Đại học Quốc gia Hà nội
Trường Đại học Khoa học Tự nhiên

Khoa Toán - Cơ - Tin học

Báo cáo thực tập

Thuật toán xấp xỉ đa thức cho bài toán TSP và Bin packing

Người hướng dẫn:	**TS. Đỗ Đức Hạnh**
Sinh viên thực tập:	**Phạm Hoàng Hải**
Mã sinh viên:	**20000548**
Thời gian thực tập:	**15/02/2024 - 20/05/2024**

Mục lục

Lời cảm ơn

Em xin cảm ơn các thầy cô khoa Toán - Cơ - Tin học đã tạo điều kiện cho em hoàn thành kỳ thực tập năm nay. Kỳ thực tập đã đem lại nhiều kinh nghiệm làm việc quý báu cho em, và sẽ là hành trang giúp em tiếp tục phát triển kỹ năng sau này.

Em xin cảm ơn TS. Đỗ Đức Hạnh và quý công ty Smartlog vì đã cho em cơ hội được học tập tại doanh nghiệp, giúp em học hỏi và phát triển bản thân.

Danh mục các từ viết tắt

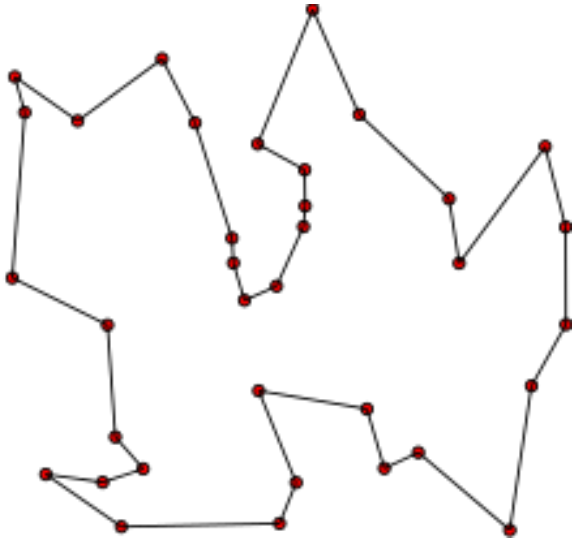
PTAS	Phương pháp xấp xỉ đa thức
TSP	Travelling Salesman Problem - Bài toán người giao hàng
E-TSP	Euclidean TSP - Bài toán TSP trong không gian Euclid
Arora PTAS	Lý thuyết xấp xỉ đa thức của S.Arora

Danh mục các định nghĩa

I. Giới thiệu

I.1. Bài toán TSP

Bài toán Người đưa hàng (Traveling Salesman Problem - TSP) là một trong những bài toán cổ điển và nổi tiếng nhất trong lĩnh vực tối ưu hóa và lý thuyết đồ thị. Bài toán được định nghĩa như sau: Cho một tập hợp các thành phố và khoảng cách giữa từng cặp thành phố, người đưa hàng cần tìm hành trình ngắn nhất để thăm mỗi thành phố đúng một lần và quay trở về thành phố ban đầu.



Hình 1: Một ví dụ về lời giải bài toán TSP. Các đỉnh đồ thị là các điểm giao hàng. Các cạnh đồ thị là đường đi.

TSP có ý nghĩa rất lớn trong nhiều lĩnh vực như logistics, vận tải, sản xuất và thậm chí cả trong các ứng dụng khoa học máy tính. Khả năng giải quyết TSP hiệu quả có thể dẫn đến tiết kiệm chi phí và thời gian đáng kể trong các ngành công nghiệp này. Ngoài ra, TSP cũng có vai trò quan trọng trong việc nghiên cứu và phát triển các thuật toán tối ưu hóa và tìm kiếm.

Tên gọi "Traveling Salesman Problem" lần đầu xuất hiện vào những năm 1930 khi nhà toán học người Ireland Hamilton và nhà toán học người Anh Kirkman nghiên cứu về các chu trình trong đồ thị. Tuy nhiên, bài toán chỉ thực sự được định nghĩa rõ ràng và trở nên phổ biến nhờ các nghiên cứu của các nhà toán học và kinh tế học trong thập kỷ 1950.

Với sự phát triển của công nghệ máy tính và các thuật toán mới, việc giải quyết bài toán TSP đã có những bước tiến vượt bậc.

I.1.1. Độ khó của bài toán

TSP là một bài toán NP-khó, có nghĩa là không có thuật toán nào có thể giải quyết tất cả các trường hợp của bài toán này trong thời gian đa thức, trừ khi P = NP. Điều này làm cho TSP trở thành một thách thức lớn đối với các nhà nghiên cứu và lập trình viên. Với số lượng thành phố tăng, số lượng các hành trình khả dĩ tăng lên theo cấp số nhân, khiến cho việc tìm kiếm giải pháp tối ưu trở nên rất khó khăn.

Vì vậy, việc tìm ra một phương án giải gần đúng đủ tốt là quan trọng hơn nhiều so với việc tìm ra một lời giải tối ưu.

I.1.2. Các phương pháp giải quyết TSP

Bài toán TSP thường được giải bằng một trong các phương pháp cổ điển như sau:

Phương phép Vét cạn (Brute Force): Phương pháp này thử tất cả các hành trình khả thi và chọn ra hành trình ngắn nhất. Tuy nhiên, phương pháp này chỉ khả thi cho các tập hợp thành phố rất nhỏ do độ phức tạp thời gian tăng theo hàm giai thừa của số thành phố.

Thuật toán Heuristic: Các thuật toán như Greedy, Nearest Neighbor, và Christofides cho phép tìm ra các giải pháp gần đúng trong thời gian hợp lý. Mặc dù các giải pháp này không đảm bảo tối ưu, nhưng chúng thường đem lại lời giải đủ tốt.

Thuật toán Metaheuristic: Các thuật toán như Simulated Annealing, Genetic Algorithms, và Ant Colony Optimization giúp tìm kiếm các giải pháp gần tối ưu bằng cách khám phá không gian nghiệm bài toán một cách thông minh và có tổ chức.

Phương pháp Quy hoạch động và Quy hoạch tuyến tính: Những phương pháp này sử dụng các kỹ thuật toán học để tìm ra giải pháp tối ưu cho TSP. Tuy nhiên, chúng thường chỉ khả thi cho các bài toán có kích thước vừa phải.

I.1.3. Ứng dụng thực tiễn của TSP

Logistics và Vận Tải: TSP được sử dụng để tối ưu hóa lộ trình cho các phương tiện giao hàng, giúp giảm chi phí nhiên liệu và thời gian di chuyển.

Sản Xuất: Trong các dây chuyền sản xuất, TSP giúp tối ưu hóa thứ tự di chuyển của các công cụ và nguyên vật liệu, cải thiện hiệu suất và giảm thời gian chờ.

Du Lịch và Lữ Hành: TSP hỗ trợ lập kế hoạch hành trình du lịch hiệu quả, giúp khách du lịch tham quan nhiều địa điểm trong thời gian ngắn nhất có thể.

I.2. Bài toán Bin Packing

Bài toán Bin Packing (còn gọi là Bin Loading) là một trong những bài toán tối ưu hóa cổ điển trong khoa học máy tính và toán học. Bài toán được định nghĩa như sau: Cho một tập hợp các đối tượng có kích thước khác nhau và một số lượng giới hạn các thùng (bins), mỗi thùng có một dung tích cố định. Nhiệm vụ là sắp xếp các đối tượng vào các thùng sao cho số lượng thùng sử dụng là ít nhất, đồng thời đảm bảo rằng tổng kích thước của các đối tượng trong mỗi thùng không vượt quá dung tích của thùng đó.

I.2.1. Ứng dụng thực tế

Bài toán Bin Packing xuất hiện trong nhiều tình huống thực tế như:

- Quản lý kho bãi: Sắp xếp hàng hóa vào các kho chứa sao cho sử dụng không gian hiệu quả.
- Vận tải và logistics: Phân chia hàng hóa vào các container vận chuyển sao cho tối thiểu số lượng container sử dụng.

- Quản lý bộ nhớ: Phân bổ các quy trình vào các khối bộ nhớ trong máy tính sao cho tối ưu dung lượng sử dụng.

I.3. Mục tiêu đề tài

Mục tiêu của đề tài này là nghiên cứu và áp dụng các phương pháp xấp xỉ đa thức (Polynomial Time Approximation Scheme - PTAS) vào giải quyết bài toán TSP và Bin packing, nhằm mang lại một hướng tiếp cận khác so với các phương pháp cổ điển như Heuristic và Metaheuristic. Cụ thể, đề tài sẽ dựa trên nền tảng lý thuyết của các nhà nghiên cứu S.Arora, S.B.Rao và V.V.Vazirani, những người đã có nhiều đóng góp quan trọng trong việc phát triển lý thuyết thuật toán xấp xỉ.

- **Khám Phá Lý Thuyết PTAS**: Nghiên cứu và hiểu rõ lý thuyết PTAS, đặc biệt là các công trình của Arora, Rao và Vazirani. PTAS cung cấp các giải pháp gần đúng với sai số $(1 + \varepsilon)$ tuỳ ý, và có thể chạy trong thời gian đa thức, làm cho nó trở thành một công cụ mạnh mẽ cho các bài toán tối ưu hóa tổ hợp phức tạp.
- **Phát Triển Thuật Toán Cụ Thể**: Dựa trên nền tảng lý thuyết đã nghiên cứu, phát triển và triển khai một thuật toán PTAS cụ thể cho bài toán TSP và Bin packing.
- **Đánh Giá Hiệu Năng**: Khảo sát thời gian chạy của các thuật toán PTAS đã phát triển. Đánh giá tính khả thi của việc áp dụng thuật toán trong các tình huống thực tế, như quản lý logistics, phân phối hàng hóa, và các dịch vụ giao hàng.
- **Đề Xuất Hướng Phát Triển Mới**: Dựa trên các kết quả đạt được, đề xuất những hướng nghiên cứu và phát triển mới cho việc áp dụng PTAS vào các bài toán tối ưu hóa tổ hợp khác. Khuyến khích việc tiếp tục nghiên cứu và cải tiến các phương pháp xấp xỉ trong lĩnh vực này.

Với các mục tiêu này, đề tài hy vọng sẽ mở ra một hướng đi mới trong nghiên cứu và ứng dụng các thuật toán xấp xỉ cho các bài toán NP-hard khác. Việc áp dụng PTAS không chỉ giúp ổn định sai số bài toán mà còn góp phần vào sự phát triển của lý thuyết thuật toán xấp xỉ và ứng dụng của chúng trong các bài toán thực tế phức tạp.

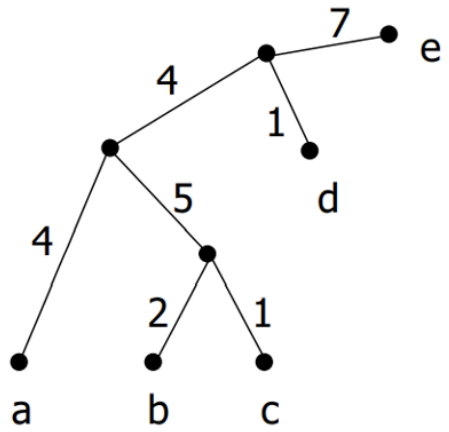
II. Nền tảng lý thuyết

II.1. Mô hình các bài toán

II.1.1. Mô hình hóa bản đồ

Trong thực tế, dữ liệu bản đồ sẽ được cho dưới dạng ma trận khoảng cách. Vì vậy, ở bước đầu tiên, ta mô hình hóa bản đồ cần xét dưới dạng đồ thị.

M	a	b	c	d	e
a	0	11	10	9	15
b	11	0	3	12	18
c	10	3	0	11	17
d	9	12	11	0	8
e	15	18	17	8	0



Hình 2: Bên trái: Ma trận khoảng cách. Bên phải: Đồ thị tương ứng với ma trận khoảng cách.

Đặt $G^0 = (V, E)$ là đồ thị có trọng số, mô tả mạng lưới giao thông trên bản đồ đang xét. Trong đó:

- $V = V_1 \cup V_2 \cup V_3 = \{v_i \in V \mid i \in \mathbb{N}^*\}$ trong đó:
  - V_1 là các ngã rẽ (ngã ba, ngã tư,...)
  - V_2 là các điểm cần giao hàng.
  - V_3 là kho hàng.
- $E = \{e_{ij} = (v_i, v_j) \mid v_i, v_j \in V; \quad i, j \in \mathbb{N}; \quad i \neq j\}$ là các đường có thể đi trên mạng lưới giao thông.
  - Để đơn giản bài toán, ta sẽ mặc định $e_{ij} = e_{ji} \quad \forall i, j \in \mathbb{N}^*$.

Từ đó, ta định nghĩa một đường đi $P \subset E$ trên đồ thị G^0 sẽ là một tập cạnh nối nhau trên G^0.

$$P = \{(e_{ij}, e_{jk}) \mid i, j, k \in \mathbb{N}^*\}$$

Ký hiệu:
- $w(e_{ij})$ là độ dài cạnh $e_{ij} \in E$
- $d(v_i, v_j)$ là độ dài đường đi ngắn nhất giữa đỉnh v_i và đỉnh v_j

Ký hiệu độ dài đường đi $P \subset E$ là:

$$d(P) = \sum_{e_{ij} \in P} w(e_{ij}) \qquad \text{với } e_{ij} \in P$$

II.1.2. Mô hình bài toán E-TSP

Trong bài toán TSP tổng quát, nhiệm vụ của ta là phải tìm đường đi trên một đồ thị $G = (V, E)$ tổng quát. Đây là một bài toán rất khó do gặp vấn đề bùng nổ tổ hợp.

Tuy nhiên, khi ta đã nhúng đồ thị $G = (V, E)$ vào không gian Euclid, các nghiên cứu của S.Arora đã chứng minh rằng, ta có thể tận dụng các cấu trúc hình học của không gian này để giải xấp xỉ bài toán trong thời gian đa thức. Bài toán TSP khi đó trở thành bài toán TSP trong không gian Euclid (Euclidean Travelling Salesman Problem, hay E-TSP).

Sau đây, ta sẽ phát biểu bài toán E-TSP.

Cho một tập X^0 các điểm cần giao hàng trong không gian Euclid:

$$X^0 = \{x_1, ..., x_n\} = \{x_i \in \mathbb{R}^d \mid i \in \mathbb{N}^*, i \leq n\}$$

và một điểm bắt đầu $x_0 = (a_1, ..., a_d) \in \mathbb{R}^d$.

