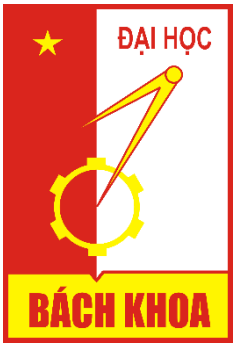
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

----- □ □ -----



ĐỒ ÁN MÔN HỌC

Giảng viên: TS. Bùi Quốc Trung

Nhóm sinh viên thực hiện:

Trịnh Quang Hùng	20190054
Lê Anh Đức	20190046
Nguyễn Văn Thành	20194678
Nguyễn Hải Dương	20190044

Hà Nội, ngày 1 tháng 2 năm 2023

MỤC LỤC

Chương 1. Giới thiệu đề tài

1.1. Bài toán

- Có N môn 1, 2, …, N cần được xếp lịch thì học kỳ.
- Môn i có số lượng sinh viên đăng ký thi là d(i)
- Giữa N môn thi có danh sách các cặp 2 môn (i,j) không thể xếp trùng kíp, ngày do có cùng sinh viên đăng ký thi
- Có M phòng thi 1, 2, …, M, trong đó phòng j có số lượng chỗ ngồi là c(j)
- Mỗi ngày được chia thành 4 kíp
- Hãy lập kế hoạch bố trí lịch và phòng cho các môn thi sao cho tổng số ngày diễn ra N môn thi là nhỏ nhất

1.2. Dữ liệu đầu vào

- Input
  - Dòng 1: ghi N
  - Dòng 2: $d_1, d_2, \ldots, d_N$
  - Dòng 3: ghi M
  - Dòng 4: $c_1, c_2, \ldots, c_M$
  - Dòng 5: ghi số nguyên dương K
  - Dòng 5 + k (k = 1,…, K): ghi i và j (là 2 môn thi trùng sinh viên đăng ký → không thể xếp trùng ngày, trùng kíp)

1.3. Bộ dữ liệu

Bộ dataset: Mỗi bộ dataset sẽ được định hình bởi các chỉ số (N, M, C). Trong đó:

- N là số lượng môn học
- M là số lượng phòng
- C là số cặp môn học mâu thuẫn với nhau

Có tổng cộng 100 bộ dữ liệu, trong đó:

- Có 20 tập kích thước khác nhau.
- Mỗi tập kích thước có 5 bộ dữ liệu.

Chương 2. Các giải thuật đề xuất

2.1. MIP

Mô hình hóa bài toán

Sẽ có các biến sau đây trong bài toán Mixed Integer Programming

- $x[i][j][k]$ là biến nhị phân xếp môn i vào phòng thứ j tại kíp thứ k trong đó $i, k \in 0, \ldots, N-1$ và $j \in 0, \ldots, M-1$
- y là số kíp thi $D(y) = 0, 1, \ldots, N-1$.
- Gọi C là tập các môn học (i, j) mà không thể xếp cùng kíp.

Mục tiêu của bài toán là minimize y.

Chúng ta có các rằng buộc gốc cho bài toán như sau

1. Mỗi môn đều được xếp lịch thi duy nhất 1 lần vào 1 phòng

$$\sum_{k=0}^{N-1} \sum_{j=0}^{M-1} x[i][j][k] = 1 \quad \forall\, i = 0 \ldots N-1$$

2. Mỗi phòng chỉ có thể được xếp tối đa 1 môn trong 1 kíp

$$\sum_{i=0}^{N-1} x[i][j][k] \leq 1 \quad \forall\, k = 0 \ldots N-1;\ j = 0 \ldots M-1$$

3. Xếp các môn thi i vào phòng thi j có sức chưa $c[j]$ phù hợp:

$$\sum_{k=0}^{N-1} x[i][j][k] * d[i] \leq c[j] \quad \forall\, i = 0 \ldots N-1;\ j = 0 \ldots M-1$$

4. Hai môn thi conflict với nhau không thể xếp cùng 1 kíp

$$\sum_{j=0}^{M-1} (x[i1][j][k] + x[i2][j][k]) \leq 1 \quad \forall\, k = 0 \ldots N-1;\ (i1, i2) \in C$$

Ngoài ra chúng ta có rằng buộc cho biến mục tiêu

$$x[i][j][k] * k \leq y \forall j = 0 \ldots M-1; i, k = 0 \ldots N-1$$

2.2. CP

Mô hình hóa bài toán

Sẽ có các biến sau đây trong bài toán Constraints Programming

Chúng ta sẽ có các biến sau đây

- $x[i]$ thể hiện kíp thi cho môn i trong đó $i \in 0, \ldots, N-1,\ D(x[i]) = 0, \ldots, N-1$
- $y[i][j]$ thể hiện môn i được xếp vào phòng j trong đó $i \in 0, \ldots, N-1, j \in 0, \ldots, M-1, D(y[i][j]) = 0, 1$
- Gọi C là tập các môn học (i, j) mà không thể xếp cùng kíp.

Mục tiêu của bài toán là minimize $max(x) \rightarrow min$.

Chúng ta có các rằng buộc gốc cho bài toán như sau:

1. Hai môn cùng kíp thi không được xếp cùng phòng

$$x[i1] = x[i2] \Rightarrow y[i1][j] + y[i2][j] \leq 1; \forall i1, i2 \in 1, \ldots, N, j \in 1, \ldots, M$$

