ĐẠI HỌC BÁCH KHOA HÀ NỘI

TỐI ƯU LẬP KẾ HOẠCH

Bin packing with lower-bound and upper-bound

NGUYỄN HUYỀN SAN

Ngành Khoa học máy tính

GVHD: TS. Bùi Quốc Trung

Chữ ký của GVHD

Bộ môn: Khoa học máy tính

Trường: Công Nghệ Thông Tin và Truyền Thông

HÀ NỘI, 6/2025

Tóm tắt nội dung báo cáo

Báo cáo tổng hợp thông tin và kết quả thực nghiệm thu được trong quá trình giải quyết bài toán tổ hợp NP-hard, đó là bin packing with lower-bound and upper-bound.

Để giải quyết bài toán này, báo cáo tiến hành khảo sát và thực nghiệm với nhiều phương pháp khác nhau bao gồm cả các thuật toán chính xác và các thuật toán gần chính xác. Mỗi thuật toán được đánh giá thông qua hai tiêu chí: chất lượng lời giải và hiệu quả thực hiện (thời gian tính theo mili giây), trên tập dữ liệu đầu vào có quy mô tăng dần.

Trong đó

Các thuật toán chính xác gồm có:

- Quay lui (Backtrack)
- Nhánh cận (Branch and bound)
- Integer programming
- Constraint Programming

Các thuật toán gần chính xác gồm có:

- Tham lam (Greedy)
- Leo đồi (Hill climbing)
- Tôi ủ (Simulated annealing)

Mục tiêu của báo cáo là phân tích, so sánh hiệu năng của từng thuật toán với từng bài toán có quy mô khác nhau, từ đó đưa ra định hướng lựa chọn thuật toán phù hợp theo quy mô bài toán và yêu cầu thực tế. Ngoài ra, báo cáo cũng cung cấp bảng so sánh trực quan giữa các thuật toán nhằm hỗ trợ việc tổng hợp và đánh giá kết quả một cách hệ thống.

MỤC LỤC

DANH MỤC HÌNH VẼ

CHƯƠNG 1. TỔNG QUẢN VỀ BÀI TOÁN

1.1 Yêu cầu bài toán

Cho N đơn hàng, mỗi đơn hàng i có trọng lượng d(i) và giá trị c(i). Có K xe chở hàng để vận chuyển các đơn hàng, mỗi xe k có giới hạn dưới là c1(k) và giới hạn trên là c2(k). Tìm lời giải cho việc xếp các đơn hàng cho các xe sao cho:

- Mỗi đơn hàng chỉ được xếp vào tối đa 1 xe
- Tổng khối lượng các đơn hàng được xếp lên ở một xe phải ở trong khoảng giới hạn dưới và giới hạn trên của nó
- Tổng giá trị các đơn hàng được xếp lên xe là tối đa

Format:

- Input:
  - Dòng đầu: 2 số nguyên dương N và K (1<=N<=1000,1<=K<=1000)
  - Dòng i+1 (i=1,...,N): gồm 2 số nguyên d(i) và c(i) (1<=d(i),c(i)<=100)
  - Dòng N+1+k: gồm 2 số nguyên c1(k) và c2(k) (1<= c1(k)<= c2(k)<= 1000)
- Output:
  - Dòng đầu: gồm số nguyên m (số đơn hàng được xếp lên xe)
  - Dòng i+1 (1,...,m): gồm 2 số nguyên i và b trong đó đơn hàng i được xếp lên xe b

Input	Output
5 2 5 9 7 2 12 6 12 4 7 6 12 14 27 31	5 1 1 2 2 3 2 4 2 5 1

1.2 Mô hình hóa bài toán.

Ta có các biến x_{ij} đại diện cho việc đơn hàng i được xếp lên xe j

Ta có x_{ij}=$\begin{cases}1\\0\end{cases}$ với i=1,..,N và j=1,...,K

x_{ij}=1 có nghĩa là đơn hàng i được xếp lên xe j

x_{ij}= 0 ngược lại nghĩa là đơn hàng i không được xếp lên xe j

Đồng thời ta có các biến y_j đại diện cho việc xe j được phép sử dụng để chở hàng

Ta có y_j=$\begin{cases}1\\0\end{cases}$ với j=1,...,K

y_j=1 có nghĩa là xe j được phép xếp hàng lên

y_j=0 ngược lại có nghĩa là xe j không được được phép xếp hàng lên

- Các ràng buộc của bài toán:
  1) $\sum_{j=1}^{K} x_{ij} \leq 1$ với i=1,...,N
  2) $\sum_{i=1}^{N} x_{ij} \times q_i \geq c1_j \times y_j$ với j=1,...,K
  3) $\sum_{i=1}^{N} x_{ij} \times q_i \leq c2_j \times y_j$ với j=1,...,K
- Hàm mục tiêu của bài toán:
  1) Cực đại hóa : $\sum_{i=1}^{N} \sum_{j=1}^{K} x_{ij} * c_i$

CHƯƠNG 2. CÁC THUẬT GIẢI SỬ DỤNG

2.1 Thuật giải chính xác.

2.1.1 Quay lui (Backtrack)

2.1.1.1. Ý tưởng thuật toán

Backtracking (quay lui) là một kỹ thuật giải quyết bài toán tổ hợp, tối ưu ràng buộc bằng cách duyệt toàn bộ không gian nghiệm một cách có tổ chức.

Thuật toán tìm kiếm lời giải bằng cách xây dựng dần lời giải từng bước và loại bỏ đi những bước đi sai sớm nhất có thể.

Thuật toán hoạt động theo nguyên tắc "thử và sai" (try and error):

- Thử chọn một bước hợp lệ để xây dựng lời giải
- Nếu bước đó dẫn đến lời giải hợp lệ thì tiếp tục
- Nếu không thì quay lai để thử một lựa chọn khác ở bước trước

Các bước thực hiện:

- Xây dựng lời giải từng phần, bắt đầu từ trạng thái rỗng
- Tại mỗi bước, liệt kê các lựa chọn hợp lệ tiếp theo
- Kiểm tra ràng buộc, nếu một lựa chọn vi phạm ràng buộc thì bỏ qua
- Nếu đi hết và tìm được lời giải đầy đủ thì lưu lại kết quả
- Nếu không thể đi tiếp thì quay lui và thử lựa chọn khác

Điểm mạnh của quay lui là:

- Đảm bảo tìm được nghiệm tối ưu nếu kiểm tra hết không gian.
- Có thể kết hợp với nhánh cận (branch and bound) để giảm số lượng nhánh cần xét.