Mục tiêu bài toán: tìm một đường đi $P = \big\{(x_0, x_i), \big(x_i, x_j\big), ..., (x_k, x_0)\big\}$ sao cho tổng độ dài đường đi $d(P) = \sum_{(x_i, x_j) \in P} \big(d\big(x_i, x_j\big)\big)$ là bé nhất.

II.1.3. Mô hình bài toán Bin packing 2D

Bài toán Bin packing có thể được phát biểu như sau:

Cho các thùng hàng hình chữ nhật (còn gọi là bin) có kích thước (W, H) cố định, và một tập I các đồ vật kích cỡ (w_i, h_i). Tìm các vị trí (x_i, y_i) của mỗi đồ vật sao cho:

- Không có đồ vật nào bị chồng lên nhau.
- Số lượng bin có chứa đồ là tối thiểu.

II.2. Phương án giải quyết

II.2.1. Phương pháp xấp xỉ đa thức (PTAS)

Trong khoa học máy tính (đặc biệt là lý thuyết thuật toán), phương pháp xấp xỉ đa thức (PTAS) là các họ thuật toán xấp xỉ cho các bài toán tối ưu hóa (thường là các bài toán tối ưu hóa NP-hard).

Một PTAS là một họ các thuật toán để giải một bài toán nhất định, trong đó mỗi thuật toán phụ thuộc tham số $\varepsilon > 0$ cho trước. Đầu ra thuật toán là một lời giải có kết quả trong khoảng $(1 + \varepsilon)$ so với lời giải tối ưu (hoặc $(1 - \varepsilon)$ đối với các bài toán tối đa hóa). Ví dụ, đối với bài toán người đưa hàng trong không gian Euclid, PTAS sẽ tạo ra một lịch trình di chuyển có độ dài tối đa là $((1 + \varepsilon) \cdot L)$, với L là độ dài của lịch trình di chuyển ngắn nhất.

II.2.1.1. Định nghĩa PTAS

Sau đây, ta sẽ đưa ra định nghĩa cho một PTAS.

Định nghĩa - Phương pháp xấp xỉ: Cho Π là một bài toán tối ưu hóa NP-hard, với hàm mục tiêu f_Π. Ta nói $\mathcal{A}$ là một phương pháp xấp xỉ cho bài toán Π nếu như với I là một đầu vào của Π và $\varepsilon > 0$ là tham số, ta có $\mathcal{A}(I, \varepsilon) = s$ thỏa mãn:

- $f_{\Pi}(I, s) \leq (1 + \varepsilon) \cdot \text{OPT}$ nếu Π là bài toán cực tiểu hóa.
- $f_{\Pi}(I, s) \geq (1 - \varepsilon) \cdot \text{OPT}$ nếu Π là bài toán cực đại hóa.

Trong đó, OPT là giá trị của hàm mục tiêu đối với nghiệm tối ưu.

Tham số ε còn thường được gọi là sai số của phương pháp xấp xỉ $\mathcal{A}$

Định nghĩa - Phương pháp xấp xỉ đa thức: Cho $\mathcal{A}$ là một phương pháp xấp xỉ của bài toán Π. Ta gọi $\mathcal{A}$ là một phương pháp xấp xỉ thời gian đa thức (PTAS) của Π nếu với tham số $\varepsilon > 0$, thời gian chạy của $\mathcal{A}$ là một đa thức theo I.

II.2.1.2. Lý do sử dụng PTAS

PTAS là một công cụ mạnh mẽ khi cần giải xấp xỉ các bài toán NP-khó.

Các khác biệt quan trọng của PTAS so với các phương pháp cổ điển như Heuristics và Metaheuristics bao gồm:

- **Độ chính xác:** Ta có thể chọn một ε bé tùy ý, vậy nên luôn biết trước sai số. Đây là kết quả tốt nhất mà ta có thể mong đợi cho các bài toán NP-hard, vốn không thể tìm lời giải tối ưu trong thời gian đa thức.
- **Tốc độ:** Thuật toán luôn chạy trong thời gian đa thức với mọi ε cố định

Tuy nhiên, định nghĩa của PTAS không xét đến thời gian chạy khi sai số ε tiến dần về 0. Vì vậy, một thuật toán chạy trong thời gian $O\left(n^{\frac{1}{\varepsilon}}\right)$ hoặc thậm chí $O\left(n^{e^{\frac{1}{\varepsilon}}}\right)$ vẫn được coi là một PTAS. Vậy nên, với một số PTAS, thời gian chạy có thể trở nên phi thực tế khi ε quá nhỏ.

PTAS có thể được áp dụng cho các bài toán tổ hợp NP-hard như bài toán xếp ba lô (knapsack problem), bài toán lập lịch (scheduling problem), và nhiều bài toán tối ưu khác.

II.2.2. PTAS của S.Arora cho bài toán E-TSP

S.Arora (Princeton University) đã phát minh ra PTAS cho bài toán E-TSP trong thời gian $O\big(n\log^k(n)\big)$ với sai số $(1 + \varepsilon)$, với k tỷ lệ với $\frac{1}{\varepsilon}$ và d khi đồ thị được nhúng sẵn. Nghiên cứu này của ông được trao giải thưởng Godel năm 2010.

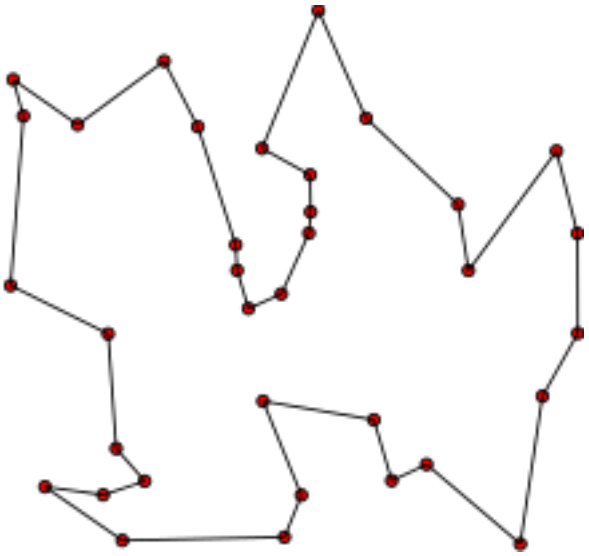
PTAS của S.Arora cho bài toán E-TSP là một đột phá quan trọng trong lĩnh vực thuật toán xấp xỉ, và đã đặt nền móng cho nhiều thuật toán xấp xỉ cho các bài toán NP-hard khác, chẳng hạn như Cây bao trùm tối thiểu, Cây Steiner tối thiểu, Ghép cặp hoàn hảo trọng số cực tiểu,....

Từ đây, ta sẽ gọi lý thuyết về PTAS của Arora là Arora PTAS.

II.2.2.1.a. Ví dụ thúc đẩy

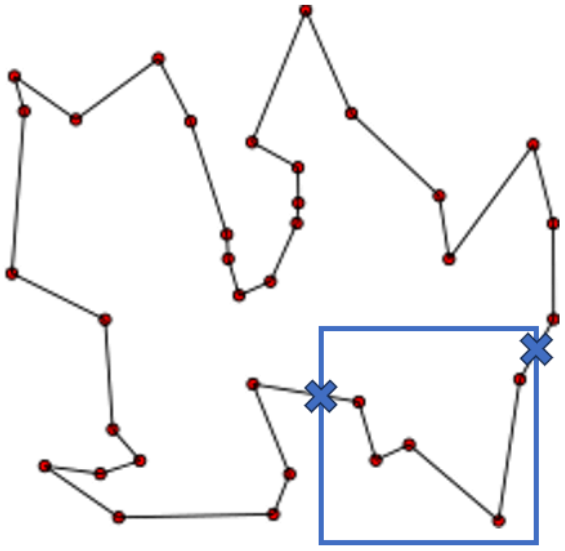
Để mô tả ý tưởng của Arora PTAS, đầu tiên, ta xét một ví dụ cụ thể.

Giả sử ta có bài toán E-TSP, trong đó ta đã biết lịch trình tối ưu của bài toán này là như sau:



Hình 3: Một ví dụ về lịch trình tối ưu của TSP

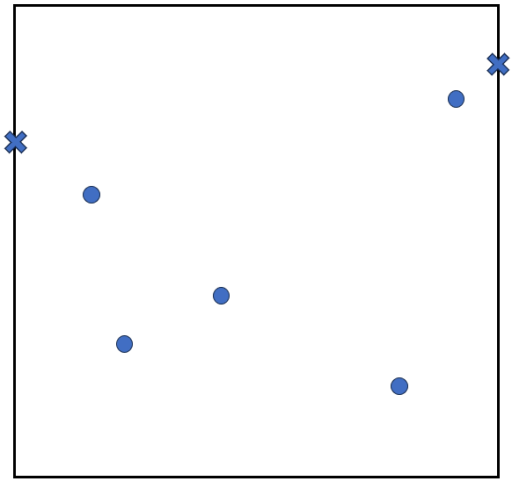
Giờ, ta lấy một hình vuông bất kỳ trong lịch trình tối ưu.



Hình 4:

Từ đây, ta có thể tách các điểm trong hình vuông thành một bài toán con. Trong lý thuyết xấp xỉ đa thức của S.Arora, đây được gọi là bài toán multipath.

Mục tiêu bài toán là tìm một tập các đường đi đi qua tất cả các điểm trên hình vuông và mọi điểm bên trong hình vuông.



Hình 5: Bài toán con cần giải quyết.

Phát biểu bài toán multipath: Đầu vào của bài toán bao gồm:

- Một hình vuông S.
- Tập điểm $V = \{v_i \in \partial S \mid i < r\}$, $r \in N^*$ gồm các điểm nằm trên biên của S.
- Một tập các điểm $X_S \subseteq X$ cần giao hàng trong hình vuông S.

Đầu ra bài toán: các đường đi $P = \big\{(v_a, x_i), ..., (x_j, v_b)\big\}$, sao cho $x \in P \ \ \forall x \in X_S$

Bài toán con này có thể tiếp tục được chia nhỏ. Ta có thể chia bài toán cho đến trường hợp cơ bản: khi chỉ có ≤ 1 điểm nằm trong S. Khi đó bài toán con có thể được giải bằng vét cạn trong thời gian $O(r) = O(\frac{1}{\varepsilon})$.

Ý tưởng chính của Arora chính là chia bài toán E-TSP thành nhiều bài toán con ở trường hợp cơ bản ($|X_S| \leq 1$), rồi giải và tổng hợp kết quả.

II.2.2.1.b. Các bước thực hiện

Bước 1: Tiền xử lý dữ liệu tọa độ

Việc nhân mọi tọa độ $x \in X^0$ với một hằng số không làm thay đổi bản chất bài toán. Vì vậy, ta lấy $X' = \{x' = kx \mid x \in X^0\}$ và $y' = ky_0$

Tiếp theo, ta làm tròn tọa độ của mọi điểm $x' \in X'$ và điểm y. Mục tiêu là gộp các điểm quá gần nhau và coi chúng như 1 điểm. Từ đó ta có dữ liệu mới:

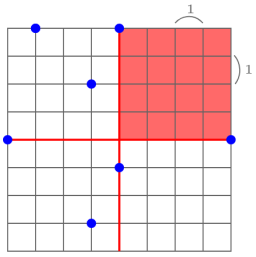
$$X = \{x = \text{round}(x') \mid x' \in X'\}$$

$$y = \text{round}(y')$$

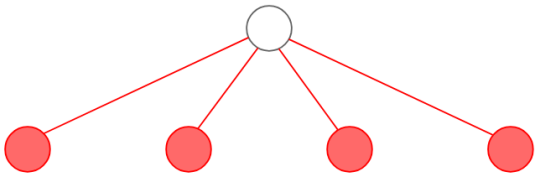
Bước 2: Chia nhỏ bài toán và thiết lập 4-tree

Các bước chia của bài toán có thể được mô tả bằng một 4-tree. Đầu tiên, ta lấy một hình vuông bất kỳ bao quanh mọi điểm với chiều dài L.

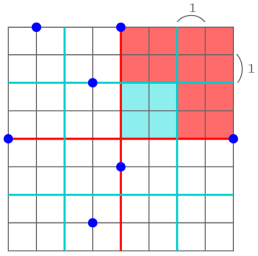
Sau đó, ta chia mỗi hình vuông làm 4 cho đến khi mỗi hình vuông chỉ còn chứa 1 nút.



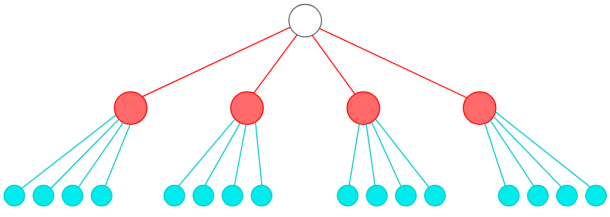
Hình 6: Chia không gian, bước thứ 1.



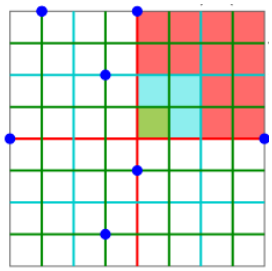
Hình 7: Cây tương ứng với bước 1.



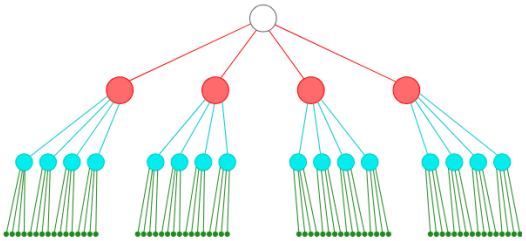
Hình 8: Chia không gian, bước thứ 2.



Hình 9: Cây tương ứng với bước 2.



Hình 10: Chia không gian, bước thứ 3.



Hình 11: Cây tương ứng với bước 3.

Với mỗi nút trên cây, ta có hình vuông S tương ứng. Ta xây dựng các bài toán multipath, mỗi bài toán ứng với một tổ hợp điểm V trên cạnh hình vuông S, ta giải bài toán để tìm đường đi P sao cho $v \in P \ \forall v \in V$.

Tuy nhiên, vì không được biết trước tập hợp các điểm $v \in V$ mà đường đi TSP cắt hình vuông S, nên không mất tổng quát, ta giả sử đường đi TSP chỉ cắt hình vuông tại các điểm cố định. Các điểm này ta gọi là **portal**.

Bước 3: Quy hoạch động

Ta lập một bảng quy hoạch động để lưu tất cả lời giải của các bài toán multipath đối với mỗi hình vuông cần xét.