2. Mỗi phòng chỉ có thể được xếp tối đa 1 môn trong 1 kíp

$$\sum_{j=0}^{M-1} y[i][j] = 1; \forall i \in 1, \ldots, N$$

3. Xếp các môn thi i vào phòng thi j có sức chưa $c[j]$ phù hợp:

$$\sum_{k=0}^{M-1} y[i][j] * c[i] \geq d[j]; \quad \forall\, i = 0 \ldots M-1$$

4. Hai môn thi conflict với nhau không thể xếp cùng 1 kíp

$$\forall (i, j) \in C \Rightarrow x[i] \neq x[j]$$

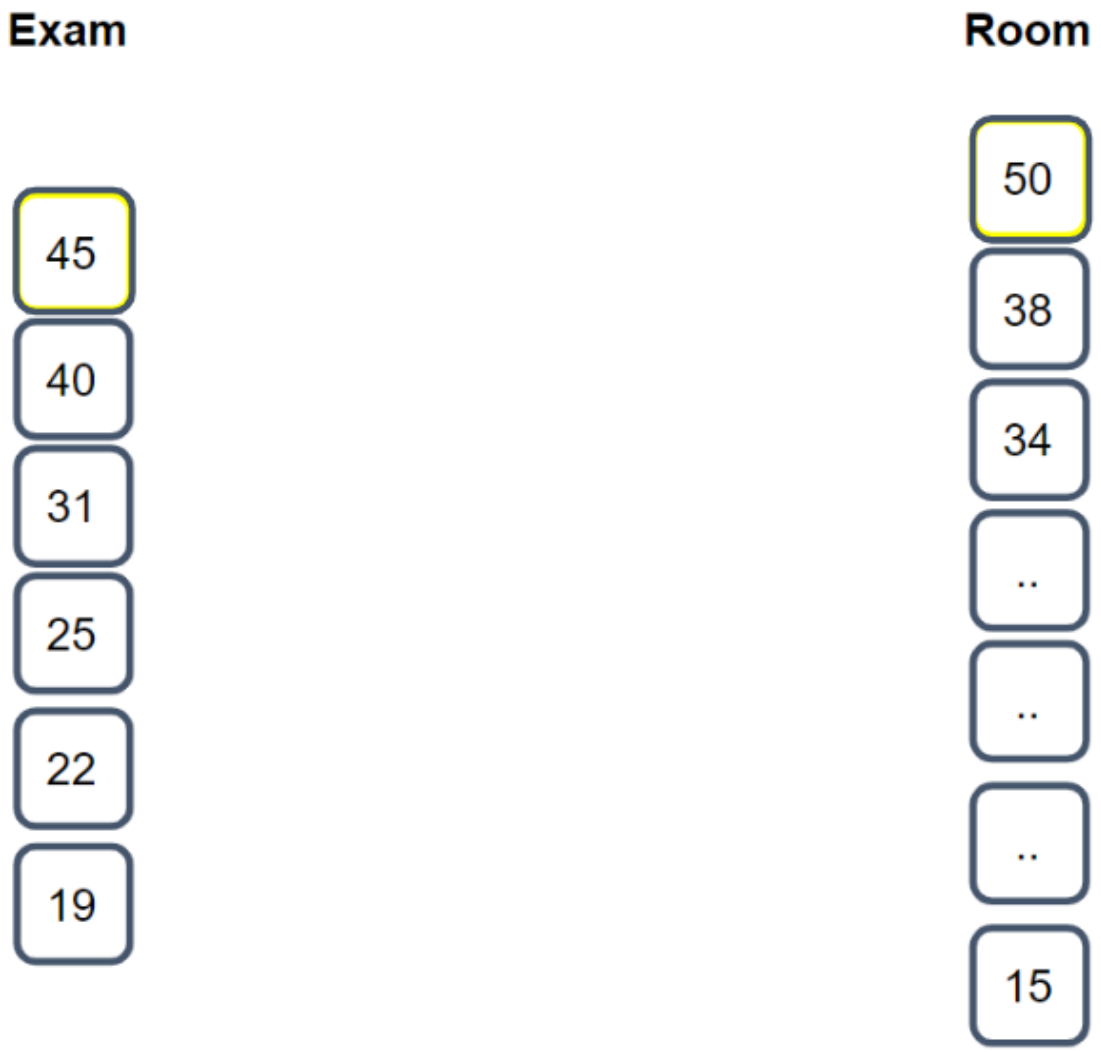
Ngoài ra chúng ta có rằng buộc cho biến mục tiêu

$$x[i][j][k] * k <= y; \forall j = 0 \ldots M-1; i, k = 0 \ldots N-1$$

2.3. Greedy

Hướng tiếp cận của thuật toán là xây dựng từng kíp thi một cách lần lượt theo phương pháp tham lam.

Trước tiên, thuật toán sắp xếp lại các môn thi theo thứ tự giảm dần của số thí sinh tham dự, tương tự, sắp xếp các phòng theo giảm dần theo số lượng chỗ ngồi.



Hình 1: Hình minh họa sau khi sắp xếp

Thuật toán sẽ xây dựng các kíp một cách lần lượt cho đến khi tất cả các môn học đã được xếp kíp. Mã giả của thuật toán như sau:

Tại mỗi vòng lặp, thuật toán duyệt lần lượt qua những môn học chưa được xếp kíp. Nếu một môn học chưa được xếp kíp và thỏa mãn 2 điều kiện sau, môn học đó sẽ được thêm vào kíp hiện tại:

1. Môn học đó không mâu thuẫn (có chung học sinh tham gia) với môn học nào đã được thêm vào kíp hiện tại.
2. Phòng lớn nhất chưa được sử dụng có đủ chỗ cho môn học đó.

Nếu thỏa mãn cả 2 điều kiện trên, môn học sẽ được thêm vào kíp hiện tại, phòng học tương ứng sẽ bị đánh dấu là đã được sử dụng. Trong trường hợp ngược lại, môn học sẽ được bỏ qua và duyệt tiếp tới môn học tiếp theo.

Khi tất cả môn học đã được xếp chỗ, số kíp cần sử dụng sẽ là kết quả của thuật toán.

