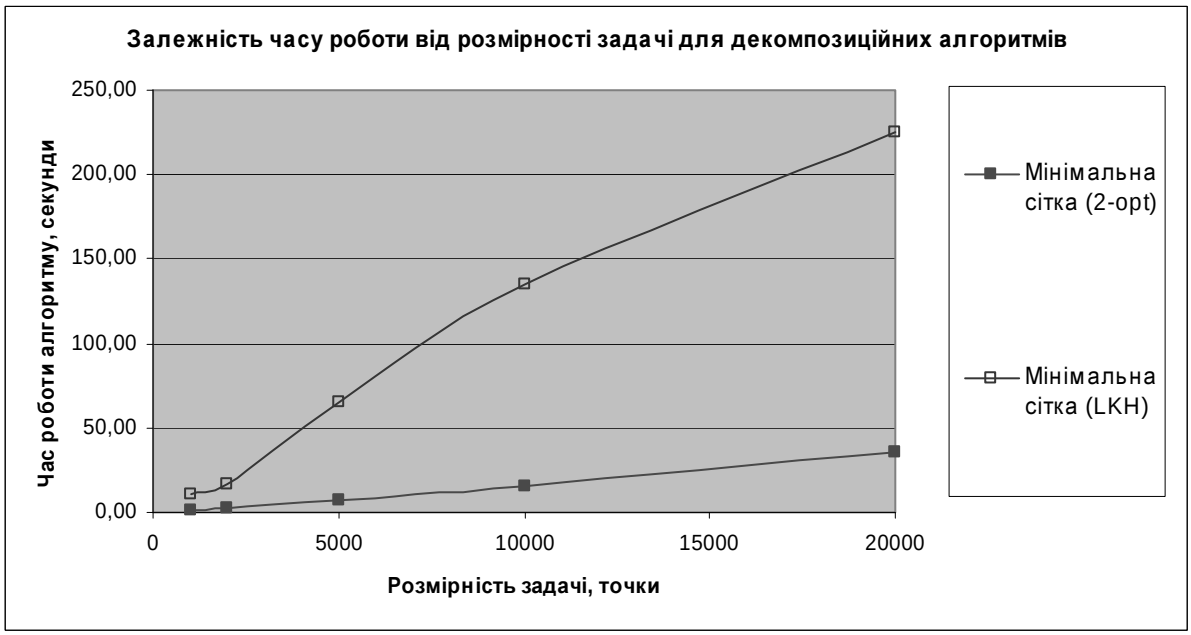

Рис. 9. Залежність часу обчислень від розмірності задачі для декомпозиційних алгоритмів

Рис. 10. Залежність часу обчислень від розмірності задачі для декомпозиційних алгоритмів разом із алгоритмом 2-opt

6. Висновки

Декомпозиційні алгоритми для швидкого знаходження розв'язків задачі комівояжера дають змогу істотно зменшити розмірність задачі за рахунок розділення вхідної множини точок на підмножини. Запропоновані підходи можуть застосовуватися для різного розподілу точок – рівномірного, кластерного чи комбінованого. Алгоритми передбачають три етапи – розділення на підмножини, визначення макромаршруту між підмножинами та знаходження розв'язків у кожній підмножині і їх об'єднання у єдиний розв'язок задачі. Як показали результати тестування, алгоритми виконуються за лінійний час, що вказує на доцільність використання запропонованих декомпозиційних алгоритмів для задачі комівояжера великих розмірностей – мільйонів точок.

При застосуванні декомпозиції, за рахунок виграшу в часі, втрачається якість розв'язку, яку забезпечує базовий алгоритм без декомпозиції. Результати експериментів показують, що втрати якості становлять 2–10% порівняно із базовим алгоритмом, незалежно від того, який алгоритм вибрано. Отже, одержані розв'язки потребують подальшої оптимізації. Найкраща якість забезпечується при розділенні вхідної множини на підмножини із великою кількістю точок – більшою ніж 200. Метод поділу вхідної множини за допомогою мінімальної сітки забезпечує дещо кращу якість одержаного розв'язку порівняно з простим поділом (найкращі результати досягаються при встановленому параметрі кількості найближчих сусідів – 5).

Для задачі розмірністю 10000 точок знаходження розв'язку виконується швидше у 100 разів порівняно з вибраним базовим алгоритмом Ліна–Кернігана–Гельсгауна, а для задачі розмірністю 20000 точок – приблизно у 400 раз швидше при погіршенні якості на 3–4%.

1. Reinelt, Gerhard, The Traveling Salesman: Computational Solutions for TSP Applications. Lecture Notes in Computer Science 840, Springer–Verlag, Berlin, 1994. 2. Reinelt, Gerhard, Fast heuristics for large geometric traveling salesman problems // ORSA Journal on computing, 4:206-217, 1992 3. David S. Johnson and Lyle A. McGeoch. Experimental Analysis of Heuristics for the STSP. In Gutin and Punnen, editors, The Traveling Salesman Problem and its Variations. Kluwer Academic Publishers, 2002. 4. Lin S. and Kernighan B.W. An effective heuristic algorithm for the Traveling salesman

problern. Operations Research, 21:498–516, 1973. 5. Lin S. Computer solutions of the travelling salesman problem. Bell System Technical Journal 44, pages 2245–2269, 1965. 6. Helsgaun K. "An effective implementation of the Lin–Kernighan Traveling Salesman Heuristic" European Journal of Operational Research,1:106-30, 2000. 7. Reinelt, Gerhard TSPLIB – A traveling salesman problem library, ORSA Journal on Computing 3, 376–384, 1991. 8. http://www.research.att.com/~dsj/chtsp/ 9. D. Applegate, R.E. Bixby, V. Chvátal, and W. Cook. On the solution of traveling salesman problems. Documenta Mathematica, Extra Volume ICM III:645–656, 1998. 10. D. Applegate, W. Cook, A. Rohe. Chained Lin–Kernighan for large traveling salesman problems. INFORMS J. Computing, to appear. 11. D. Applegate, R.E. Bixby, V. Chvátal, and W. Cook. Findong tours in the TSP.1998. 12. D. Applegate, R.E. Bixby, V. Chvátal, and W. Cook. Findong cuts in the TSP.1995 13. http://www.tsp.gatech.edu/concorde.html 14. D. Neto. Efficient cluster compensation for Lin–Kernighan Heuristics. PhD thesis, Department of Computer Science, University of Toronto, 1999. 15. G. Laporte, J-Y. Potvin, F. Quilleret. A. Tabu Search using Genetic Diversification for the Clustered Traveling Salesman Problem // Journal of Heuristics, Vol 2 (3), p. 187-200, 1996. 16. Базилевич Р.П., Кутельмах Р.К. Алгоритми динамічного формування моделі робочого поля для задачі комівояжера з кластерним розподілом точок // Вісник НУ "Львівська політехніка"№565, с.200-207, Львів, 2006. 17. Базилевич Р.П., Ремі Дюпа, Кутельмах Р.К., Використання алгоритмів локальної оптимізації для розв'язування задачі комівояжера з кластерним розподілом точок // Вісник НУ "Львівська політехніка" №565, с.207-212, Львів, 2006. 18. Bazylevych R., Dupas R, Kutelmakh R.. Scanning-area algorithms for clustered TSP // Proceedings of International conference "Comp. science and Information Technologies", Lviv, Polytechnic University, 2006, pp. 148-152. 19. Held, M., and Karp, R. M. The Traveling Salesman Problem and Minimum Spanning Trees // Operations Research. 18:1138–1162, 1970. 20. http://www.akira.ruc.dk/~keld/research/LKH /