	Nút	Bài toán 1	Bài toán 2	Bài toán 3	Bài toán 4	...
Nút trên cây	1	...	...	...	...	...
	2	...	...	...	...	...
	3	...	...	...	...	...
	4	...	...	...	...	...
	5	...	...	...	...	...
	6	...	...	...	...	...

Với các nút lá trên cây, bài toán ở trường hợp cơ bản (0 hoặc 1 điểm cần đi qua). Bảng quy hoạch động sẽ lưu lời giải của nút lá.

Với các nút còn lại của cây, bảng quy hoạch động lưu vị trí các bài toán con đem lại kết quả.

Ta giải từng bài toán trong bảng quy hoạch động từ dưới lên trên. Các nút lá lần lượt được giải bằng vét cạn, và các nút cha lần lượt tìm 4 bài toán con đem lại đường đi ngắn nhất.

Bước 4: Tổng hợp kết quả

Hàng đầu tiên trên bảng quy hoạch động sẽ trỏ đến 4 bài toán con tốt nhất.

Tại mỗi bài toán con, lần lượt đi xuống bảng quy hoạch động để tìm đường đi tốt nhất.

II.2.2.1.c. Cơ sở lý thuyết của Arora PTAS

Một trong những kết quả quan trọng nhất trong lý thuyết thuật toán xấp xỉ của Arora là việc chứng minh định lý cấu trúc. Định lý này đưa ra mối liên hệ giữa sai số ε cho trước và kết quả của thuật toán xấp xỉ.

Định lý: Định lý cấu trúc: Cho một sai số $\varepsilon > 0$ cho trước. Không mất tổng quát, giả sử khoảng cách tối thiểu giữa các nút là 2 và L là độ dài cạnh hình hộp vuông bao quanh mọi điểm $x \in X$ cần giao hàng. Với một vector $a \in R^d$ ngẫu nhiên, đặt $X_a = \left\{x + a \mid x \in X, a = (a_1, a_2, ..., a_d) \in \mathbb{R}^d, 0 < a_i < \frac{L}{2} \ \forall\ 0 < i \leq d\right\}$. Khi đó, với xác suất tốt thiểu $\frac{1}{2}$, tồn tại đường đi TSP có tổng chiều dài $(1 + \varepsilon) \cdot \text{OPT}$, cắt các cạnh trên 4-tree tối đa r lần tại các portal.

Định lý cấu trúc cho thấy, với một phép tịnh tiến ngẫu nhiên cho mọi $x \in X$, ta có thể giải xấp xỉ TSP với một sai số biết trước.

II.2.3. Mở rộng Arora PTAS để giải bài toán Bin packing 2D

Những ý tưởng chính của Arora PTAS bao gồm:

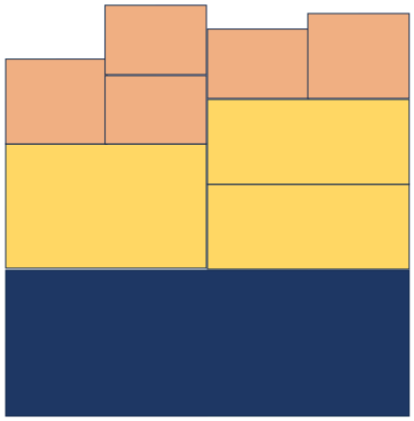
- Gộp các đối tượng giống nhau bằng cách làm tròn.
- Chia bài toán lớn thành các bài toán con, cho đến khi gặp trường hợp cơ bản nhất.
- Lần lượt giải từ bài toán con lên bài toán mẹ bằng quy hoạch động.

Các phương pháp này cũng có thể được mở rộng cho các bài toán khác, bao gồm bin packing.

Sau đây, ta sẽ thiết kế một PTAS cho bin packing 2D, dựa trên các kỹ thuật của Arora.

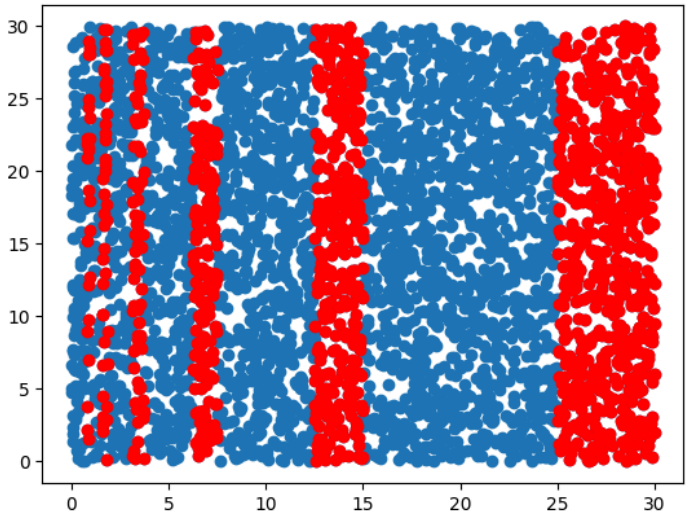
II.2.3.1. Tiền xử lý dữ liệu

Ý tưởng chính của thuật toán: Giống như Arora chia đôi mặt phẳng để tạo thành các bài toán con, ta chia đôi bin tại các điểm nhất định, mỗi điểm tạo thành 2 bài toán bin packing nhỏ hơn.



Hình 12: Mục tiêu: chia bin làm 2 ở các vị trí nhất định.

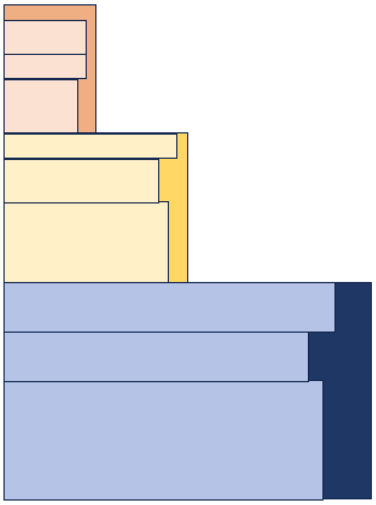
Chọn trước tham số a. Với sai số ε đã biết, ta chọn các vật có chiều dài $x_i \in \left(\frac{a-\varepsilon}{k}, \frac{a}{k}\right)$, trong đó $k = 1, 2, ...$



Hình 13: Trục hoành: chiều dài của vật. Trục tung: chiều cao của vật
Màu đỏ: Các điểm được chọn. Màu xanh: Các điểm chưa được chọn

$\forall x_i \in \left(\frac{a-\varepsilon}{k}, \frac{a}{k}\right)$, ta coi $x_i \approx \frac{a}{k}$ và lần lượt xếp x_i vào bin theo thứ tự y_i từ lớn đến bé.

Ý nghĩa của việc chia đôi chính là, khi ta đã xấp xỉ chiều dài $x_i \approx \frac{a}{k}$ và đặt bin con chiều dài $\frac{a}{k}$, thì ta không phải xét chiều dài x_i của mỗi vật khi xếp vào bin con. Khi đó bài toán trở thành bài toán bin packing 1D, ta chỉ cần chú ý chiều dài bin và chiều cao y_i



Hình 14: Các vật khi xếp xấp xỉ với chiều dài bin

Khi đó, sau mỗi bước bin packing 1D với $x_i \in \left(\frac{a-\varepsilon}{k}, \frac{a}{k}\right)$, ta thu được 2 bài toán bin packing 1D với $x_i \in \left(\frac{a-\varepsilon}{k+1}, \frac{a}{k+1}\right)$

III. Triển khai thuật toán

III.1. Các công nghệ được sử dụng

Theo chỉ dẫn của công ty thực tập, em lập trình toàn bộ các dự án nêu trên trong Python, chỉ sử dụng các thư viện tính toán cơ bản như NumPy, Numba, và các thư viện đồ họa NetworkX, Matplotlib.

- **NumPy:** NumPy là một thư viện mã nguồn mở cho ngôn ngữ lập trình Python, được sử dụng rộng rãi trong lĩnh vực khoa học dữ liệu và tính toán khoa học. Tên của nó viết tắt từ "Numerical Python". NumPy cung cấp các cấu trúc dữ liệu mạnh mẽ như mảng (array) và ma trận (matrix), đi kèm với một bộ hàm toán học phong phú để thao tác và tính toán trên các dữ liệu này. Nhờ có NumPy, các nhà khoa học và kỹ sư có thể thực hiện các phép tính số học phức tạp một cách hiệu quả và nhanh chóng, từ đó hỗ trợ cho việc phân tích dữ liệu, mô phỏng và xây dựng các thuật toán phức tạp. NumPy là nền tảng cho nhiều thư viện khác trong Python như SciPy, Pandas và Matplotlib, góp phần làm cho Python trở thành một ngôn ngữ mạnh mẽ trong lĩnh vực khoa học dữ liệu và học máy.
- **Numba:** Numba là một thư viện mã nguồn mở dành cho Python, được thiết kế để tăng tốc độ thực thi các đoạn mã Python bằng cách biên dịch chúng thành mã máy (machine code) trong thời gian thực. Sử dụng Just-in-Time (JIT) compilation, Numba cho phép các hàm Python đạt được tốc độ gần như tương đương với các ngôn ngữ lập trình cấp thấp như C hoặc Fortran mà không cần thay đổi nhiều trong mã nguồn gốc. Điều này đặc biệt hữu ích trong các ứng dụng khoa học dữ liệu, tính toán khoa học và học máy, nơi mà hiệu suất là yếu tố quan trọng. Numba tương thích tốt với các thư viện phổ biến như NumPy, giúp tối ưu hóa các thao tác trên mảng và ma trận một cách hiệu quả. Bằng cách đơn giản hóa quá trình tăng tốc mã Python, Numba trở thành một công cụ mạnh mẽ cho các nhà phát triển và nhà nghiên cứu trong việc nâng cao hiệu suất của các ứng dụng tính toán phức tạp.
- **Matplotlib:** Matplotlib là một thư viện mã nguồn mở dành cho Python, được sử dụng rộng rãi để tạo ra các biểu đồ và hình ảnh minh họa dữ liệu. Thư viện này cung cấp các công cụ linh hoạt và mạnh mẽ cho việc vẽ các loại biểu đồ như biểu đồ đường, biểu đồ thanh, biểu đồ tán xạ, biểu đồ bánh và nhiều loại biểu đồ khác. Matplotlib hỗ trợ việc tùy chỉnh chi tiết các yếu tố đồ họa, từ màu sắc, kiểu dáng đến chú thích và nhãn, giúp người dùng tạo ra những hình ảnh trực quan và chuyên nghiệp. Thư viện này được đánh giá cao trong cộng đồng khoa học dữ liệu và phân tích dữ liệu vì khả năng kết hợp dễ dàng với các thư viện khác như NumPy, Pandas và SciPy. Matplotlib là một công cụ quan trọng trong việc trình bày và khám phá dữ liệu, giúp các nhà nghiên cứu và nhà phân tích truyền tải thông tin một cách rõ ràng và hiệu quả.

III.2. Phép nhúng lò xo

Để đưa bài toán TSP tổng quát về bài toán E-TSP, ta dùng phép nhúng lò xo để nhúng đồ thị vào không gian Euclid.

Đầu vào: Ma trận khoảng cách A, mô tả đồ thị $G = (V, E)$

Đầu ra: Mảng 2 chiều, chứa tọa độ các đỉnh $v \in V$ của đồ thị G khi đã được nhúng.

III.3. Giải E-TSP bằng PTAS của S.Arora

Từ kết quả của phép nhúng lò xo, ta triển khai các bước của Arora PTAS trong Numba.

Đầu vào: Tọa độ các điểm cần đi qua, dưới dạng một mảng 2 chiều:

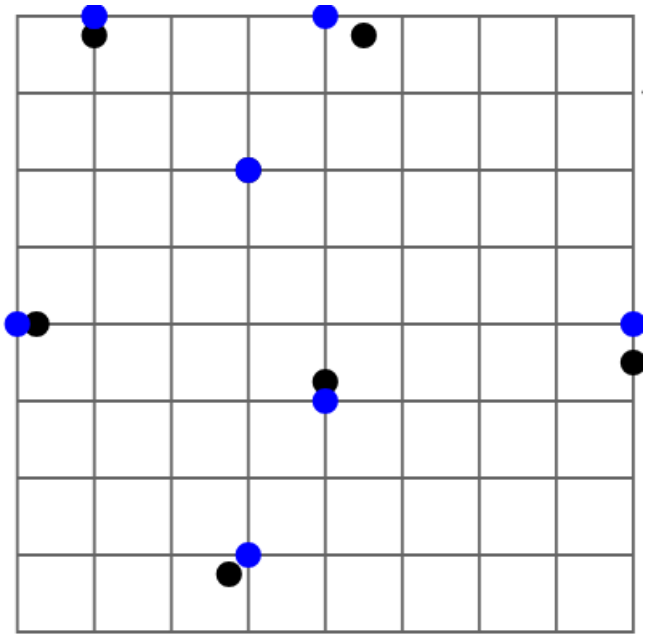
$$X^0 = \{(x_0, x_1) \in R^2\}$$

Đầu ra: Tọa độ các điểm cần đi qua, xếp theo thứ tự đến trước → đến sau, dưới dạng một mảng 2 chiều.

III.3.1. Dữ liệu tọa độ

Vì nhân mọi tọa độ với một hằng số không làm thay đổi bản chất bài toán, ta lấy $X = kX^0$, với $k = O(\frac{1}{\varepsilon})$.

Tiếp theo, ta làm tròn tọa độ của mọi điểm. Các điểm quá gần nhau sẽ được hợp thành 1 điểm duy nhất.



Hình 15: Các tọa độ trước và sau khi làm tròn. Màu đen: Trước khi làm tròn. Màu xanh: Sau khi làm tròn.

III.3.2. Xây dựng 4-tree

Trong Numba, 4-Tree có thể được biểu diễn bằng cấu trúc dữ liệu cây bằng `@jitclass`

Tuy nhiên, khả năng hỗ trợ `jitclass` của Numba rất hạn chế. Vậy nên, để tăng tốc độ tính toán, cấu trúc cây được biểu diễn dưới dạng một `array` trong NumPy.

Mỗi hàng của `array` chứa thông tin về một nút trên cây, và mỗi hàng chứa thông số về các nút tương ứng.

	ID	Nút mẹ	Hoạt động	Bậc	Nút con	AABB
Các nút	1	NULL	TRUE	0	2	...
	2	1	TRUE	1	NULL	...
	3	1	TRUE	1	6	...
	4	1	TRUE	1	NULL	...
	5	1	TRUE	1	NULL	...
	6	3	TRUE	2	NULL	...
	7	3	TRUE	2	NULL	...
	8	3	TRUE	2	NULL	...
	9	3	TRUE	2	NULL	...