2.1.1.2. Phương án thực hiện

Giải bài toán bằng cách duyệt qua tất cả các phương án gán đơn hàng cho xe một cách có hệ thống.

Ở mỗi bước, ta gán thử 1 đơn hàng cho lần lượt các xe nếu load hiện tại của xe cộng thêm trọng lượng của đơn hàng không vượt quá cận trên của xe. Nếu đơn hàng i được gán cho xe j thì load của j sẽ tăng thêm q[i] (trọng lượng đơn hàng i), đồng thời hàm mục tiêu được tăng thêm c[i] (giá trị đơn hàng i) và chuyển qua xét đơn hàng tiếp theo i+1. Ngược lại

nếu đơn hàng i không gán được cho xe nào thì ta chuyển sang xét cho đơn hàng tiếp theo i+1.

Nếu tất cả đơn hàng đã được xét, ta kiểm tra ràng buộc cận dưới của các xe, nếu các xe có load > 0 đêu thỏa mãn ràng buộc lower_bound<= load <= upper_bound thì đây là một lời giải chấp nhận được và được lưu lại nếu tổng giá trị các đơn hàng được xếp lên xe lớn hơn tổng giá trị các đơn hàng được xếp lên xe của lời giải được lưu trước đó.

Kết quả cuối cùng trả về là nghiệm tối ưu thỏa mãn các ràng buộc.

2.1.1.3. Nhận xét

Thuật toán đảm bảo trả về nghiệm tối ưu toàn cục của hàm mục tiêu, có tính khái quát hóa tốt , dễ áp dụng cho các bài toán tổ hợp (NP-hard). Không cần bộ nhớ lớn, do tìm kiếm lời giải theo chiều sau không yêu cầu dữ lại toàn bộ không gian tìm kiếm, chi phí bộ nhớ là $O(N * K)$.

Tuy nhiên độ phức tạp thời gian là rất lớn $(O(K + 1)^N)$ dẫn đến thời gian chạy tăng nhanh.Đồng thời đối với các bài toán có quá nhiều trạng thái quay lui thường thất bại do độ sâu tìm kiếm quá lớn cộng thêm với giới hạn đệ quy của máy tính, thuật toán chỉ có thể chạy được bới N<30.

2.1.2 Nhánh cận (Branch and bound)

2.1.2.1. Ý tưởng thuật toán

Phát triển từ thuật toán quay lui ta nhận thấy có những nhánh không thể cải thiện hàm mục tiêu, nên nếu cứ tính toán lời giải đầy đủ cho những nhánh như vậy là vô cùng lãng phí thì thời. Từ đó để cải thiện, người ta tiến hành cắt tỉa (prune) những nhánh không thể cải thiện hàm mục tiêu.

Các bước thực hiện:

- Xây dựng lời giải từng phần bắt đầu từ lời giải rỗng, tổ chức không gian tìm kiếm dưới dạng cây
- Ở mỗi bước, tính giá trị cận cho mỗi nút
- Cập nhật lời giải nếu thỏa mãn cận
- Nếu không cắt nhánh đó
- Cập nhật lời giải tốt nhất mỗi khi gặp một lời giải khả dĩ (feasible solution)

2.1.2.2. Phương án thực hiện.

Tương tự thuật toán quay lui ta có:

Ở mỗi bước, ta gán thử 1 đơn hàng cho lần lượt các xe nếu load hiện tại của xe cộng thêm trọng lượng của đơn hàng không vượt quá cận trên của xe. Nếu đơn hàng i được gán cho xe j thì load của j sẽ tăng thêm q[i] (trọng lượng đơn hàng i), đồng thời hàm mục tiêu được tăng thêm c[i] (giá trị đơn hàng i) và chuyển qua xét đơn hàng tiếp theo i+1. Ngược lại nếu đơn hàng i không gán được cho xe nào thì ta chuyển sang xét cho đơn hàng tiếp theo i+1.

Tuy nhiện sau mỗi bước gán, ta kiểm tra nhanh ước lượng giá trị lơn nhất có thể của tổng giá trị các đơn hàng được xếp lên xe theo phương án hiện tại (giả sử tất cả các đơn hàng còn lại đều được gán). Nếu tổng giá trị các đơn hàng được xếp lên xe ở phương án hiện tại cộng với giá trị tối đa các đơn hàng còn lại vẫn không tốt hơn được nghiệm tốt nhất được lưu, thì ta tiến hành cắt tỉa nhánh này.

Nếu tất cả đơn hàng đã được xét, ta kiểm tra ràng buộc cận dưới của các xe, nếu các xe có load > 0 đêu thỏa mãn ràng buộc lower_bound<= load <= upper_bound thì đây là một lời giải chấp nhận được và được lưu lại nếu tổng giá trị các đơn hàng được xếp lên xe lớn hơn tổng giá trị các đơn hàng được xếp lên xe của lời giải được lưu trước đó.

Kết quả cuối cùng trả về là nghiệm tối ưu thỏa mãn các rằng buộc.

2.1.2.3. Nhận xét

Thuật toán nhánh cận là 1 phiên bản cải tiến của quay lui với quá trình cắt tỉa, kết quả của thuật toán cũng là nghiệm tối ưu của bài toán. Nhờ sử dụng cận trên, chỉ mở rộng những nhánh có tiềm năng tối ưu từ đó có thể cải thiện thời gian tính toán. Chi phí về bộ nhớ nhỏ với O(N*K).

Tuy thuật toán có cải thiện thời gian tính toán do có cắt tỉa bớt những nhánh không thể cải thiện, tuy nhiên độ phức tạp về thời gian về mặt lý thuyết vẫn là tương đối lớn khi bằng với quay lui là $(O(K+1)^N)$. Chỉ có thể cải thiện thời gian tính toán với các giá trị đầu vào nhỏ, còn lại là không đáng kể. Đòng thời để việc cắt tỉa hiệu quả cũng cần phải chọn hàm cận tối ưu để cận không quá lỏng, từ đó khiến thuật toán gần như quay lui.

2.1.3 Integer Programming

2.1.3.1. Ý tưởng thuật toán

Integer Programming (IP), hay quy hoạch nguyên trong đó một hoặc nhiều biến nguyên quyết định trong mô hình được ràng buộc phải nhân giá trị nguyên .