2.4. Simulated Annealing

Trước hết, hướng tiếp cận dựa trên thuật toán SA (Simulated Annealing) là chuyển từ bài toán tối ưu (Optimization) sang bài toán tìm lời giải thỏa mãn ràng buộc (Find feasible solution). Sau đây là mã giả của việc chuyển đổi:

Pseudo-code: Convert "Optimization" to "Find Feasible"

```
1. begin:
2. k = init_start_solution()
3. while(1):
4.    if find_feasible_solution(k) == True:
5.       k -= 1
6.    else: break
7. return k
8. end
```

Hình 3: Convert "Optimization" to "Find Feasible"

Trong thuật toán Find feasible solution, số kíp **k** sẽ được cho trước, mục tiêu của thuật toán này là kiểm tra xem có xếp được lịch thỏa mãn tất cả các ràng buộc với **k** kíp hay không.

Lời giải sẽ được khởi tạo bằng thuật toán Greedy ở trên, thuật toán Greedy sẽ khởi tạo đến kíp thứ **k** và dừng lại, những môn học còn lại sẽ được nhóm vào

tập chưa được sắp xếp. Thuật toán sau đó dựa trên việc tìm kiếm các lân cận xung quanh để tìm ra các lời giải hợp lệ. Mã giả của thuật toán như sau:

Pseudo-code: Find-feasible_solution

1. Input: Number of section 'k'
2. Output: Can find a feasible solution with k sections?

```
3. X = Initial_Greedy_Solution()
4. repeat:
5.   X'= keep a copy of X
6.   X = SA_based_local_search(X)
7.   X = swap_two_sections(X)
8.   if f(X') < f(X):
9.      X = X'
10.until the stop criterion is met
```

Hình 4: Find feasible solution

2 thành phần chính của thuật toán là

- **SA_based_local_search:** với mục tiêu tìm kiếm dựa trên các lời giải lân cận và xem xét chúng dựa trên ý tưởng của thuật toán SA (Simulated Annealing).
- **swap_two_section:** làm xáo trộn các môn học giữa 2 kíp với mục tiêu giúp thuật toán vượt qua được lời giải tối ưu cục bộ.

2.4.1. SA based local search.

Mã giả của thành phần này như sau:

Pseudo-code: SA_based_local_search

1. Input: a solution X(SOL, UNASSIGN)
2. Output: a new solution

```
3. Initial temperature T
5. for section in SOL:
4.   Keep a copy section_COPY <- section, UNASSIGN_COPY <- UNASSIGN
6.   section', UNASSIGN' = neighbor(section, UNASSIGN)
7.   Δ ← len(UNASSIGN_COPY) - len(UNASSIGN')
8.   generate a uniform random number 'r' on [0, 1]
9.   if (Δ > 0) or (Δ < 0 and r < exp((Δ-1)/T)):
10.      section = section', UNASSIGN = UNASSIGN'
12.  else:
13.      section = section_COPY, UNASSIGN = UNASSIGN_COPY
14.T ← 1/((1/T) + 0.15)
```

Hình 5: SA based local search

Trong đó, cấu trúc hàng xóm lân cận (neighbor) của thuật toán này được xây dựng như sau:

Pseudo-code: Neighbor structure:
1. Input: Section: X, Unassign: U
2. Output: New Section: X', New Unassign: U'

```
3.  Random an element 'e' in section X
4.  Remove 'e' from X
5.  Conflict = All subject conflict with X:
6.  for 's' in Unassign:
7.     if 's' not in Conflict:
8.        insert 's' to X
9.  Accept, Reject = Greedy (X)
10. X = Accept, Unassign += Reject
11. Return X, Unassign
```

Hình 6: Cấu trúc hàng xóm lân cận

2.4.2. Swap_two_section

Mục tiêu của thành phần này là làm xáo trộn lời giải hiện tại với mong muốn vượt qua những điểm tối ưu cục bộ. Mã giả của thành phần này như sau:

Pseudo-code Swap two section

1. Input: Section: X, Section: Y, UNASSIGN
2. Output: New Section: X', New Section: Y', UNASSIGN'

```
keep a copy of Copy_X, Copy_Y, Copy_UNASSIGN
Random an element 'e' in section X
Remove 'e' from X
Con(e, Y) <- set of events in Y that have conflict with e
X = X ∪ Con(e, Y) \ {e}
Y = Y ∪ {e} \ Con(e, Y)
if !conflict(X):
  X_Accept, X_Reject = Greedy(X)
  Y_Accept, Y_Reject = Greedy(Y)
  if len(X_Reject) > 0 or len(X_Reject) > 0:
    return
  else:
    for sub in UNASSIGN:
      if Con(sub, X) = 0:
        move 'sub' from UNASSIGN to X
      else if Con(sub, Y) = 0:
        move 'sub' from UNASSIGN to Y
  X_Accept, X_Reject = Greedy(X)
  Y_Accept, Y_Reject = Greedy(Y)
  X = X_Accept, Y = Y_Accept, UNASSIGN += X_Reject + Y_Reject
  if len(UNASSIGN) < len(Copy_UNASSIGN):
      return X, Y, UNASSIGN
```