Bảng 1: Một ví dụ về biểu diễn cây bằng ma trận trong NumPy

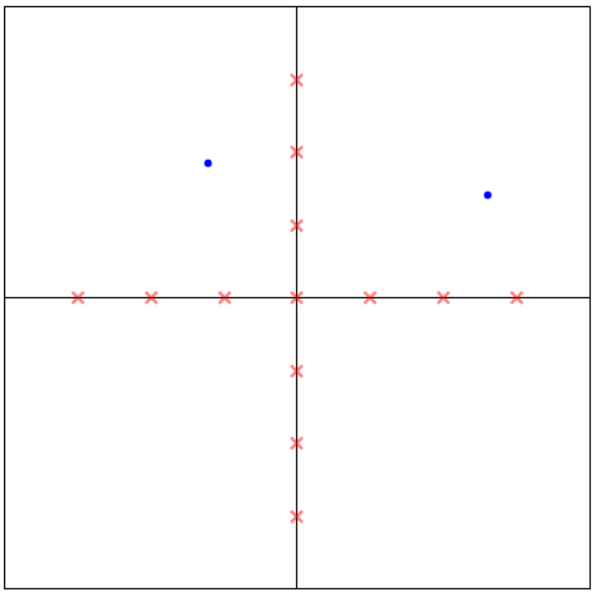
Trong đó:

- ID: Chỉ số của nút trên ma trận.
- Nút mẹ: ID của nút mẹ.
- Hoạt động: NULL/TRUE - hàng hiện có chứa dữ liệu về nút không.
- Bậc: Bậc của nút trên cây
- Nút con: ID của nút con đầu tiên. Để ta chỉ cần lưu nút con đầu tiên, mọi nút con của một nút sẽ được xếp cạnh nhau

  VD: Để tìm nút con thứ 3 của nút a, lấy 2 + ID nút con thứ 1 của a
- AABB: Axis-aligned bounding box - Mô tả vị trí và kích cỡ của ô vuông tương ứng với nút này.

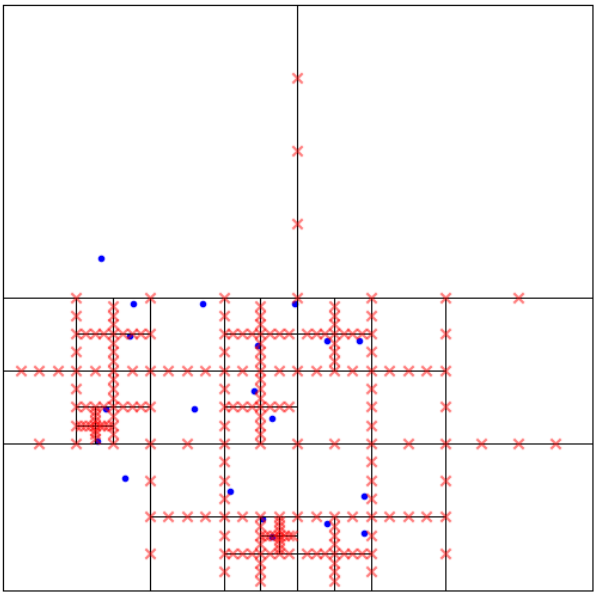
III.3.3. Chia nhỏ bài toán

Tại mỗi nút trên cây, ta tạo các portal cách đều nhau ở các đường chia. Portal được lưu trong một mảng 2 chiều `portallist`



Hình 16: Ví dụ về đặt portal. Chấm xanh là các điểm đường đi TSP cần đi qua. Dấu x đỏ là các portal.

Lặp lại bước trên với mọi nút trên cây.



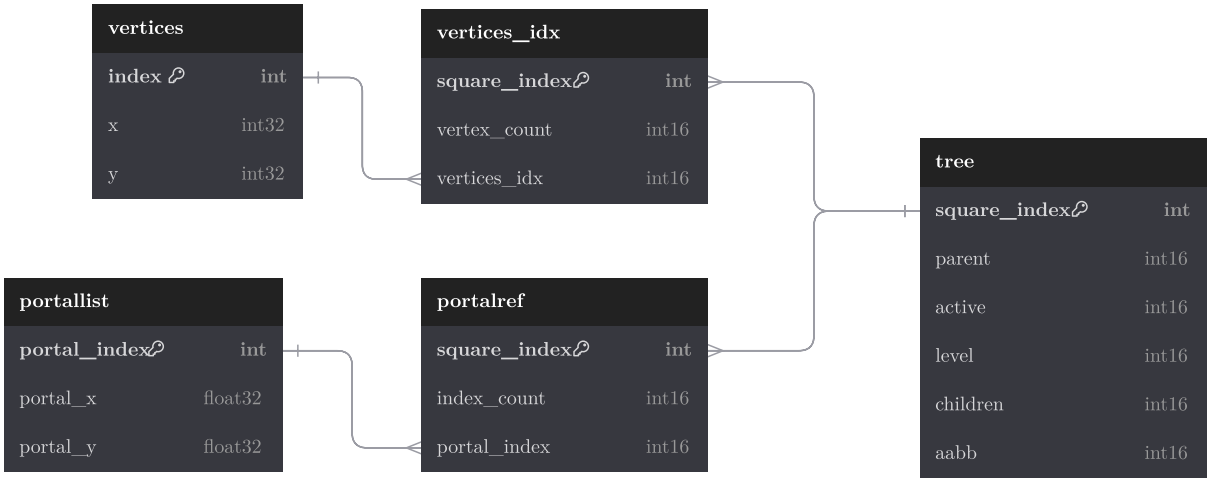
Hình 17: Kết quả sau khi chia cây và đặt portal

Dữ liệu của cây được lưu trong các mảng sau.

- `tree`: Thông tin về mỗi nút trên cây.
- `vertices`: Mọi điểm cần đi qua.
- `vertices_idx`: Thông tin về các điểm nằm trong mỗi nút
  - VD: nếu ô thứ 5 chứa các điểm thứ 0, 1, 2 trong `vertices`, thì tại `square_index = 5` , ta có `vertex_count = 3`, `vertices_idx = [0, 1, 2]`

- `portallist`: Danh sách các portal.
- `portalref`: Danh sách các portal trong mỗi ô
  - VD: nếu ô thứ 5 của cây chứa các portal thứ 0, 1, 2 trong `portallist`, thì tại `square_index = 5` thì `index_count = 3` và `portal_index = [0, 1, 2]`

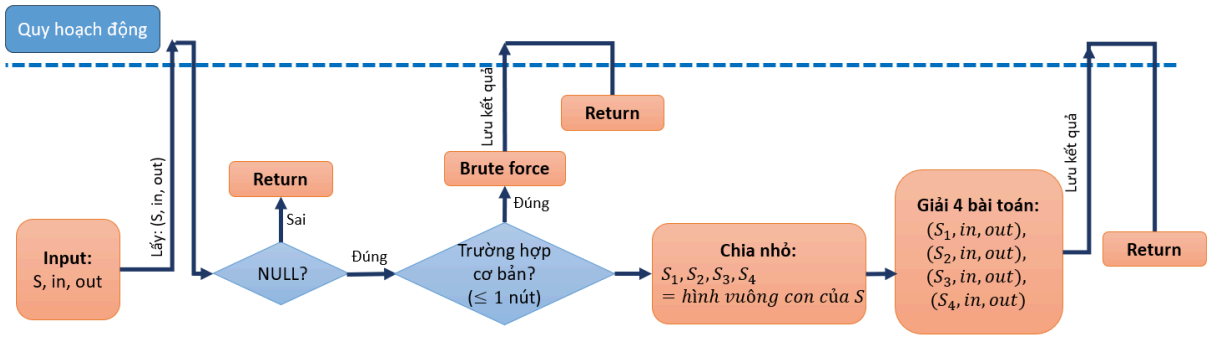
Ta



Hình 18: Cấu trúc và mối liên hệ giữa các mảng. Hàng đầu tiên của mỗi bảng là chỉ số của hàng đó trong mảng. Các hàng còn lại tương ứng với mỗi phần tử trong phẩn. Các mũi tên mô tả quan hệ của các biến lưu chỉ số.

III.3.4. Quy hoạch động

Quá trình quy hoạch động được tóm tắt như sau:



Hình 19: Sơ đồ thuật toán giải quy hoạch động.

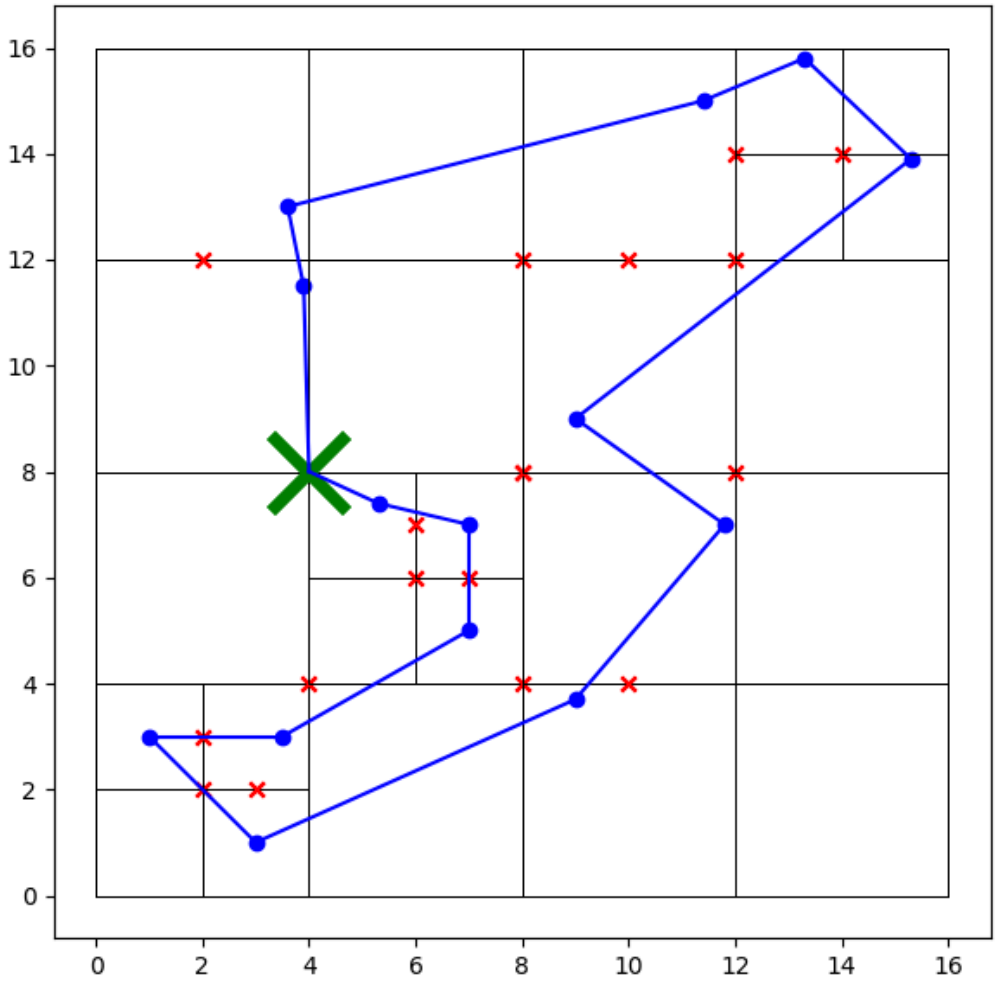
Ta dựng bảng quy hoạch động theo cấu trúc sau:

	Nút	Bài toán 1	Bài toán 2	Bài toán 3	Bài toán 4	...
Nút trên cây	1	...	...	...	...	...
	2	...	...	...	...	...
	3	...	...	...	...	...
	4	...	...	...	...	...
	5	...	...	...	...	...
	6	...	...	...	...	...

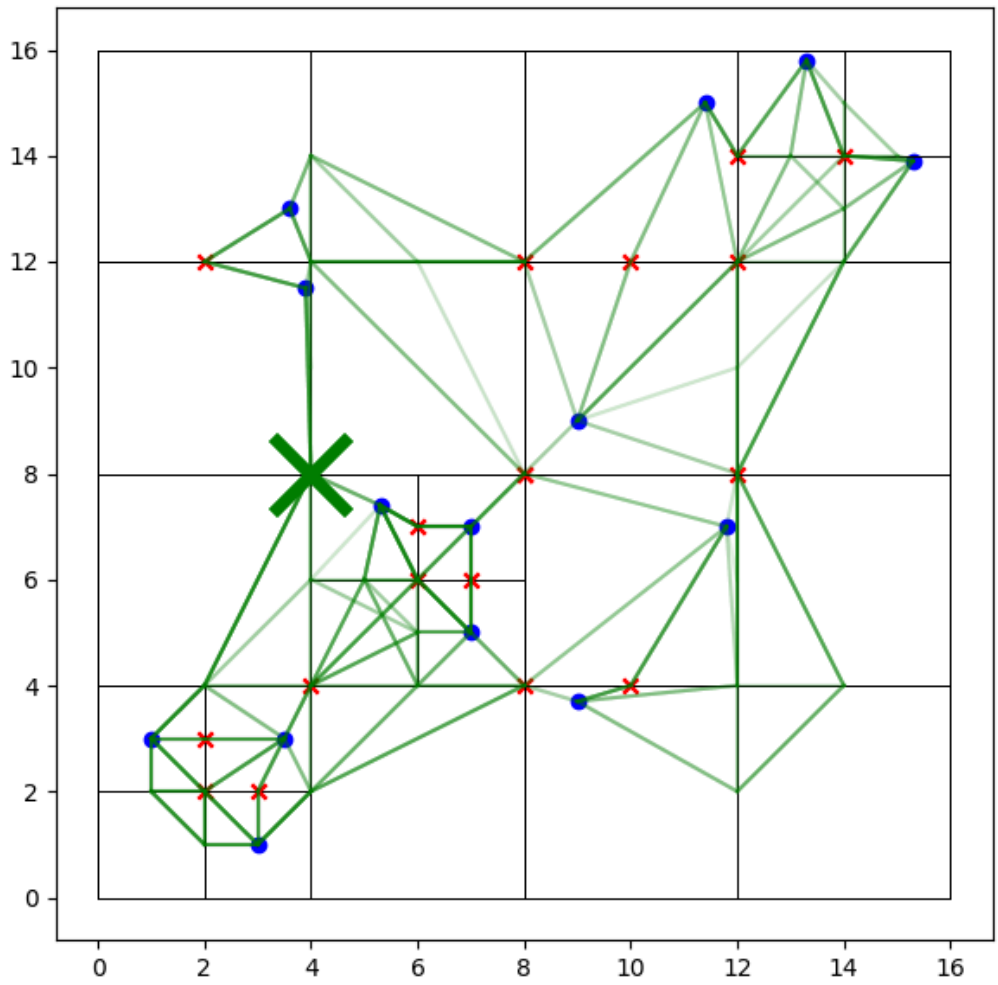
Bảng quy hoạch động sẽ lần lượt được dựng từ nút con đến nút cha.