Biểu diễn bài toán bằng hệ phương trình tuyến tính:

- Hàm mục tiêu: max/min tuyến tính theo biến nguyên.
- Ràng buộc:hệ bất đẳng thức tuyến tính
- Các biến: có thể là số nguyên hoặc nhị phân

Ý tưởng chung: tối ưu hàm tuyến tính trong tập rời rạc được xác định bởi ràng buộc tuyến tính.

Mô hình IP có dạng tổng quát như sau:

Min $c^T x$

Subject to : $Ax \leq b$, $x \in Z^n$

Trong một số trường hợp, không phải tất cả các biến cần nguyên – mô hình này được gọi là Mixed Integer Programming (MIP)

2.1.3.2. Phương án thực hiện

Như đã nêu trong phần mô hình hóa bài toán [1.2]

Bên trong class xây dựng các biến toàn cục như sau:

- orders_num : số lượng order
- vehicles_num: số luọng xe chở hàng
- quantity :list trọng lượng của các đơn hàng
- cost : list chứa giá trị của các đơn hàng
- lower_bound : list chứa giá trị cận dưới của các xe chở hàng
- upper_bound : list chứa giá trị cận trên của các xe chở hàng

Xây dựng các biến quyết định :

- $x_{ij} \epsilon \{0 , 1\}$: Biến nhị phân bằng 1 nếu đơn hàng i được xếp lên xe j
- $y_j \epsilon \{0 , 1\}$: Biến nhị phân 1 nếu xe j được phép xếp hàng lên xe

Với $i \epsilon \{1, \dots , N\}$ và $j \epsilon \{1, \dots , K\}$

Các ràng buộc:

- $\sum_{j=1}^{K} x_{ij} \leq 1 \; \forall \; i \epsilon \{1, \dots , N\}$
- $\sum_{i=1}^{N} x_{ij} \; quantity_i \leq \; upper_{bound_j} \forall \; j \epsilon \{1, \dots , N\}$ và $y_j = 1$
- $\sum_{i=1}^{N} x_{ij} \; quantity_i \geq \; lower_{bound_j} \forall \; j \epsilon \{1, \dots , N\}$ và $y_j = 1$
- $\sum_{i=1}^{N} x_{ij} \; quantity_i = 0 \; \forall \; j \epsilon \{1, \dots , N\}$ và $y_j = 0$

Hàn mục tiêu : max $(\sum_{i=1}^{N} \sum_{j=1}^{K} x_{ij} cost_{ij})$

2.1.3.3. Thực hiện cài đặt

Sử dụng bộ giải SCIP của thư viện ortools.linear_solver với pywraplp

Tạo các biến số nị phân x[i][j], với phương thức BoolVar()

Tạo các ràng buộc với Add():

```
Add(sum(x[i][j] for j in range(self.vehicles_num))<=1)
Add(sum(x[i][j]*self.quantiy[i] for i in
range(self.orders_num)) <= self.upper_bound[j] if y[j]
else 0)
Add(sum(x[i][j]*self.quantiy[i] for i in
range(self.orders_num)) >= self.lower_bound[j] if y[j]
else 0)
```

Tạo lập hệ số hàm mục tiêu là Maximize:

```
Maximize(sum(x[i][j]*self.cost[i] for i in
range(self.orders_num) for j in
range(self.vehicles_num)))
```

2.1.3.4. Nhận xét.

Xử lý tốt và chính xác với các test case nhỏ (N<100) đảm bảo tìm được nghiệm tối ưu. Dễ dàng xây dựng một khi có mô hình hóa cụ thể của bài toán. Dễ dàng thêm các ràng buộc, không cần viết lại thuật toán mà chỉ cần hiệu chỉnh mô hình. IP solver có sẵn các kỹ thuật cắt mặt mạnh (cutting-planes), heuristic nội tại, phân tách nhánh tự động… giúp tìm

nhanh nghiệm tốt.Có khả năng khai thác song song, trị giá biên (reduced cost) từ LP relaxation để cắt nhánh hiệu quả

Tuy nhiên việc mô hình hóa bài toán sao cho hiệu quả và hợp lý cũng là một vấn đề. Quan trọng nhất là vấn đề về ràng buộc nhanh chóng phình to với test case tăng. Về lý thuyết bài toán là NP-hard nên số biến nhị phân K x N và số ràng buộc $(O(K+N))$ lớn sẽ khiến solver nổ ra rất nhiều nhánh. Việc xây dựng ma trận ràng buộc lớn tốn thời gian và bộ nhớ. Mỗi lần thay đổi dữ liệu hoặc bổ sung biến/ràng buộc phải rebuild hoàn toàn mô hình. Thuật toán chỉ phù hợp cho các test case từ nhỏ đến vừa.

2.1.4 Constraint Programming

2.1.4.1. Ý tưởng thuật toán

Constraint programming là một phương pháp giải đúng bài toán tối ưu:

- Dùng các ràng buộc để tỉa không gian tìm kiếm : loại bỏ các giá trị thừa ra khỏi miền giá trị của các biến.
- Phân nhánh và tìm kiếm quay lui: Chia không gian tìm kiếm ra thành các không gian con
    - Liệt kế các giá trị cho biến được lựa chọn
    - Phân hoạc tập giá trị của mỗi biến được lựa chọn thành 2 hoặc nhiều tập con

Biểu diễn bài toán bằng các biến rời rạc và ràng buộc logic:

- Biến: rời rạc (IntVar, BoolVar) với các miền giá trị rời rạc.
- Ràng buộc: logic, đại số, tổ hợp, toàn cục (global constraint) như AllDifferent,...
- Tìm kiếm ràng buộc thỏa mãn mà không cần biểu diễn tuyến tính

Ý tưởng chung: Duyệt không gian lời giải bằng ràng buộc và chiến lược tìm kiếm để tìm ra lời giải hợp lệ.