Hình 7: Swap two section

Chương 3. Thực nghiệm so sánh

3.1. Môi trường thử nghiệm

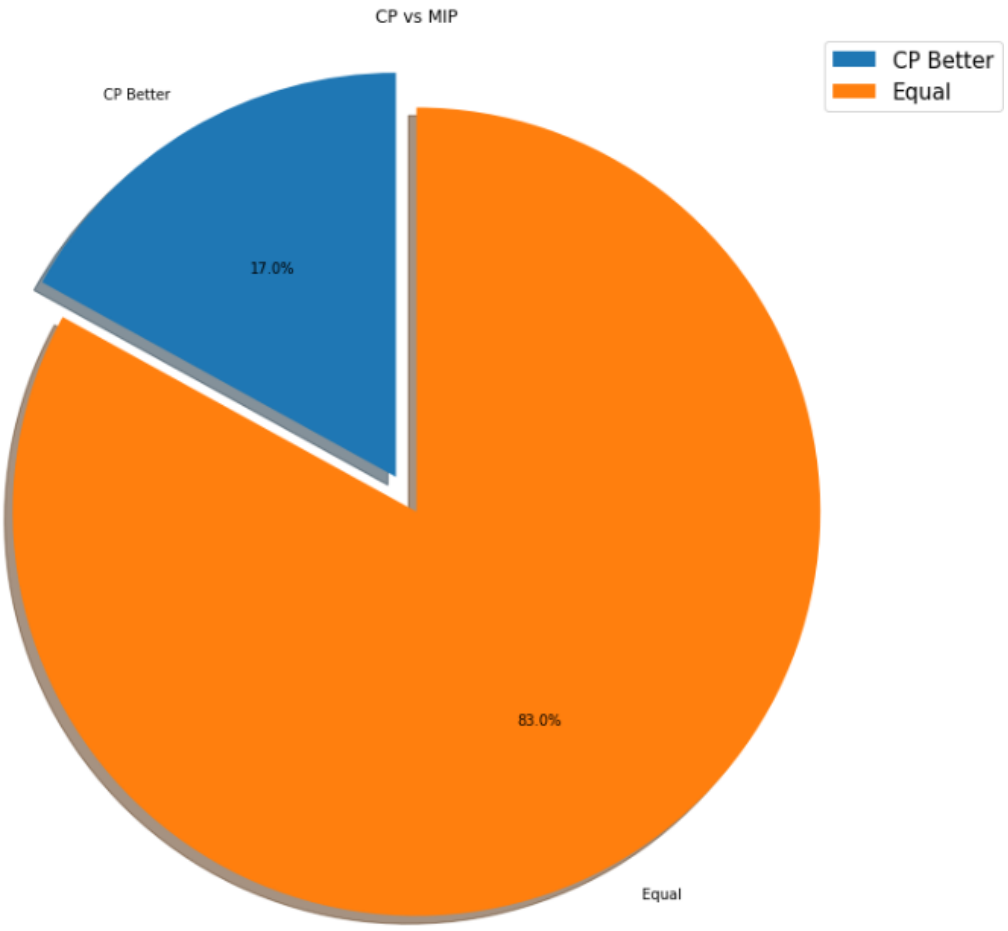
- Ngôn ngữ cài đặt: Python
- Môi trường chạy: Core i5, 16GB RAM
- Các siêu tham số:
    - MIP và CP:
        - Giới hạn thời gian chạy 2 phút
        - Chạy quá 2 phút sẽ trả về kết quả hiện tại
    - Simulated Annealing
        - Số vòng lặp: 150
        - Khởi tạo Temperature: 10

3.2. Nhận xét tổng quan.

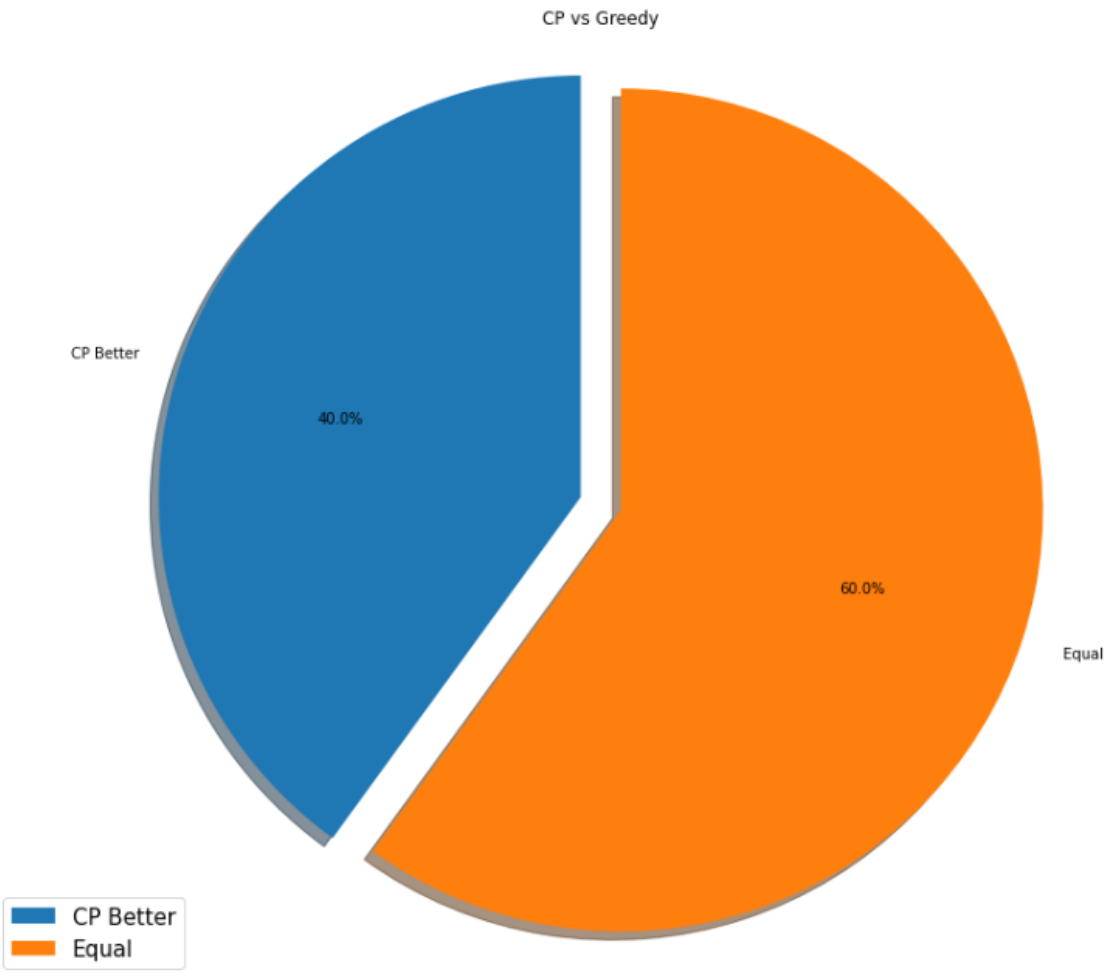
- Trên toàn bộ (100) tập dữ liệu, CP luôn là thuật toán đưa ra lời giải tốt nhất. Coi đây là lời giải tối ưu.
- Các thuật toán đưa ra lời giải càng tốt, thời gian chạy càng lâu
- Do có nhiều ràng buộc, thuật toán SA chỉ vượt qua được Greedy trong một số ít lần
- Thuật toán Greedy tuy chưa đưa ra lời giải tối ưu nhưng chấp nhận được, thời gian thực hiện ngắn, có tính ứng dụng cao