III.3.5. Tổng hợp kết quả

Một khi có bảng quy hoạch động, ta lần lượt đi từ nút mẹ xuống để tổng hợp kết quả.



Hình 20: Đường đi TSP thu được từ bảng quy hoạch động

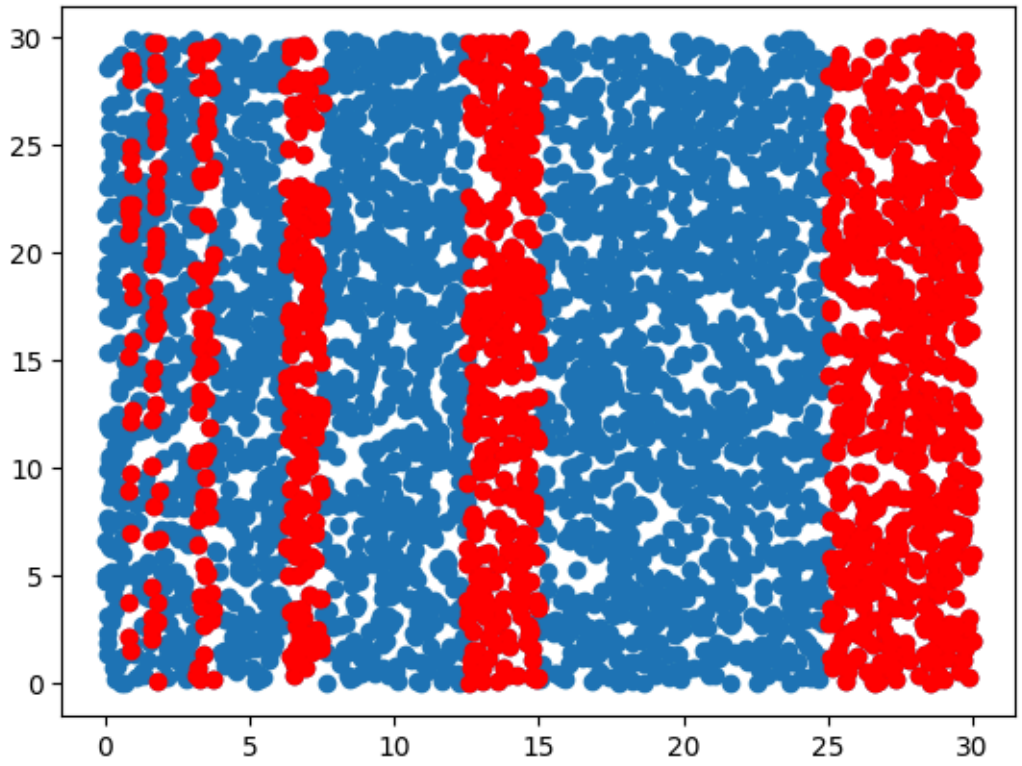


Hình 21: Thông tin trong bảng quy hoạch động

III.4. Triển khai PTAS cho bài toán Bin packing 2D

III.4.1. Tiền xử lý dữ liệu

Ta tách dữ liệu ra theo các phần $\left(\frac{a-\varepsilon}{k}, \frac{a}{k}\right)$



Hình 22: Tách dữ liệu

Với mỗi k xác định, ta lưu mỗi vật trong khoảng $\left(\frac{a-\varepsilon}{k}, \frac{a}{k}\right)$ vào một max heap. Như vậy thời gian tìm vật lớn nhất sẽ là $O(1)$.

III.4.2. Thiết lập bài toán con

Bài toán con của ta sẽ là một bài toán bin packing 1D:

Cho một tập các vật I cho trước, mỗi vật có chiều cao y_i. Xếp các vật vào bin B có chiều cao Y sao cho:

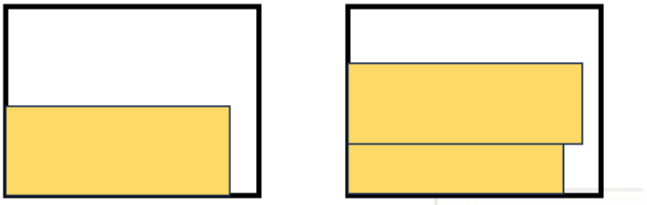
$$\sum_{i \in B} y_i \leq Y$$

III.4.3. Thuật toán

Bắt đầu với bin đầu tiên, lần lượt xếp các vật lớn nhất vào đến khi không thể xếp tiếp, rồi chia đôi bin.

Hình 23: Bước 1: Xếp đẩy bin

Hình 24: Bước 2: Chia đôi bin



Hình 25: Bước 3: Giải bài toán con ở các bin con

Sau khi xếp đầy 1 bin, ta chuyển sang bin tiếp theo, đến khi không còn vật nào để xét.

IV. Tài liệu tham khảo

Sanjeev Arora. 1998. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. J. ACM 45, 5 (Sept. 1998), 753–782. https://doi.org/10.1145/290179.290180

Rao, Satish and Warren D. Smith. "Approximating geometrical graphs via "spanners" and "banyans"." Symposium on the Theory of Computing (1998).

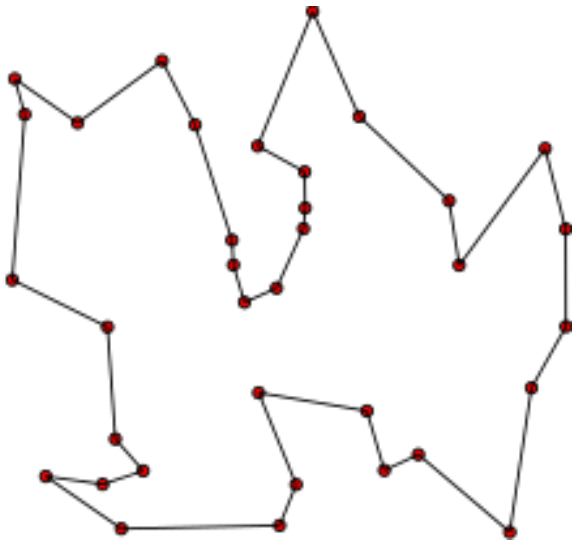
Vijay V. Vazirani. 2010. Approximation Algorithms. Springer Publishing Company, Incorporated.

Sanjeev Arora, James R. Lee, and Assaf Naor. 2005. Euclidean distortion and the sparsest cut. In Proceedings of the thirty-seventh annual ACM symposium on Theory of computing (STOC '05). Association for Computing Machinery, New York, NY, USA, 553–562. https://doi.org/10.1145/1060590.1060673

I. Introduction

I.1. Travelling Salesman Problem

The Traveling Salesman Problem (TSP) is one of the most classic and famous problems in the fields of optimization and graph theory. The problem is defined as follows: Given a set of cities and the distances between each pair of cities, the salesman needs to find the shortest possible route that visits each city exactly once and returns to the original city.



Hình 26: An example of a solution to the TSP. The vertices of the graph are the delivery points. The edges of the graph are the paths..

The TSP is highly significant in many fields such as logistics, transportation, manufacturing, and even in computer science applications. The ability to effectively solve the TSP can lead to significant cost and time savings in these industries. Additionally, the TSP also plays an important role in the research and development of optimization and search algorithms.

The name "Traveling Salesman Problem" first appeared in the 1930s when Irish mathematician Hamilton and British mathematician Kirkman studied cycles in graphs. However, the problem was only clearly defined and became popular thanks to the research of mathematicians and economists in the 1950s.

With the development of computer technology and new algorithms, solving the TSP has made significant strides.

I.1.1. Difficulty of the Problem

The TSP is an NP-hard problem, which means that there is no algorithm that can solve all instances of this problem in polynomial time, unless P = NP. This makes the TSP a major challenge for researchers and programmers. As the number of cities increases, the number of possible routes increases exponentially, making it very difficult to find the optimal solution.

Therefore, finding a sufficiently good approximate solution is much more important than finding an optimal solution.

I.1.2. Methods for Solving the TSP

The TSP is typically solved using one of the following classical methods:

Brute Force Method: This method tries all possible routes and selects the shortest one. However, this method is only feasible for very small sets of cities due to the factorial time complexity as the number of cities increases.

Heuristic Algorithms: Algorithms such as Greedy, Nearest Neighbor, and Christofides allow finding approximate solutions in reasonable time. While these solutions are not guaranteed to be optimal, they often provide sufficiently good results.

Metaheuristic Algorithms: Algorithms like Simulated Annealing, Genetic Algorithms, and Ant Colony Optimization help in finding near-optimal solutions by intelligently and systematically exploring the solution space.

Dynamic Programming and Linear Programming Methods: These methods use mathematical techniques to find the optimal solution for the TSP. However, they are usually only feasible for problems of moderate size.

I.1.3. Practical Applications of the TSP

Logistics and Transportation: TSP is used to optimize routes for delivery vehicles, helping to reduce fuel costs and travel time.

Manufacturing: In production lines, TSP helps optimize the movement sequence of tools and materials, improving efficiency and reducing waiting times.

Travel and Tourism: TSP aids in planning efficient travel itineraries, enabling tourists to visit many locations in the shortest possible time.

I.2. The Bin Packing Problem

The Bin Packing Problem (also known as Bin Loading) is a classic optimization problem in computer science and mathematics. The problem is defined as follows: Given a set of objects of different sizes and a limited number of bins, each with a fixed capacity, the task is to arrange the objects into the bins so that the number of bins used is minimized, while ensuring that the total size of the objects in each bin does not exceed the bin's capacity.

I.2.1. Real-World Applications

The Bin Packing problem arises in various real-world situations such as:

- Warehouse management: Organizing goods into storage warehouses to utilize space efficiently.
- Transportation and logistics: Allocating goods into shipping containers to minimize the number of containers used.
- Memory management: Allocating processes into memory blocks in computers to optimize memory usage.

I.3. Research Objective

The objective of this project is to investigate and apply Polynomial Time Approximation Scheme (PTAS) methods to solve the Traveling Salesman Problem (TSP) and Bin Packing problem, aiming to provide an alternative approach to classical methods such as Heuristic and Metaheuristic. Specifically, the project will be based on the theoretical foundation laid out by researchers S. Arora, S. B. Rao, and V. V. Vazirani, who have made significant contributions to the development of approximation algorithm theory.

- **Exploration of PTAS Theory**: Study and gain a deep understanding of PTAS theory, especially the works of Arora, Rao, and Vazirani. PTAS offers approximate solutions with arbitrary $(1 + \varepsilon)$ error, and can run in polynomial time, making it a powerful tool for complex combinatorial optimization problems.
- **Development of Specific Algorithms**: Based on the studied theory, develop and implement a specific PTAS algorithm for the TSP and Bin Packing problems.
- **Performance Evaluation**: Investigate the runtime of the developed PTAS algorithms. Evaluate the feasibility of applying the algorithms in real-world scenarios, such as logistics management, goods distribution, and delivery services.
- **Proposal of New Development Directions**: Based on the achieved results, propose new research and development directions for applying PTAS to other NP-hard optimization problems. Encourage further research and improvement of approximation methods in this field.

With these objectives, the project aims to open up a new direction in the research and application of approximation algorithms for other NP-hard problems. Applying PTAS not only stabilizes the error of the problems but also contributes to the development of approximation algorithm theory and their application in complex real-world problems.

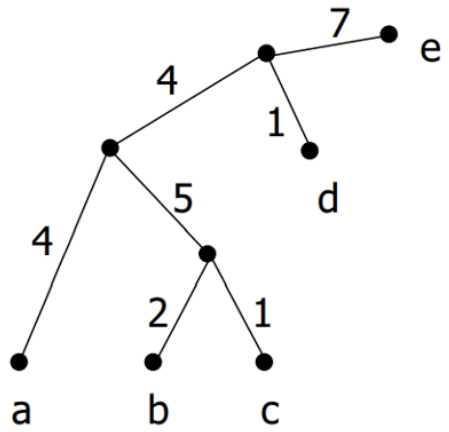
II. Theoretical Foundation

II.1. Mathematical Models

II.1.1. Map Modeling

In practice, map data is often provided in the form of a distance matrix. Therefore, in the first step, we model the map to be considered as a graph.

M	a	b	c	d	e
a	0	11	10	9	15
b	11	0	3	12	18
c	10	3	0	11	17
d	9	12	11	0	8
e	15	18	17	8	0



Hình 27: On the left: Distance matrix. On the right: Graph corresponding to the distance matrix..

Let $G^0 = (V, E)$ be the weighted graph describing the traffic network on the map under consideration. Where:

- $V = V_1 \cup V_2 \cup V_3 = \{v_i \in V \mid i \in \mathbb{N}^*\}$ where:
  - V_1 are intersections (T-junctions, crossroads, etc.).
  - V_2 are delivery points.
  - V_3 are warehouses.
- $E = \left\{e_{\{ij\}} = (v_i, v_j), |, v_i, v_j \in V; \quad i, j \in \mathbb{N}; \quad i \neq j\right\}$ are the roads on the traffic network.
  - To simplify the problem, we will assume $e_{\{ij\}} = e_{\{ji\}} \quad \forall i, j \in \mathbb{N}^*$.

From there, we define a path $P \subset E$ on the graph G^0 to be a set of connected edges on G^0.

$$P = \left\{\left(e_{\{ij\}}, e_{\{jk\}}\right) \mid i, j, k \in \mathbb{N}^*\right\}$$

Notation:

- $w\left(e_{\{ij\}}\right)$ is the length of edge $e_{\{ij\}} \in E$.
- $d(v_i, v_j)$ is the shortest path length between vertex v_i and vertex v_j.

The length of the path $P \subset E$ is denoted as:

$$d(P) = \sum_{\{e_{\{ij\}} \in P\}} w\left(e_{\{ij\}}\right) \qquad \text{for } e_{\{ij\}} \in P$$

II.1.2. Mathematical Model: E-TSP

In the general TSP problem, our task is to find a path on a graph $G = (V, E)$ in a generalized setting. This is a very challenging problem due to the combinatorial explosion.