2.1.4.2. Phương án thực hiện

Như đã nêu trong phần mô hình hóa bài toán [1.2]

Bên trong class xây dựng các biến toàn cục như sau:

- orders_num : số lượng order
- vehicles_num: số luọng xe chở hàng
- quantity :list trọng lượng của các đơn hàng
- cost : list chứa giá trị của các đơn hàng
- lower_bound : list chứa giá trị cận dưới của các xe chở hàng
- upper_bound : list chứa giá trị cận trên của các xe chở hàng

Xây dựng các biến quyết định :

- $x_{ij} \epsilon \{0 , 1\}$: Biến nhị phân bằng 1 nếu đơn hàng i được xếp lên xe j
- $y_j \epsilon \{0 , 1\}$: Biến nhị phân 1 nếu xe j được phép xếp hàng lên xe

Với $i \in \{1, \dots, N\}$ và $j \in \{1, \dots, K\}$

Các ràng buộc:

- $\sum_{j=1}^{K} x_{ij} \leq 1 \; \forall \, i \in \{1, \dots, N\}$
- $\sum_{i=1}^{N} x_{ij} \, quantity_i \leq \; upper_{bound_j} \forall \, j \in \{1, \dots, N\}$ và $y_j = 1$
- $\sum_{i=1}^{N} x_{ij} \, quantity_i \geq \; lower_{bound_j} \forall \, j \in \{1, \dots, N\}$ và $y_j = 1$
- $\sum_{i=1}^{N} x_{ij} \, quantity_i = 0 \; \forall \, j \in \{1, \dots, N\}$ và $y_j = 0$

Hàn mục tiêu : max $(\sum_{i=1}^{N} \sum_{j=1}^{K} x_{ij} cost_{ij})$

2.1.4.3. Thực hiện cài đặt

Sử dụng bộ giải cp_model từ thư viện ortools.sat.python

Tạo các biến nhị phân x[i][j] và y[j] với phương thức NewBoolVar()

Thêm các ràng buộc với model.Add() – Sử dụng OnlyEnforceIf()

để thiết lập ràng buộc có điều kiện

```
• model.Add(sum(x[i][j] for j in range(self.vehicles_num))
  <= 1)r
• for j in range(self.vehicles_num):
    o load_j= sum(x[i][j] * self.quantity[i] for i in
      range(self.orders_num))
    o model.Add(load_j <=
      self.upper_bound[j]).OnlyEnforceIf(y[j])
    o model.Add(load_j >=
      self.lower_bound[j]).OnlyEnforceIf(y[j])
    o model.Add(load_j == 0).OnlyEnforceIf(y[j].Not())
```

Tạo lập hệ số hàm mục tiêu Maximize()

```
• Maximize(sum(x[i][j]*self.cost[i] for i in
  range(self.orders_num) for j in
  range(self.vehicles_num)))
```

2.1.4.4. Nhận xét.

Mô hình hóa trực quan: Ràng buộc "mỗi đơn một xe", "tải trọng trong khoảng hoặc bằng 0" đều được viết thẳng dưới dạng ràng buộc tuyến tính hoặc ràng buộc "OnlyEnforceIf", rất rõ ràng và gần gũi với phát biểu bài toán.

Tự động hóa mạnh mẽ: Solver CP-SAT của OR-Tools sẽ tự động lựa chọn chiến lược. Cắt tỉa nhanh nhờ propagation (forward checking)

Dễ mở rộng: Khi cần thêm ràng buộc mới (ví dụ: thời gian giao hàng, phụ phí khởi động xe, hạn mức công suất…), chỉ cần thêm biến và ràng buộc vào model, không phải viết lại cả thuật toán.

Tuy nhiên thuật toán có thể không xử lý tốt bài toán có hàm mục tiêu tuyến tính phức tạp. Đồng thời dễ chuyển thành duyệt cạn nếu không có chiến lược tốt và không đảm bảo tối ưu nếu không cài đặt chi tiết hàm mục tiêu.

Quan trọng nhất là gặp vấn đề giống IP, khi ràng buộc nhanh chóng phình to với test case tăng. Khởi tạo model tốn kém: Việc xây dựng toàn bộ biến và ràng buộc (vòng lặp lồng) một lần cũng tốn thời gian, đôi khi chiếm một phần đáng kể thời gian chạy.

Với số đơn hàng N lớn (vài trăm, vài nghìn) hoặc số xe K cũng lớn, số biến nhị phân $N \times K$ có thể lên đến hàng chục ngàn, dẫn đến vượt ngưỡng thời gian giải (TLE). Vấn đề nghiêm trọng nhất của CP đó chính là thời gian giải bài toán bất kể quy mô thời khá lâu so với IP rất dễ xảy ra tình trạng vượt ngưỡng thời gian giải nếu bài toán áp dụng yêu cầu cao về tốc độ tính toán (cụ thể là trong bài toán tổ hợp NP-hard đang được xét trong báo cáo)

2.2 Thuật giải gần chính xác

2.2.1 Thuật toán tham lam (Greedy)

2.2.1.1. Ý tưởng thuật toán

Là một thuật toán heuristics, ở mỗi bước thuật toán sẽ xem xét lựa chọn phương án tốt nhất ở thời điểm hiện tại theo một tiêu chí với hy vong sẽ ra nghiệm gần tối ưu toàn cục. Tuy nhiên các phương án đã được lựa chọn sẽ không được xem xét lại nên nghiệm cuối có thể không phải nghiệm tối ưu, bù lại độ phức tạp thời gian của thuật toán sẽ thấp hơn và có thể chạy với các bộ dữ liệu lớn.

Các bước thực hiện:

- Khởi tạo lời giải rỗng
- Chọn phần tử tốt nhất tạo mỗi bước gán theo một tiêu chí nào đó
- Nếu thỏa mãn ràng buộc thêm phần tử đó vào lời giải
- Nếu không thỏa mãn thì xét phần tử xếp sau theo tiêu chí ưu tiên
- Khi không cong phần tử nào để lựa chọn thì trả về lời giải cuối cùng

2.2.1.2. Phương án thực hiện

Trong bài ta thực hiện thuật toán tham lam theo phương án sau:

- Ta kết hợp tất cả các tổ hợp chiến lược ưu tiên gồm có:
    - Ưu tiên đới với đơn hàng : giá trị, khối lượng và tỷ lệ giá trị/khối lượng
    - Ưu tiên đối với xe hàng : cận dưới, cận trên khỏng tải trọng của xe
- Chạy thuật toán với từng tổ hợp ưu tiên và trả về nghiệm tối nhất thu được từ các chiến lược
- Ta chia quá trình xếp hang lên xe làm 2 giai đoạn:
    - Giai đoạn đầu ta sẽ xếp hàng lên xe sao cho các xe đặt đủ cận dưới theo tổ hợp ưu tiên đang được lựa chọn. Sau khi xếp xong kiểm tra lại các xe nếu không đạt đủ cận dưới bị loại bỏ khỏi danh sách các xe được xếp hàng lên đồng thời

lấy các hàng đã xếp lên xe này xuống để chuẩn bị cho sự sắp xếp ở giai đoạn 2

- Giai đoạn 2 ta sẽ xếp tiếp hàng lên danh sách các xe sau khi đã loại bỏ các xe không thỏa mãn cận dưới ở giai đoạn 1. Các đơn hàng còn lại sau khi thực hiện giai đoạn 1 được xếp lên các xe theo tiêu chí ưu tiên được lựa chọn, sao cho tải trọng của các xe được xếp không vượt quá cận trên của nó. Kết quả thu được sau giai đoạn này chính là nghiệm cuối của thuật toán theo tiêu chí được lựa chọn. Kết quả này sau đó được đem đi so sánh với kết quả tốt nhất trong các tiêu chí ưu tiên đã lựa chọn, nếu lớn hơn thì kết quả sẽ được gán làm nghiệm của bài toán.

2.2.1.3. Nhận xét.

Ý tưởng của thuật toán khá đơn giản dễ cài đặt, chỉ cần một vòng lặp chọn phần tử tốt nhất tạo mỗi bước gán, không cần duyệt toàn bộ không gian. Tốc độ tính toán tương đối nhanh khi độ phức tạp chỉ vào khoảng $O(N \times K + N \log N)$, phù hợp với các bài toán có bộ dữ liệu đầu vào lớn luôn trả về kết quả thỏa mãn ràng buộc trong tức thì. Độ phức tạp về thời gian là tương đối thấp khi chỉ lưu trữ lời giải hiện tại.

Tuy nhiên thuật toán không đảm bảo trả về lời giải tối ưu toàn cục.

Thuật toán chỉ tối ưu với một tiêu chí duy nhất không xét phối hợp nhiều yếu tố, đồng thời không thể quay lui nếu chọn sai từ đó dẫn đến chỉ có thể đạt tối ưu cục bộ. Greedy chỉ quan tâm đến đơn hàng, không tối ưu phân phối giữa các xe nên có thể một xe bị quá tải, trong khi xe khác rảnh. Phải thêm bước loại bỏ xe không đủ lower-bound, làm mất một số đơn đã gán, dẫn đến giảm chất lượng nghiệm.

2.2.2 Thuật toán leo đồi (Hill Climbing)

2.2.2.1. Ý tưởng thuật toán

Thuật toán leo đồi là một thuật toán heuristic, là một phương pháp tối ưu hóa cục bộ, tại mỗi bước chọn giải pháp hàng xóm tốt nhất để cải thiện lời giải hiện tại, với mục tiêu đạt cực trị toàn cục.

Các bước thực hiện:

- Khởi tạo từ một lời giải ban đầu (ngẫu nhiên hoặc heuristic)
- Sinh các hàng xóm của lời giải hiện tại
- Chọn hàng xóm tốt nhất (có hàm đánh giá tốt hơn)
- Di chuyển đến hàng xóm đó nếu nó cải thiện lời giải
- Dừng khi không cong hàng xóm nào tốt hơn (đạt cực trị cục bộ)

2.2.2.2. Phương án thực hiện

Trong bài toán xếp hàng lên xe, thuật toán được dùng để tìm phương án xếp hàng lên các xe sao cho thỏa mãn các ràng buộc và tối đa giá trị của tổng các đơn hàng được xếp lên xe.

Khởi tạo phương án ban đầu bằng thuật toán tham lam sử dụng tiêu chí ưu tiên theo giá trị đơn hàng.

Chiến lược lựa chọn hàng xóm: gồm 3 phương án

- Gán thêm đơn hàng chưa được xếp cho xe ngẫu nhiên có khả năng (hàm add_unassigned)
- Chuyển ngẫu nhiên 1 đơn hàng đã được xếp sang cho 1 xe ngẫu nhiên khác có khả năng (hàm generate_neighboor_move)
- Chuyển ngẫu nhau vị trí của 2 đơn hàng ở 2 xe khách nhau cho nhau (hàm generate_neighboor_swap)

Tạo hàng xóm mới bằng cách kết hợp phương án 1 với 1 trong hai phương án còn lại do mục tiêu là cải thiện giá trị hàm mục tiêu nên cần có thêm đơn hàng được xếp vào.

Khởi tạo lại ngẫu nhiên (Random restart) : nếu bị kẹt ở cực trị địa phương, tạo lại lộ trình mới.

Các tham số trong bài:

- Max_restart: số lần thử tạo lại cách sắp xếp mới từ cách săos xếp khởi tạo (=5)
- Max_iterations: số lần lặp để tìm hàng xóm không cải thiện trước khi restart(=850)
- Time_limit: thời gian chạy tối đa(=1.0s).
- Assignment (các hàm tạo hàng xóm) :cách sắp xếp của lời giải hiện tại
- Load (các hàm tạo hàng xóm): tải trọng của cách xe theo lời giải hiện tại.

2.2.2.3. Nhận xét.

Thuật toán Hill Climbing mang lại một giải pháp rất nhanh chóng và dễ dàng triển khai, bởi nó chỉ đòi hỏi khai báo một hàm sinh lân cận đơn giản và lặp cho đến khi không còn cải thiện. Nhờ bắt đầu từ một nghiệm khởi tạo tốt (thường là Greedy), Hill Climbing có thể tiếp cận nhanh các đỉnh cao trong không gian nghiệm cục bộ và thường thu được cải thiện đáng kể ngay trong vài chục đến vài trăm vòng lặp đầu tiên. Độ phức tạp của mỗi vòng lặp là O(M), với M là số láng giềng được sinh, và số vòng lặp I thường rất nhỏ so với kích thước bài toán, nên tổng thời gian chạy thường chỉ tính bằng mili- đến vài chục mili-giây ngay cả khi N, K lên đến hàng trăm.

Tuy nhiên, Hill Climbing cũng có những hạn chế cố hữu: do luôn chỉ chấp nhận những bước đi tốt hơn (hoặc bước đi đầu tiên cải thiện), thuật toán rất dễ bị kẹt tại một đỉnh cục bộ mà không thể tìm ra lối thoát để tiến tới nghiệm toàn cục. Việc chọn bộ láng giềng phù hợp (độ lớn, phân bố, tỉ lệ các phép phép toán) và khởi tạo một giải pháp seed đủ tốt là rất quan trọng để giảm thiểu vấn đề này.