3.3. So sánh chi tiết.

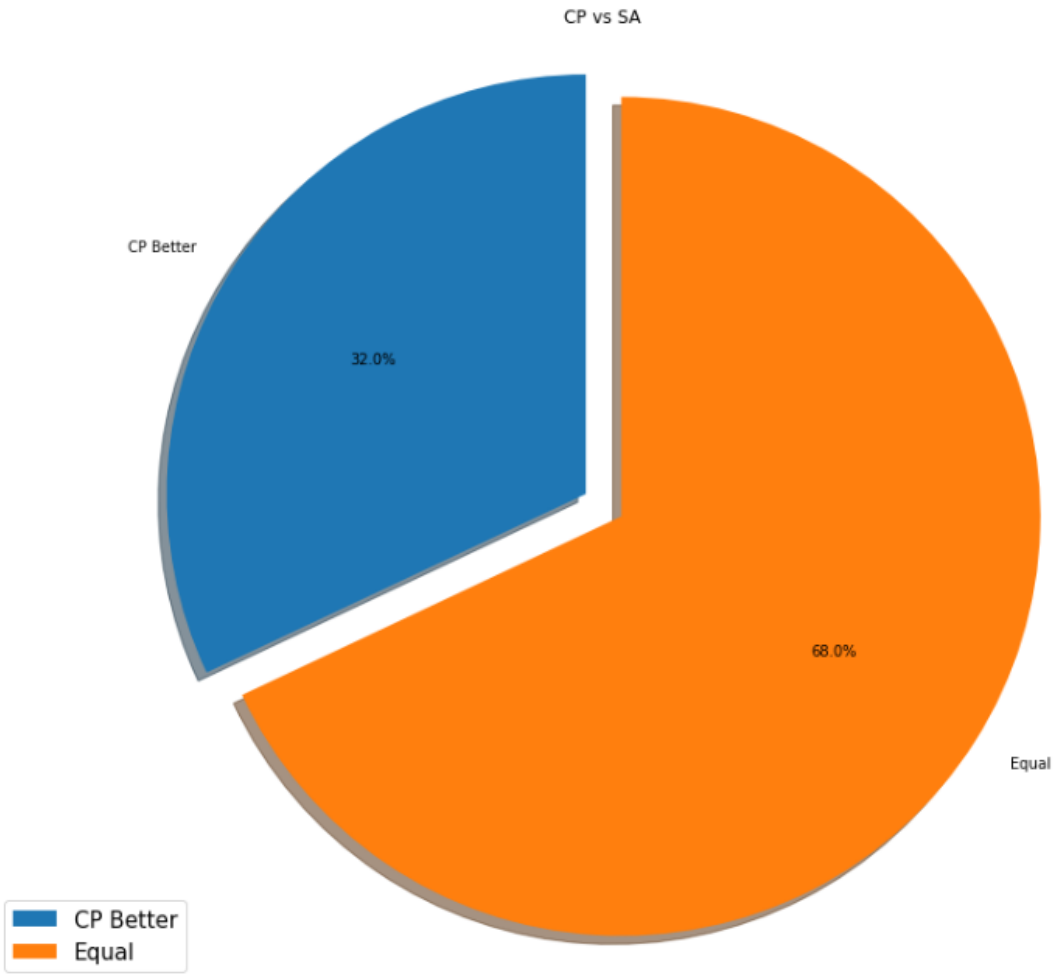
3.3.1. So sánh kết quả các thuật toán với CP