However, when we embed the graph $G = (V, E)$ into Euclidean space, research by S. Arora has shown that we can leverage the geometric structures of this space to approximate the problem in polynomial time. The TSP problem then becomes the Euclidean Travelling Salesman Problem (E-TSP).

Next, we will state the E-TSP problem.

Given a set X^0 of delivery points in Euclidean space:

$$X^0 = \{x_1, ..., x_n\} = \{x_i \in \mathbb{R}^d \mid i \in \mathbb{N}^*, i \leq n\}$$

and a starting point $x_0 = (a_1, ..., a_d) \in \mathbb{R}^d$.

The problem objective: find a route $P = \{(x_0, x_i), (x_i, x_j), ..., (x_k, x_0)\}$ that minimizes the total length of the route $d(P) = \sum_{(x_i, x_j) \in P} \big(d(x_i, x_j)\big)$.

II.1.3. Problem Model: 2D Bin Packing

The 2D Bin Packing problem can be stated as follows:

Given fixed-size rectangular bins (also called bins) with dimensions (W, H), and a set I of items with sizes (w_i, h_i). Find positions (x_i, y_i) for each item such that:

- No items overlap.
- The minimum number of bins containing items is used.

II.2. Approaches

II.2.1. Polynomial Time Approximation Scheme (PTAS)

In computer science (especially algorithm theory), Polynomial Time Approximation Scheme (PTAS) is a family of approximation algorithms for optimization problems (often NP-hard optimization problems).

A PTAS is a family of algorithms to solve a certain problem, where each algorithm depends on a predefined parameter $\varepsilon > 0$. The output of the algorithm is a solution with a result within $(1 + \varepsilon)$ range of the optimal solution (or $(1 - \varepsilon)$ for maximization problems). For example, for the Euclidean Travelling Salesman Problem, a PTAS will produce a travel itinerary with a maximum length of $((1 + \varepsilon)c \cdot L)$, where L is the length of the shortest travel itinerary.

II.2.1.1. Definition of PTAS

Below, we will provide a definition for a PTAS.

Definition - Approximation Method: Given Π as an NP-hard optimization problem, with the objective function f_Π. We say $\mathcal{A}$ is an approximation method for

problem Π if for I being an input of Π and $\varepsilon > 0$ being a parameter, we have $\mathcal{A}(I, \varepsilon) = s$ satisfying:

- $f_\Pi(I, s) \leq (1 + \varepsilon) \cdot \text{OPT}$ if Π is a minimization problem.
- $f_\Pi(I, s) \geq (1 - \varepsilon) \cdot \text{OPT}$ if Π is a maximization problem.

Where OPT is the value of the objective function for the optimal solution.

The parameter ε is also commonly referred to as the approximation factor of the approximation method $\mathcal{A}$.

Definition - Polynomial Time Approximation Scheme: Given $\mathcal{A}$ as an approximation method for problem Π. We call $\mathcal{A}$ a Polynomial Time Approximation Scheme (PTAS) for Π if for parameter $\varepsilon > 0$, the running time of $\mathcal{A}$ is a polynomial in I.

II.2.1.2. Reasons for Using PTAS

PTAS is a powerful tool when approximating NP-hard problems is necessary.

Key differences of PTAS compared to classical methods such as Heuristics and Metaheuristics include:

- **Accuracy:** We can choose a small ε arbitrarily, so the error is always known beforehand. This is the best result we can expect for NP-hard problems, which we cannot find optimal solutions in polynomial time.
- **Speed:** The algorithm always runs in polynomial time for every fixed ε.

However, the definition of PTAS does not consider the running time as ε approaches 0. Therefore, an algorithm running in $O\left(n^{\frac{1}{\varepsilon}}\right)$ time or even $O\left(n^{e^{\frac{1}{\varepsilon}}}\right)$ is still considered a PTAS. Hence, for some PTAS, the running time can become impractical when ε is too small.

PTAS can be applied to combinatorial NP-hard problems such as the knapsack problem, scheduling problem, and many other optimization problems.

II.2.2. PTAS for E-TSP by S. Arora

S. Arora (Princeton University) devised a PTAS for the E-TSP problem in $O\left(n\log^k(n)\right)$ time with an approximation factor of $(1 + \varepsilon)$, where k is proportional to $\frac{1}{\varepsilon}$ and d when the graph is pre-embedded. His research on this was awarded the Gödel Prize in 2010.

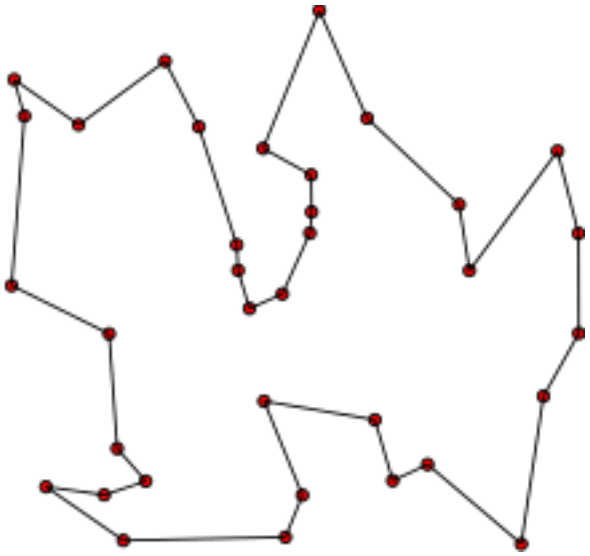
S. Arora's PTAS for the E-TSP problem represents a significant breakthrough in the field of approximation algorithms, laying the groundwork for many approximation algorithms for other NP-hard problems, such as Minimum Spanning Tree, Minimum Steiner Tree, Minimum Weight Perfect Matching, etc.

From here, we will refer to Arora's PTAS theory as Arora PTAS.

II.2.2.1.a. Illustrative Example

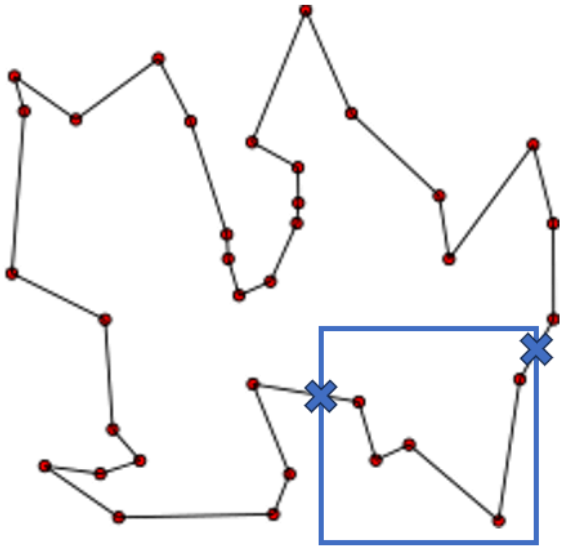
To illustrate the idea of Arora PTAS, let's first consider a specific example.

Suppose we have an E-TSP problem, for which we already know the optimal tour as follows:



Hình 28: An example of an optimal TSP tour

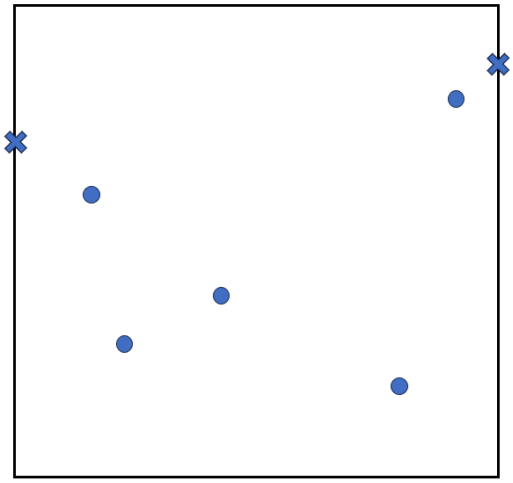
Now, let's take an arbitrary square from the optimal tour.



Hình 29:

From here, we can separate the points within the square into a subproblem. In S. Arora's polynomial approximation theory, this is called the multipath problem.

The objective of the problem is to find a set of paths that pass through all the points on the square and all the points inside the square.



Hình 30: The subproblem.

Statement of the multipath problem: The input of the problem includes:

- A square S.
- A set of points $V = \{v_i \in \partial S \texttt{ mid } i < r\}$, where $r \in \mathbb{N}^*$ consists of points on the boundary of S.
- A set of points $X_S \subseteq X$ that need to be visited within square S.

The problem output: paths $P = \{(v_a, x_i), ..., (x_j, v_b)\}$, such that $x \in P$ for all $x \in X_S$.

This subproblem can be further subdivided. We can divide the problem until the base case: when there is only ≤ 1 point inside S. In that case, the subproblem can be solved by brute force in $O(r) = O(\frac{1}{\varepsilon})$ time.

Arora's main idea is to divide the E-TSP problem into multiple subproblems in the base case ($|X_S| \leq 1$), then solve and combine the results.

II.2.2.1.b. The algorithm

Step 1: Preprocessing Coordinate Data

Multiplying every coordinate $x \in X^0$ by a constant doesn't change the essence of the problem. Therefore, we take $X' = \{x' = kx \mid x \in X^0\}$ and $y' = ky_0$.

Next, we round the coordinates of every point $x' \in X'$ and point y. The goal is to merge points that are too close to each other and consider them as one point. Then we have the new data:

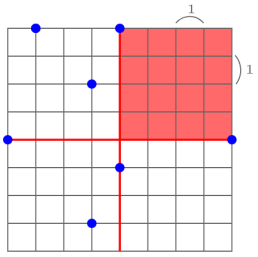
$$X = \{x = \text{round}(x') \mid x' \in X'\}$$

$$y = \text{round}(y')$$

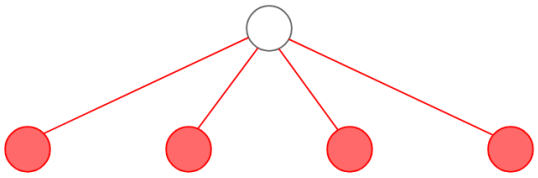
Step 2: Divide the problem and set up a 4-tree

The division steps of the problem can be described by a 4-tree. First, we take an arbitrary square enclosing all points with a side length of L.

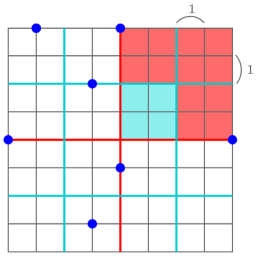
Then, we divide each square into 4 squares until each square contains only 1 node.



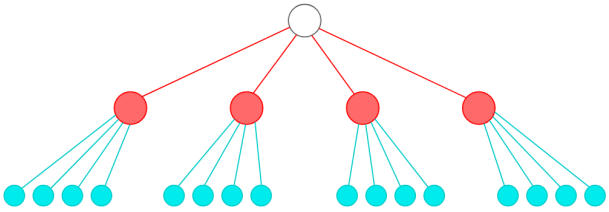
Hình 31: Space partitioning, step 1.



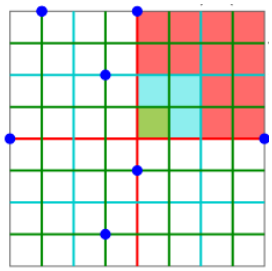
Hình 32: Step 1 tree.



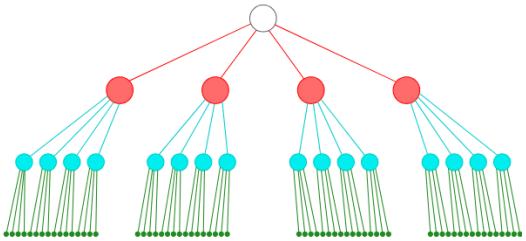
Hình 33: Space partitioning, step 2.



Hình 34: Step 2 tree.



Hình 35: Space partitioning, step 3.



Hình 36: Step 3 tree.

For each node on the tree, we have a corresponding square S. We construct multipath problems, with each problem corresponding to a combination of points V on the edge of square S. We solve the problem to find a path P such that v is in P for all v in V.

However, since we don't know the set of points v in V that the TSP path intersects with the square S beforehand, without loss of generality, we assume that the TSP path intersects the square only at fixed points. These points are called **portals**.

Step 3: Dynamic Programming

We create a dynamic programming table to store all the solutions for the multipath problems for each square under consideration.

	Nút	Subproblem 1	Subproblem 2	Subproblemn 3	Subproblem 4	...
Nút trên cây	1	...	...	...	...	...
	2	...	...	...	...	...
	3	...	...	...	...	...
	4	...	...	...	...	...
	5	...	...	...	...	...
	6	...	...	...	...	...

For the leaf nodes in the tree, representing the base cases (0 or 1 point to pass through), the dynamic programming table will store the solutions for the leaf nodes.

For the remaining nodes of the tree, the dynamic programming table stores the positions of the subproblems that yield results.

We solve each problem in the dynamic programming table from bottom to top. The leaf nodes are solved sequentially by brute force, and the parent nodes sequentially find the 4 subproblems that yield the shortest paths.

Step 4: Combining solutions

The top row of the dynamic programming table will point to the 4 best subproblems.

For each subproblem, traverse down the dynamic programming table to find the best path.

II.2.2.1.c. Theoretical Foundation of Arora PTAS

One of the most significant results in Arora's approximation algorithm theory is the proof of the structure theorem. This theorem establishes a relationship between the predetermined error margin epsilon and the outcome of the approximation algorithm.

Theorem: Structure Theorem: Given a predetermined error margin $\varepsilon > 0$. Without loss of generality, assume the minimum distance between nodes is 2 and L is the length of the edge of the square box surrounding every point x in the set of delivery points X. For a random vector a in R^d, let $X_a = \{x + a \mid x \in X, a = (a_1, a_2, ..., a_d) \in \mathbb{R}^d, 0 < a_i < \frac{L}{2}\}$ for all $0 < i \leq d\}$. Then, with a probability of at least $\frac{1}{2}$, there exists a TSP path with a total length of $(1 + \varepsilon) \cdot \text{OPT}$, cutting the edges on the maximum 4-tree r times at portals.

The Structure Theorem demonstrates that with a random translation for every x in X, we can approximate the TSP with a predetermined error margin.