Hill Climbing với Restart là một phương pháp cực kỳ linh hoạt và nhanh chóng khi vẫn giữ được các ưu điểm của Hill Climbing. Restart nhiều lần giúp khai thác tốt hơn không gian nghiệm, giảm khả năng kẹt ở đỉnh cục bộ.

Tuy nhiên chất lượng nghiệm lệ thuộc vào số lần restart và seed ban đầu. Mỗi lần restart lại tốn thêm thời gian, nên tổng thời gian có thể lên khá cao nếu restart quá nhiều. Vẫn không đảm bảo tìm được nghiệm toàn cục nếu không đủ restart hoặc không đủ bước cục bộ. Độ phức tạp thời gian vào khoảng O(R×I×M) với I bước mỗi bước sinh ra M láng giềng và R lần restart.

2.2.3 Thuật toán tôi ủ (Simulated Anealing)

2.2.3.1. Ý tưởng thuật toán

Là 1 thuật toán metaheuristic, mô phỏng quá trình luyện kim. Giải thuật này được áp dụng trong tối ưu hóa để tìm ra tối ưu toàn cục.

Thuật toán là sự kết hợp của 2 kỹ thuật "hill climbing" và "pure random walk". Kỹ thuật hill climbing sẽ giúp tìm giá trị cực trị toàn cục, còn kỹ thuật pure random walk giúp tăng hiệu quả tìm kiếm giá trị tối ưu

Nó cố gắng thoát khỏi cực trị địa phương bằng cách chấp nhận lời giải tệ hơn dựa trên xác suất.

Thuật toán gồm các siêu tham số sau : Nhiệt độ ban đầu, hệ số làm nguội, thời gian chạy.

Các bước thực hiện:

- Sinh ra một giải pháp ngẫu nhiên
- Tính chi phí của giải pháp đó bằng một hàm chi phí
- Sinh ra một giải pháp lân cận ngẫu nhiên và tính chi phí của nó
- So sánh chi phí giữa giải pháp mới và giải pháp cũ
- Nếu chi phí của giải pháp mới tốt hơn giải pháp cũ thì chọn giải pháp mới, ngược lại chọn giải pháp mới với tỷ lệ $p = e^{\frac{\Delta}{T}}$ với $\Delta = f(s') - f(s)$
- Giảm nhiệt độ theo hàm làm nguội (cooling shedule): $T = \alpha * T$ $(0 < \alpha < 1)$

2.2.3.2. Phương án thực hiện

Khởi tạo phương án ban đầu bằng thuật toán tham lam sử dụng tiêu chí ưu tiên theo giá trị đơn hàng.

Chiến lược lựa chọn hàng xóm: gồm 3 phương án

- Gán thêm đơn hàng chưa được xếp cho xe ngẫu nhiên có khả năng (hàm add_unassigned)
- Chuyển ngẫu nhiên 1 đơn hàng đã được xếp sang cho 1 xe ngẫu nhiên khác có khả năng (hàm generate_neighboor_move)
- Chuyển ngẫu nhau vị trí của 2 đơn hàng ở 2 xe khách nhau cho nhau (hàm generate_neighboor_swap)

- Bỏ bất 1 đơn hàng bất kỳ xuống khỏi xe nếu không vi phạm ràng buộc

Chọn ngẫu nhiên các phương án theo tỷ lệ (0.3,0.3,0.2,0,2) lặp lại quá trình chọn tôi ủ trong time_limit.
Chấp nhận lời giải hàng xóm:
Gỉa sử Δ=phương án hàng xóm – phương án hiện tại
Nếu lời giải mới tốt hơn ($\Delta > 0$): Chấp nhận
Ngược lại chấp nhận với xác suất : $e^{\frac{\Delta}{T}}$
Lúc đầu (T cao) dễ chấp nhận các lời giải tệ hơn từ giúp có thể thoát khỏi tối ưu cục bộ.
Khi T giảm dần, thuật toán trở nên tham lam hơn
Các siêu tham số của bài toán:

- Nhiệt độ đầu :T_0=2000
- Hệ số làm nguội: α=0.995
- Thời gian tôi ủ: time_limit=0.8 (s)

2.2.3.3. Nhận xét

Simulated Annealing (SA) là một metaheuristic mô phỏng quá trình tôi luyện vật liệu, bắt đầu từ một nghiệm khởi tạo (thường là giải pháp Greedy), sau đó lặp qua các bước sinh nghiệm lân với xác suất chấp nhận nghiệm kém hơn tỉ lệ nghịch với mức độ kém đi so với nghiệm hiện tại và nhiệt độ T. Nhờ cơ chế "chấp nhận bước tệ" này, SA có khả năng thoát khỏi các đỉnh cục bộ và tiến gần hơn đến nghiệm toàn cục so với Hill Climbing thuần túy.Tuy nhiên nghiệm thu được có thể vẫn không phải là tối ưu toàn cục. Đồng thời SA cũng có nhược điểm là tốn nhiều thời gian hơn do phải sinh và đánh giá nhiều láng giềng, và phụ thuộc mạnh vào tham số: nhiệt độ ban đầu T_0 , hàm làm nguội (cooling schedule), và số vòng lặp I. Độ phức tạp thời gian xấp xỉ $O(I \times M)$, với M là số láng giềng mỗi bước (thường $O(1)$–$O(K)$), do đó cần cân bằng giữa chất lượng nghiệm và thời gian chạy bằng cách điều chỉnh I và tốc độ làm nguội. SA rất phù hợp khi ta cần nghiệm có chất lượng cao và có thể nhường chỗ cho một chút dư thừa về thời gian tính toán.