Hình 8: So sánh kết quả CP với MIP



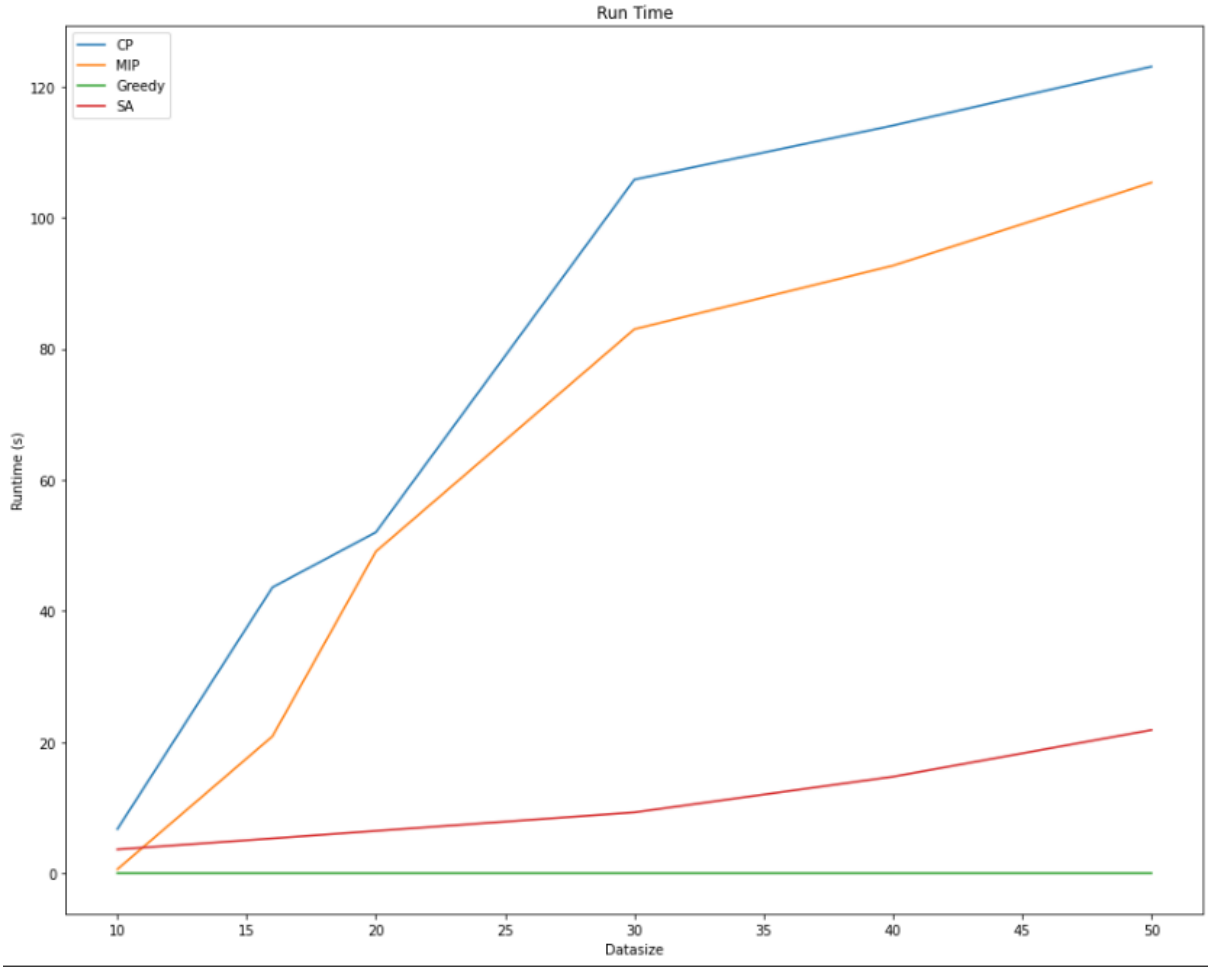
Hình 9: So sánh kết quả CP với Greedy



Hình 10: So sánh kết quả CP với Simulated Annealing

3.3.2. So sánh thời gian chạy của các thuật toán.

Dưới đây là kết quả so sánh thời gian chạy giữa các thuật toán theo kích thước bộ dữ liệu.



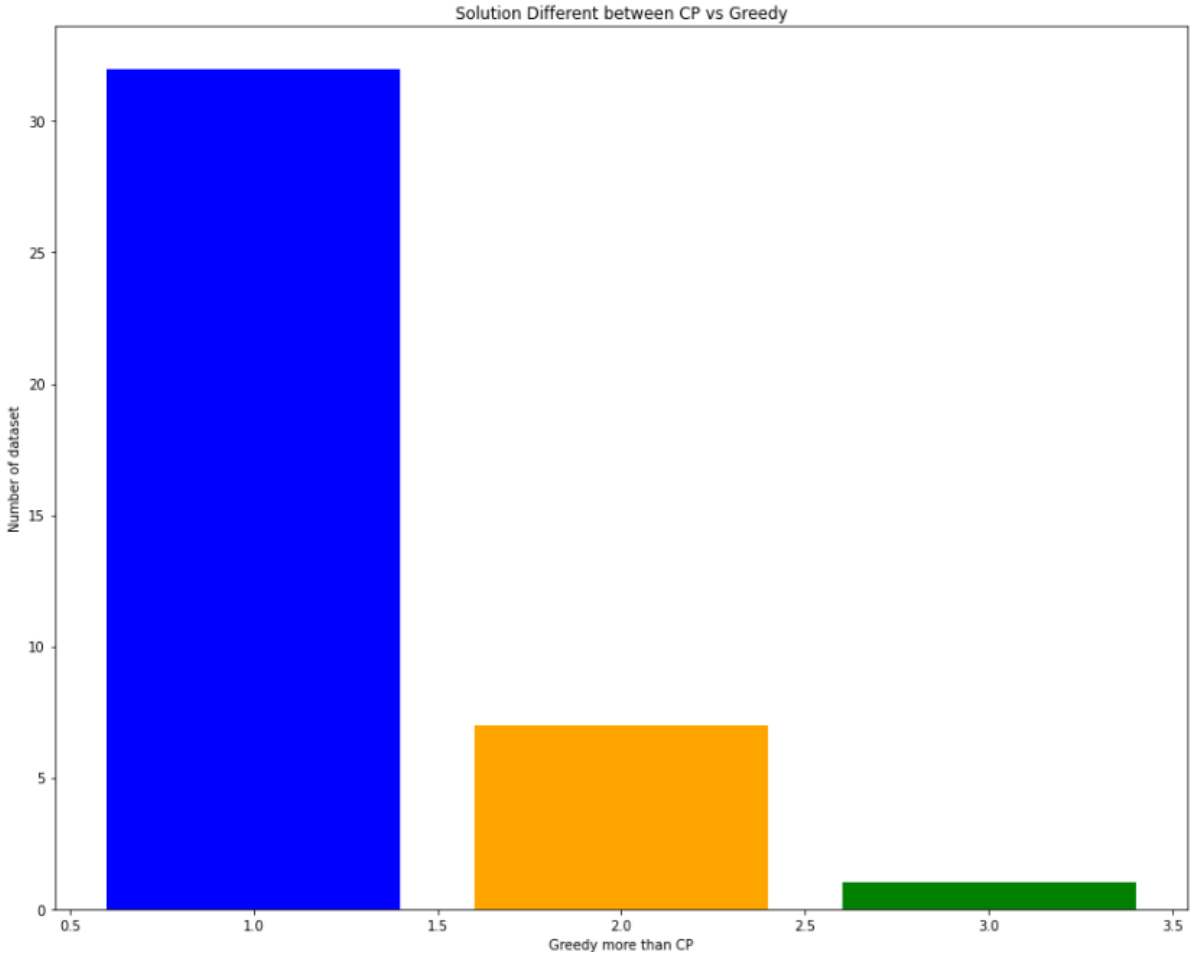
Hình 11: Thời gian chạy của các thuật toán