II.2.3. Extending Arora's PTAS to solve the 2D Bin Packing Problem

The main ideas of Arora's PTAS include:

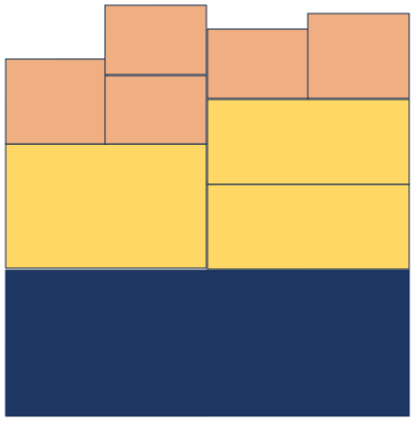
- Grouping similar objects by rounding.
- Dividing the large problem into subproblems until reaching the simplest base case.
- Solving recursively from the subproblems up to the parent problem using dynamic programming.

These methods can also be extended to other problems, including bin packing.

Here, we will design a PTAS for the 2D bin packing problem based on Arora's techniques.

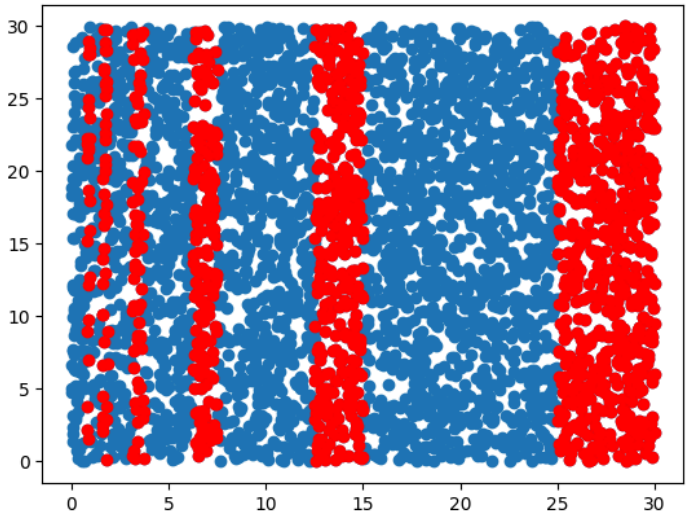
II.2.3.1. Data preprocessing

The main idea of the algorithm: Similar to Arora's approach of dividing the plane into quarters to create subproblems, we divide bins at certain points, each point forming 2 smaller bin packing problems.



Hình 37: Objective: Divide bins into 2 at certain positions

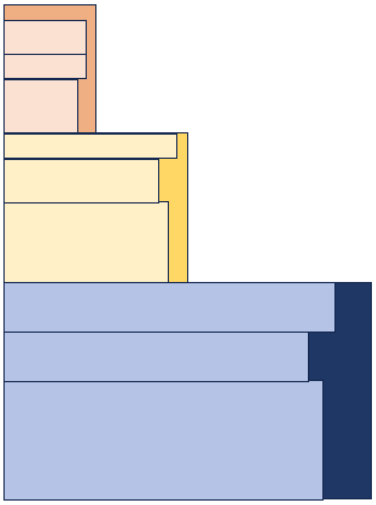
Choose a parameter a beforehand. With the known error ε, select objects with lengths x_i in the range $\left(\frac{a-\varepsilon}{k}, \frac{a}{k}\right)$, where $k = 1, 2, ...$



Hình 38: Horizontal axis: Length of the object. Vertical axis: Height of the object. Red color: Selected points. Blue color: Unselected points.

$\forall x_i \in \left(\frac{a-\varepsilon}{k}, \frac{a}{k}\right)$, we consider $x_i \approx \frac{a}{k}$ and iteratively place x_i into each bin in descending order according to y_i.

The significance of dividing is that, when we have approximated the length $x_i \approx \frac{a}{k}$ and set the sub-bin length to $\frac{a}{k}$, we no longer need to consider the length x_i of each item when packing it into the sub-bin. Then the problem becomes a 1D bin packing problem, where we only need to focus on the bin length and the height y_i.



Hình 39: Items when lining up to their bin approximations

III. Algorithm Implementation

III.1. Technologies Used

Following the internship company's guidance, I implemented all the mentioned projects in Python, using only basic computational libraries like NumPy, Numba, and graphical libraries like NetworkX, Matplotlib.

- **NumPy:** NumPy is an open-source library for the Python programming language, widely used in the fields of data science and scientific computing. Its name is derived from "Numerical Python". NumPy provides powerful data structures such as arrays and matrices, along with a rich set of mathematical functions for manipulating and computing on these data. With NumPy, scientists and engineers can perform complex numerical computations efficiently and quickly, thus supporting data analysis, simulation, and the development of complex algorithms. NumPy serves as the foundation for many other libraries in Python such as SciPy, Pandas, and Matplotlib, contributing to making Python a powerful language in the fields of data science and machine learning.
- **Numba:** Numba is an open-source library for Python designed to accelerate the execution speed of Python code by compiling it into machine code at runtime. Using Just-in-Time (JIT) compilation, Numba enables Python functions to achieve speeds nearly equivalent to low-level programming languages like C or Fortran without much alteration in the original source code. This is particularly useful in applications involving data science, scientific computing, and machine learning, where performance is a critical factor. Numba integrates well with popular libraries like NumPy, helping optimize operations on arrays and matrices efficiently. By simplifying the process of accelerating Python code, Numba becomes a powerful tool for developers and researchers in improving the performance of complex computational applications.
- **Matplotlib:** Matplotlib is an open-source library for Python widely used to create charts and visualizations of data. This library provides flexible and powerful tools for creating various types of plots such as line plots, bar plots, scatter plots, pie charts, and many others. Matplotlib supports detailed customization of graphic elements, from colors and styles to annotations and labels, allowing users to create visually appealing and professional images. It is highly regarded in the data science and data analysis community for its ease of integration with other libraries such as NumPy, Pandas, and SciPy. Matplotlib is an essential tool for presenting and exploring data, helping researchers and analysts communicate information clearly and effectively.

III.2. Spring embedding

To generalize the Traveling Salesman Problem (TSP) to the Euclidean Traveling Salesman Problem (E-TSP), we use the spring embedding technique to embed the graph into Euclidean space.

Input: The distance matrix A, describing the graph $G = (V, E)$.

Output: A 2D array containing the coordinates of the vertices $vinV$ of the graph G after embedding.

III.3. Solving E-TSP using S.Arora's PTAS

From the result of the spring embedding, we implement the steps of Arora's PTAS in Numba.

Input: The coordinates of the points to be visited, represented as a 2D array:

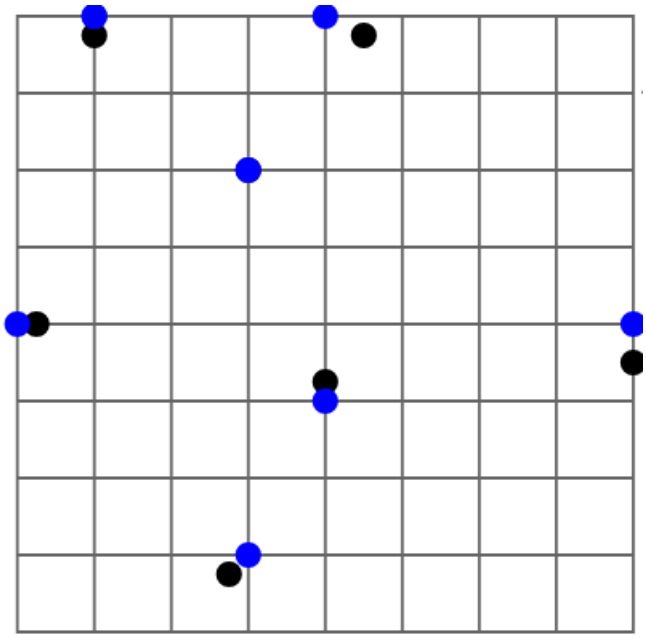
$$X^0 = \{(x_0, x_1)inR^2\}$$

Output: The coordinates of the points to be visited, ordered from front to back, as a 2D array.

III.3.1. Coordinate Data

Since scaling all coordinates by a constant does not change the nature of the problem, we take $X = kX^0$, where $k = O(\frac{1}{\varepsilon})$.

Next, we round the coordinates of each point. Points that are too close together will be merged into a single point.



Hình 40: Coordinates before and after rounding. Black color: Before rounding. Green color: After rounding..

III.3.2. Build 4-tree

In Numba, a 4-Tree can be represented using a tree data structure with `@jitclass`.

However, the `jitclass` support of Numba is quite limited. Therefore, to speed up computation, the tree structure is represented as an `array` in NumPy.

Each row of the `array` contains information about a node in the tree, and each row contains parameters about the corresponding nodes.

	ID	Parent	Is active	Level	Children	AABB
Các nút	1	NULL	TRUE	0	2	...
	2	1	TRUE	1	NULL	...
	3	1	TRUE	1	6	...
	4	1	TRUE	1	NULL	...
	5	1	TRUE	1	NULL	...
	6	3	TRUE	2	NULL	...
	7	3	TRUE	2	NULL	...
	8	3	TRUE	2	NULL	...
	9	3	TRUE	2	NULL	...

Bảng 2: An example of tree representation using a NumPy array

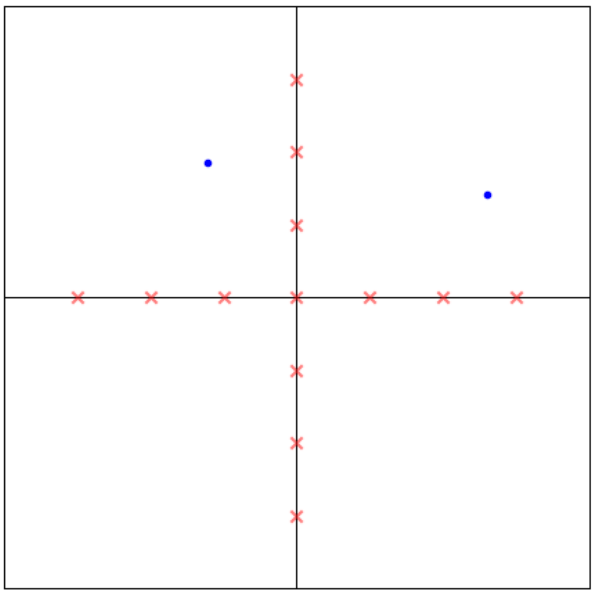
In which:

- ID: Index of the node in the matrix.
- Parent node: ID of the parent node.
- Active: NULL/TRUE - indicates if the row contains data about the node.
- Degree: Degree of the node in the tree.
- Child node: ID of the first child node. We only need to store the ID of the first child node; all child nodes of a node will be arranged next to each other.

  Example: To find the third child node of node a, take 2 + the ID of the first child node of a.
- AABB: Axis-aligned bounding box - Describes the position and size of the square corresponding to this node.

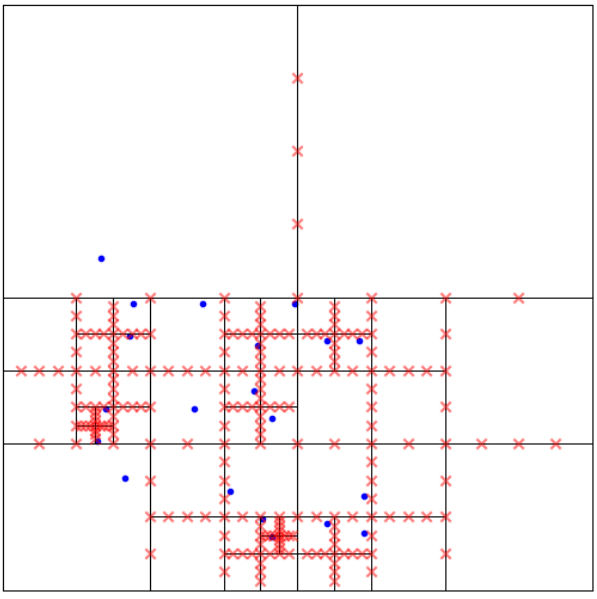
III.3.3. Divide the problem

At each node in the tree, we create portals evenly spaced along the dividing lines. Portals are stored in a 2D array `portallist`.



Hình 41: An example of placing portals. The blue dots represent the points the TSP path needs to pass through. The red 'x' marks represent the portals.

Repeat the step above for every node in the tree.



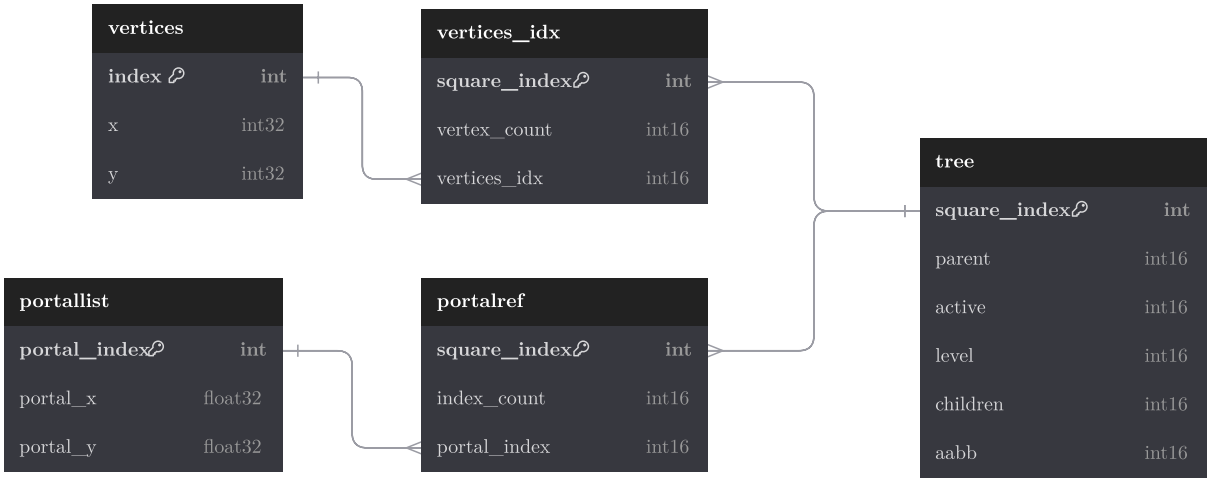
Hình 42: After building the tree and placing portals

The data of the tree is stored in the following arrays:

- `tree`: Information about each node in the tree.
- `vertices`: All points to be visited.
- `vertices_idx`: Information about the points contained in each node.
  - For example, if square number 5 contains points 0, 1, 2 in `vertices`, then at `square_index = 5`, we have `vertex_count = 3`, `vertices_idx = [0, 1, 2]`.

- `portallist`: List of portals.
- `portalref`: List of portals in each square.
  - For example, if square number 5 of the tree contains portals 0, 1, 2 in `portallist`, then at `square_index = 5`, we have `index_count = 3` and `portal_index = [0, 1, 2]`.