CHƯƠNG 3. THỰC NGHIỆM VÀ ĐÁNH GIÁ

3.1 So sánh kết quả của các thuật toán.

TC		Backtrack		B&B		ILP		CP		Greedy		Hill Climbing		Simulated Annealing	
		best	time	best	time	best	time	best	time	best	time	best	time	best	time
5 2	27	27	41	27	24	27	126	27	1095 (TLE)	27	45	27	54	27	844
10 2	57	57	118	57	39	57	122	57	1130 (TLE)	57	46	57	120	57	843
50 3	257	-	-	257	44	257	148	257	1477 (TLE)	257	48	257	128	257	845
100 5	1065	-	-	-	-	1139	531	1139	2250 (TLE)	1139	48	1065	71	1065	844
200 10	2037	-	-	-	-	2102	1524 (TLE)	2102	3052 (TLE)	2081	53	2089	285	2086	945
300 10	2971	-	-	-	-	-	-	-	-	3033	58	2995	305	2995	847
500 50	5217	-	-	-	-	-	-	-	-	5227	89	5221	591	5222	849
800 80	8536	-	-	-	-	-	-	-	-	8559	135	8549	801	8544	849
900 90	9410	-	-	-	-	-	-	-	-	9526	165	9502	717	9501	850
1000 100	10368	-	-	-	-	-	-	-	-	10380	178	10377	901	10376	849

Bảng 1. Bảng so sánh kết quả chạy thực nghiệm của các thuật toán

3.2 Phân tích và kết luận.

3.2.1 Thuật toán chính xác.

Các giải thuật chính xác như quay lui (Backtrack), nhánh cận (Branch and Bound), lập trình nguyên (Integer Programming – IP) và lập trình ràng buộc (Constraint Programming – CP) được thiết kế để tìm lời giải tối ưu toàn cục bằng các duyệt toàn bộ hoặc một không gian tìm kiếm một cách hệ thống. Trong thực nghiệm nhóm này cho thấy khả năng trả về nghiệm tối ưu với các bài toán có quy mô vừa và nhỏ ($N \leq 200$):

- Trong nhóm thuật toán chính xác, quay lui cho kết quả chính xác tuyệt đối nhưng chỉ áp dụng cho những bài toán nhỏ ($N \leq 10$), do thời gian chạy tăng nhanh không kiểm soát theo hàm mũ.
- Nhánh cận cải thiện đáng kể so với quay lui và giải được các bài toán có kích thước trung bình ($N \leq 50$), tuy nhiên cũng không khả thi khi ($N \geq 100$) do không gian nhánh về lý thuyết tăng theo cấp số mũ.
- IP và CP cho kết quả tối ưu đến $N = 200$ tuy dựa vào kết quả thu được ta có thể thấy CP tỏ ra kém hiệu quả hơn hẳn IP về mặt thời

gian dù cùng đạt chất lượng lời giải tối ưu, khi thời gian giải của CP là rất lớn đối với bất kể quy mô của bài toán dẫn đến vượt ngưỡng thời gian giải của hệ thống (TLE) .

- Ưu điểm:
    - Bảo đảm nghiệm tối ưu:Các giải thuật đảm bảo trả về nghiệm tối ưu cho bài toán
    - Phù hợp với các bài toán bộ dữ liệu đầu vào nhỏ và yêu cầu kiểm chứng mô hình
- Nhược điểm:
    - Không mở rộng được với các bài toán lớn: Khi số lượng hành khách n tăng, không gian nghiệm tăng theo cấp số giai thừa do phải xác định thứ tự ghép cặp đón-trả, khiến thời gian giải tăng đột biến.
    - Thời gian và bộ nhớ tăng rất nhanh theo quy mô bài toán.
    - Tốn tài nguyên (với ip và cp): Các mô hình yêu cầu nhiều bộ nhớ (RAM) và tài nguyên xử lý, không phù hợp với thiết bị giới hạn hoặc môi trường thời gian thực

3.2.2 Thuật toán gần chính xác

Các giải thuật gần chính xác như tham lam (greedy), leo đồi (Hill Climbing) và tối ủ (Simulated Annealing) không đảm bảo trả về lời giải tối ưu toàn cục tuy nhiên có thể trả về lời giải tốt gần tối ưu trong thời gian rất ngắn, đặc biệt phù hợp cho các bài toán có quy mô lớn ($N \geq 500$):

- Thuật toán tham lam vốn thường được xem là một thuật toán đơn giản và dễ bị kẹt tại cực trị cục bộ, lại đạt hiệu suất tốt nhất toàn bảng cả về tốc độ và giá trị hàm mục tiêu. Thuật toán nổi bật về tốc độ khi giải được bài toán với $N = 1000$ chỉ trong 178 ms và đang cho kết quả tốt nhất trong nhóm thuật toán
- Các thuật toán leo đồi và tôi ủ đều cho kết quả gần với lời giải tối ưu, tuy nhiên trong phần lớn các trường hợp, giá trị hàm mục tiêu đạt được vẫn thấp hơn so với thuật toán tham lam. Tuy nhiên phương án khởi tạo được sử dụng cho hai thuật toán này là một phiên bản tham lam đơn giản, chỉ sắp xếp đơn hàng theo tiêu chí giá trị giảm dần mà không tối ưu hoá toàn cục như thuật toán tham lam được thiết kế riêng. Trong bối cảnh đó, cả leo đồi và tôi ủ đều thể hiện khả năng cải thiện đáng kể lời giải ban đầu từ khởi tạo (có thể tham khảo bảng so sánh kết quả ở dưới). Trong đó:
    - Thuật toán Hill Climbing có khả năng tiến dần đến lời giải tốt hơn thông qua cải tiến cục bộ, và thường đạt kết quả gần với Greedy, đặc biệt trong các bài toán quy mô lớn. Tuy nhiên, do bản chất tìm kiếm cục bộ, Hill Climbing dễ bị mắc kẹt tại các cực trị cục bộ, dẫn đến hiệu suất không ổn định giữa các lần chạy. Trong thực nghiệm, thời gian thực thi trung bình của

thuật toán này vào khoảng 900ms tại N = 1000, một mức chấp nhận được cho các ứng dụng thực tế.

- Trong khi đó, Simulated Annealing (SA) cho thấy tính ổn định vượt trội nhờ cơ chế chấp nhận lời giải tạm thời kém hơn với xác suất có kiểm soát. Thuật toán này không bị kẹt ở cực trị cục bộ như Hill Climbing, và duy trì thời gian thực thi ổn định quanh 850ms trong toàn bộ các bài toán thử nghiệm. Hàm mục tiêu của SA cải thiện hơn khá nhiều so với phương án Greedy khởi tạo trong hầu hết các trường hợp, cùng với việc thuật toán này được đánh giá cao về khả năng khái quát và độ tin cậy giữa các lần chạy, đặc biệt hữu ích trong các không gian tìm kiếm có nhiều cực trị.