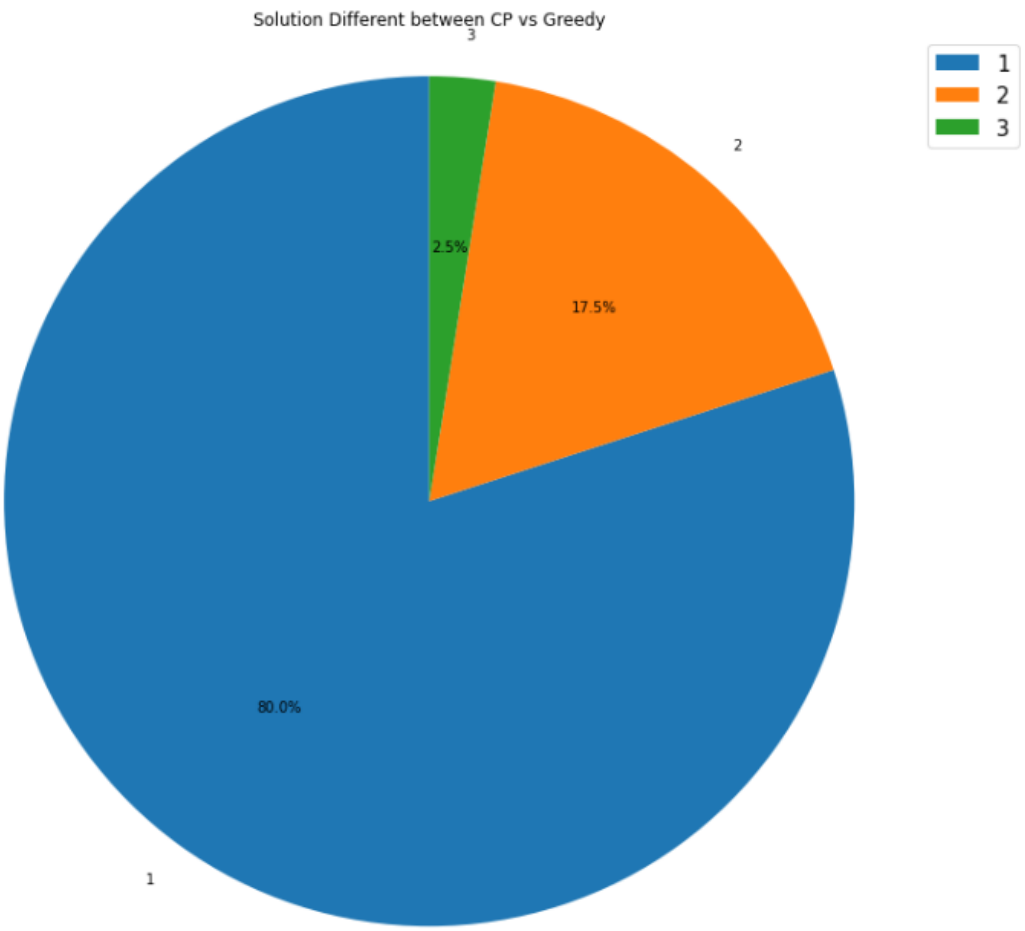
3.3.3. Đánh giá hiệu quả của thuật toán Greedy.

Thuật toán Greedy có thời gian chạy rất nhanh (với 4.5s cho bộ dữ liệu lớn nhất), tuy nhiên kết quả cho ra lại có thể chấp nhận được. Dưới đây là biểu đồ so sánh kết quả của thuật toán Greedy với thuật toán tối ưu. Theo hình 9, có tới 60% bộ dữ liệu trong đó Greedy đưa ra lời giải tối ưu. Đối với 40% còn lại (tương ứng 40 bộ dữ liệu), ta có quan sát theo các biểu đồ sau đây:

- Có tới 32 bộ dữ liệu (80%) mà kết quả Greedy chỉ đưa ra nhiều hơn CP 1 kíp.
- Có 7 bộ dữ liệu (17.5%) kết quả Greedy đưa ra nhiều hơn CP 2 kíp.
- Chỉ có duy nhất 1 bộ dữ liệu (2.5%) kết quả Greedy nhiều hơn CP 3 kíp.