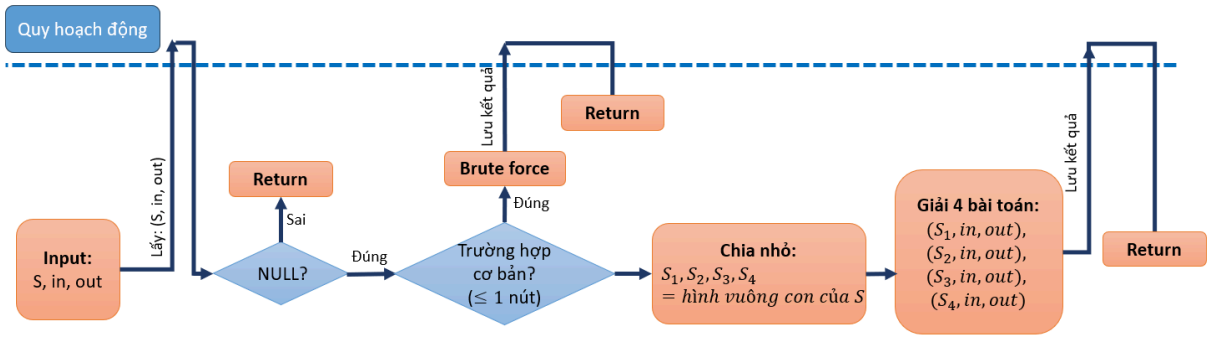
Ta



Hình 43: The structure and relationship between the arrays. The first row of each table is the index of that row in the array. The remaining rows correspond to each element in the section. The arrows describe the relationship between the variables storing the indices..

III.3.4. Dynamic Programming

The dynamic programming scheme can be summarized as follows:



Hình 44: Dynamic programming scheme.

We build the dynamic programming table in the following manner:

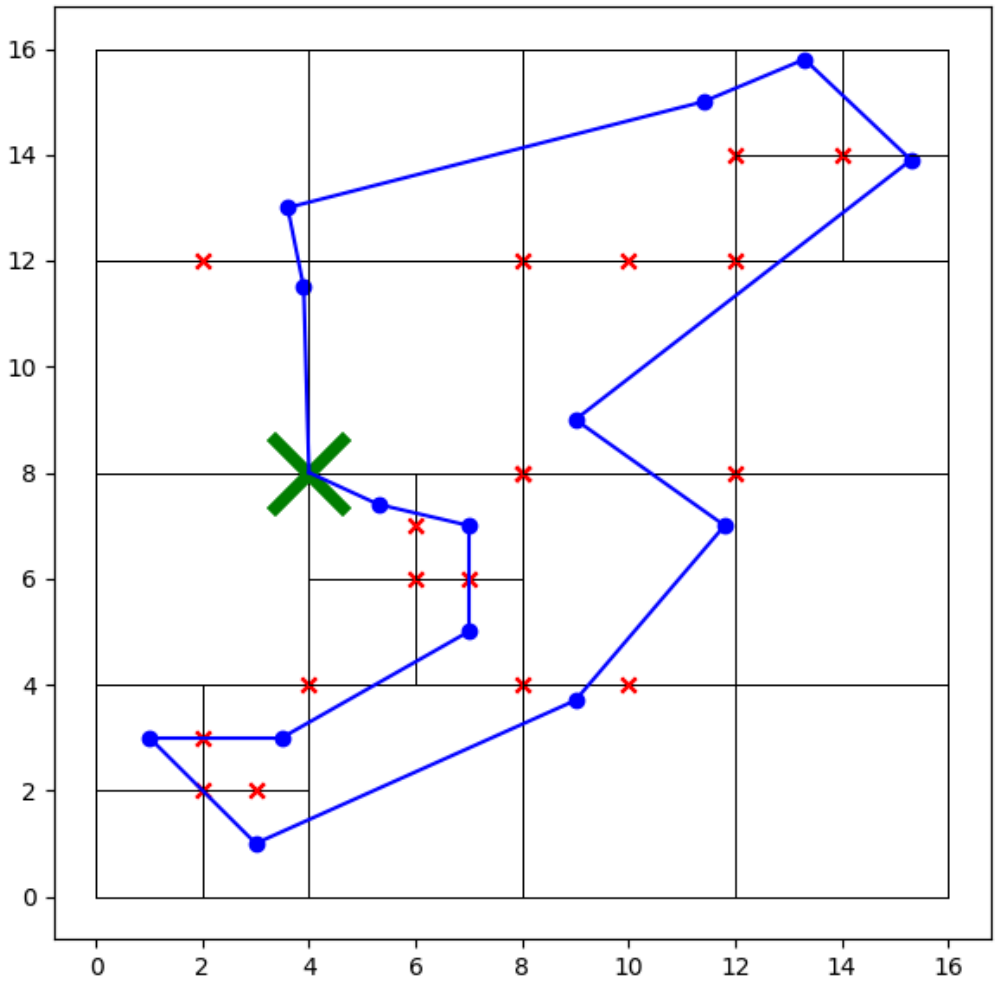
	Node	Subproblem 1	Subproblem 2	Subproblem 3	Subproblem 4	...
Tree nodes	1	...	...	...	...	...
	2	...	...	...	...	...
	3	...	...	...	...	...
	4	...	...	...	...	...
	5	...	...	...	...	...

	Node	Subproblem 1	Subproblem 2	Subproblem 3	Subproblem 4	...
	6	...	...	...	...	...

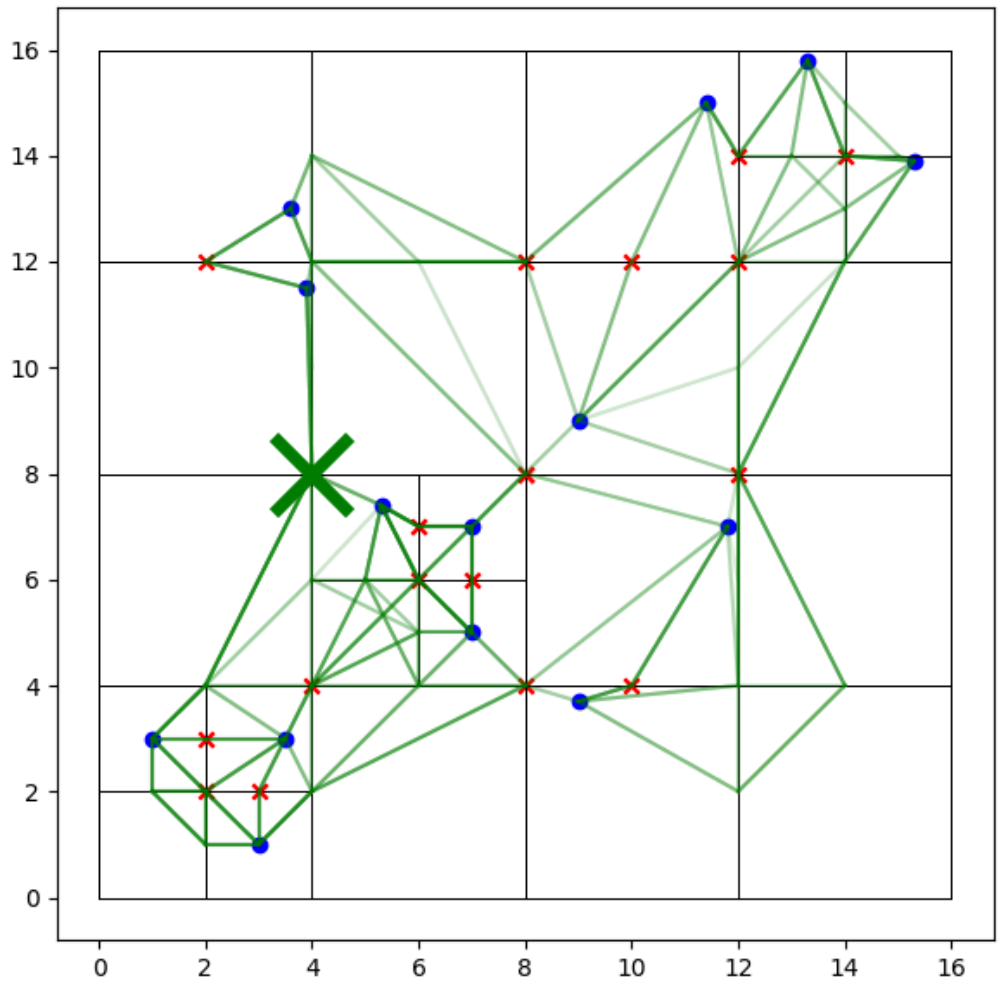
The dynamic programming table will be iteratively built from children to parents.

III.3.5. Obtaining the results

Once we have the dynamic programming table, we proceed to aggregate the results by traversing from the parent node downwards.



Hình 45: TSP tour build from dynamic programming table

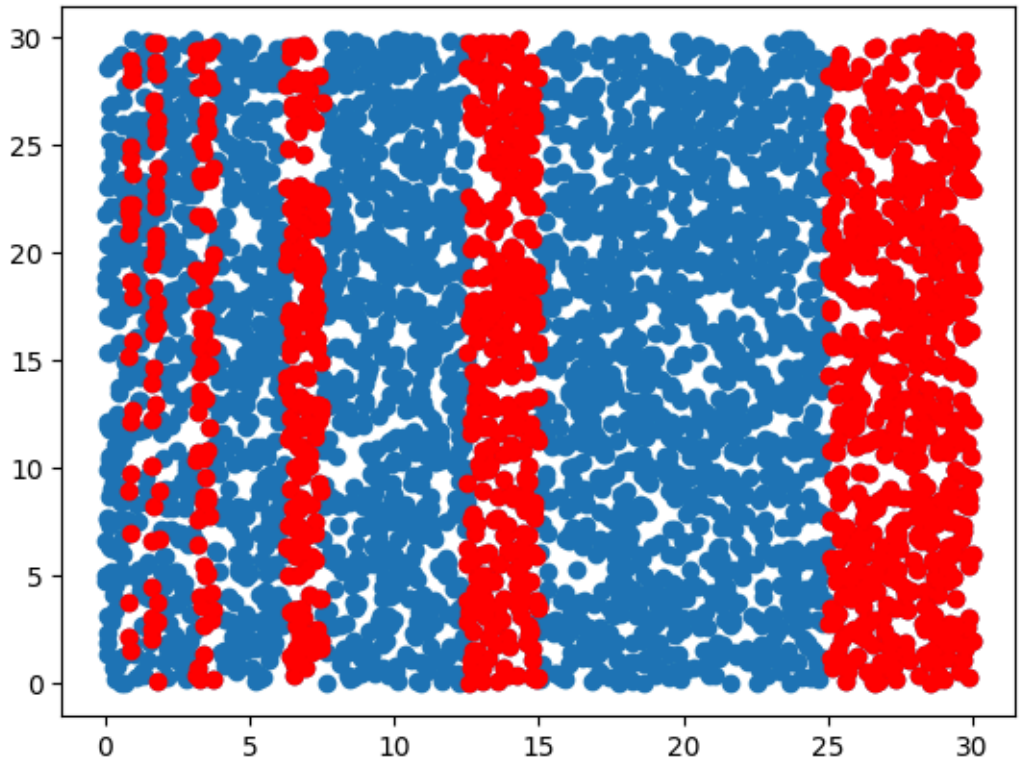


Hình 46: Information contained in dynamic programming table

III.4. Developing PTAS for the 2D Bin packing problem

III.4.1. Data preprocessing

We partition the data in invervals of $\left(\frac{a-\varepsilon}{k}, \frac{a}{k}\right)$



Hình 47: Data partitioning

For each k, we store each item in the interval $\left(\frac{a-\varepsilon}{k}, \frac{a}{k}\right)$ into a max heap. This way, extracting the max value takes $O(1)$ time.

III.4.2. Defining subproblems

Our subproblem will be an instance of the 1D bin packing problem:

Given a set if items I, each with height y_i. Place items into bin B with height Y so that:

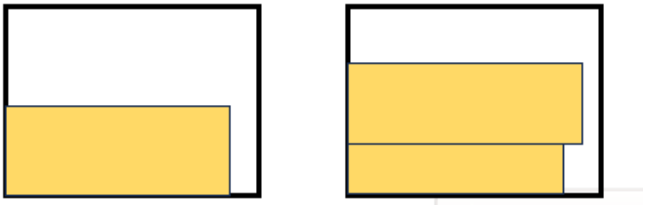
$$\sum_{i \in B} y_i \leq Y$$

III.4.3. Algorithm

Starting with the first bin, iteratively place items in descending order into the bin, until no other items of the same x_i size can be placed.

Hình 48: Step 1: Pack the bin

Hình 49: Step 2: Divide the bin



Hình 50: Bước 3: Solve subproblems in child bins

After successfully filling up a bin, we continue to the next, until there are no items left to be placed.

IV. References

Sanjeev Arora. 1998. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. J. ACM 45, 5 (Sept. 1998), 753–782. https://doi.org/10.1145/290179.290180

Rao, Satish and Warren D. Smith. "Approximating geometrical graphs via "spanners" and "banyans"." Symposium on the Theory of Computing (1998).

Vijay V. Vazirani. 2010. Approximation Algorithms. Springer Publishing Company, Incorporated.

Sanjeev Arora, James R. Lee, and Assaf Naor. 2005. Euclidean distortion and the sparsest cut. In Proceedings of the thirty-seventh annual ACM symposium on Theory of computing (STOC '05). Association for Computing Machinery, New York, NY, USA, 553–562. https://doi.org/10.1145/1060590.1060673

Khoa Toán - Cơ - Tin học
Trường ĐH Khoa học Tự nhiên - ĐHQGHN

NHẬN XÉT THỰC TẬP

Công ty thực tập:

Người hướng dẫn:

Thời gian thực tập:

Họ và tên sinh viên: Phạm Hoàng Hải

1. Ý thức làm việc của sinh viên:

……

……

……

……

……

……

2. Khả năng làm việc/học hỏi:

……

……

……

……

……

……

3. Mức độ hoàn thành những nhiệm vụ được giao:

……

……

……

……

……

……

4. Xác nhận công việc được báo cáo có đúng như công việc được giao?

……

……

……

……

5. Những vấn đề cần góp ý với thực tập sinh/ nhà trường để công việc có thể tiến hành tốt hơn:

……

……

……

……

……

……

……

……

……

……

……

6. Điểm đánh giá (theo thang điểm 10):

……

Hà Nội, ngày …… tháng 08 năm 2023

Xác nhận của người đánh giá

Xác nhận của doanh nghiệp

(kí và đóng dấu)