TC		Greedy khởi tạo		Hill Climbing		Simulated Annealing	
		best	time	best	time	best	time
5 2	27	27	45	27	54	27	844
10 2	57	57	43	57	120	57	843
50 3	257	257	45	257	128	257	845
100 5	1065	1065	44	1065	71	1065	844
200 10	2037	2051	46	2089	285	2086	945
300 10	2971	2995	49	2995	305	2995	847
500 50	5217	5212	56	5221	591	5222	849
800 80	8536	8521	71	8549	801	8544	849
900 90	9410	9433	74	9502	717	9501	850
1000 100	10368	10350	78	10377	901	10376	849

Bảng 2.Bảng so sánh kết quả của thuật toán leo đồi và tôi ủ với phương án greedy được khởi tạo

- Ưu điểm:
  - Mở rộng tốt, giải được bào toán có quy mô lớn
  - Tốc độ nhanh, chi phí tính toán thấp
  - Có thể két hợp linh hoạt với các chiến lược sinh láng giềng hoặc khởi tạo.
- Nhược điểm:
  - Không đảm bảo tối ưu toàn cục
  - Kết quả phụ thuộc vào tham số và chiến lược tìm kiếm

3.2.3 Kết luận.

Qua quá trình thực nghiệm và phân tích hiệu năng của các thuật toán trên bài toán, ta có thể rút ra một số kết luận quan trọng liên quan đến khả năng áp dụng thực tế, chất lượng lời giải và khả năng mở rộng của từng phương pháp. Ta có bảng so sánh kết hiệu quả sử dụng của từng thuật toán như sau:

	Backtrack	B&B	ILP	CP	Greedy	Hill Climbing	Simulated Annealing
Kết quả thu được	Tối ưu toàn cục	Tối ưu toàn cục	Tối ưu toàn cục	Tối ưu toàn cục	Tối ưu cục bộ	Tối ưu cục bộ	Tối ưu cục bộ
Tính mở rộng	Không khả thi với N >10	Không khả thi với N > 50	Chỉ hiệu quả với N ≤ 200	Chỉ hiệu quả với N ≤ 200	Có khả năng mở rộng với cả các N lớn	Có khả năng mở rộng với cả các N lớn	Có khả năng mở rộng với cả các N lớn
Tốc độ	Rất chậm tăng theo hàm số mũ	Tương tự backtrack tuy nhiên cải thiện đáng kể với các N nhỏ	Trung bình	Chậm	Rất nhanh	Nhanh	Trung bình
Chất lượng lời giải	Tốt với các N nhỏ	Tốt với N từ nhỏ đến vừa	Tốt	Tốt	Trả về lời giải có chất lượng tốt với chiến lược phù hợp	Trả về lời giải có chất lượng tốt với chiến lược phù hợp	Trả về lời giải có chất lượng tương đối tốt với chiến lược phù hợp
Nhận xét	Chỉ phù hợp với các bài toán nhỏ	Chỉ phù hợp với các bài toán vừa và nhỏ cần có cận tốt	Cần mô hình hóa tuyến tính chỉ phù hợp với bài toán vừa và nhỏ	Mạnh với ràng buộc logic. Tuy nhiên chỉ phù hợp với bài toán vừa và nhỏ thời gian chạy rất lâu dễ dẫn đến TLE	Tốt nhất trong các thuật toán được lựa chọn phù hợp với các bài toán lớn. Trả về lời giải có chất lượng cao với chiến lược gán phù hợp	Phù hợp với các bài toán lớn. Dễ cài đặt và cho lời giải có chất lượng tương đối tốt. Tuy nhiên dễ bị kẹt ở tối ưu cục bộ	Phù hợp với các bài toán lớn. Dễ cài đặt và cho lời giải có chất lượng tương đối tốt. Có khả năng thoát khỏi tối ưu cục bộ, tuy nhiên không đảm bảo.

Bảng 3. Bảng so sánh hiệu quả các thuật toán được sử dụng

Từ đó ta có nhận xét cụ thể cho các nhóm thuật toán được sử dụng như sau:

- Các thuật toán chính xác như Backtracking, Branch and Bound, Integer Linear Programming (ILP) và Constraint Programming (CP) cho thấy hiệu quả rõ rệt ở các bài toán có quy mô nhỏ đến trung bình. Trong đó, ILP và CP có thể giải tối ưu bài toán đến N = 200 trong thời gian hợp lý, và do đó thích hợp với các ứng dụng yêu cầu lời giải tối ưu tuyệt đối hoặc phục vụ cho việc đánh giá độ chính xác của các thuật toán xấp xỉ. Tuy nhiên, những thuật toán này gặp hạn chế lớn về thời gian và bộ nhớ khi kích thước bài toán tăng, khiến chúng không phù hợp cho các trường hợp quy mô lớn hoặc yêu cầu phản hồi thời gian thực.
- Ở chiều ngược lại, các thuật toán gần chính xác như Greedy, Hill Climbing và Simulated Annealing tỏ ra đặc biệt hiệu quả với các bài toán có quy mô lớn (N ≥ 500). Trong số này, Greedy cho thấy hiệu suất vượt trội cả về tốc độ và chất lượng lời giải, nhờ khai thác tốt cấu trúc cụ thể của bài toán. Simulated Annealing, dù có giá trị hàm mục tiêu thấp hơn đôi chút trong thực nghiệm, lại thể hiện tính ổn định và khả năng khái quát tốt hơn, đặc biệt phù hợp với các bài toán có không gian tìm kiếm phức tạp và nhiều cực trị cục bộ. Hill

Climbing đóng vai trò trung gian, với hiệu năng tốt nhưng độ ổn định phụ thuộc vào điểm khởi tạo.

Tóm lại, đối với bài toán được xét, Greedy là lựa chọn hiệu quả nhất trong các bài toán quy mô lớn, nhờ sự kết hợp giữa tốc độ và chất lượng lời giải. Hill Climbing và Simulated Annealing là một phương án thay thế đáng tin cậy khi cần giải pháp có tính ổn định cao. Các thuật toán chính xác nên được ưu tiên sử dụng trong các bài toán quy mô nhỏ hoặc đóng vai trò làm chuẩn tham chiếu để đánh giá chất lượng lời giải xấp xỉ trong các thiết lập thực nghiệm.