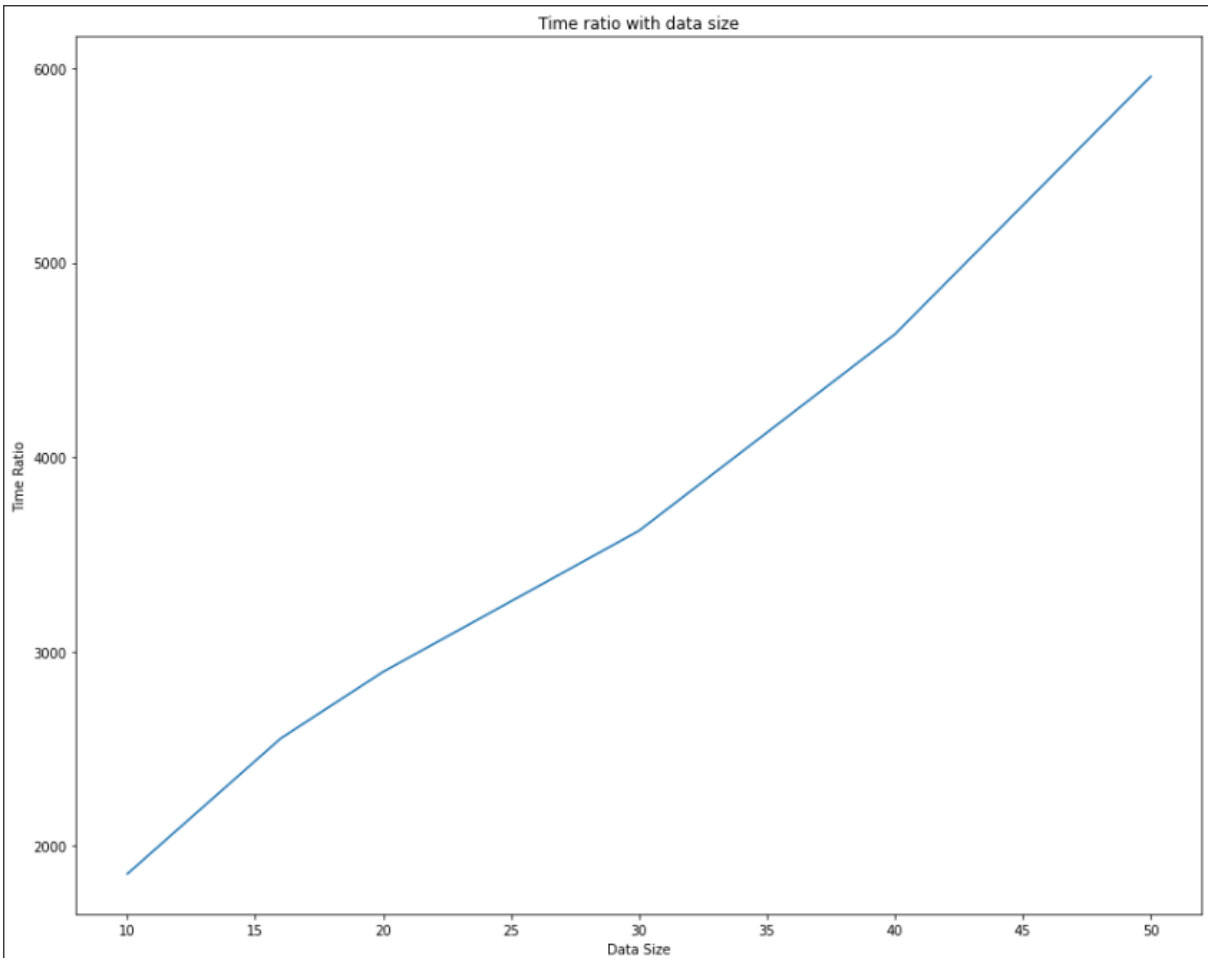
Thuật toán Greedy có thời gian chạy nhanh, kết quả so với kết quả tối ưu là chấp nhận được do đó có tính ứng dụng cao.



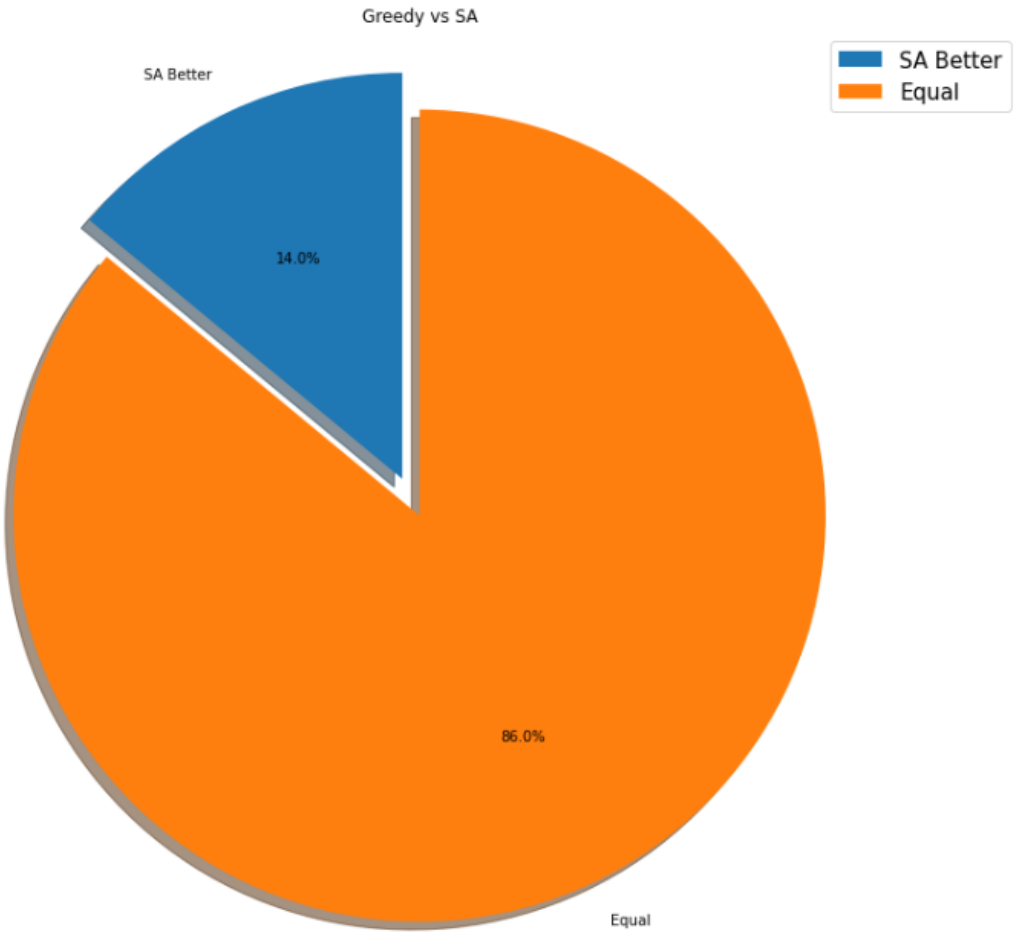


Hình 12: So sánh lời giải của thuật toán Greedy với lời giải tối ưu.

3.3.4. So sánh thuật toán SA với Greedy.



Hình 13: Tỷ lệ thời gian chạy của thuật toán SA so với thuật toán Greedy



Hình 14: So sánh kết quả của thuật toán SA so với Greedy.

Do bài toán có nhiều ràng buộc và hướng tiếp cận của nhóm chỉ duyệt trên những lời giải hợp lệ, cộng với setting các tham số cho thuật toán, do đó thời gian thực thi thuật toán SA khá lâu, trung bình lâu hơn thuật toán Greedy 100 lần, tuy nhiên lại chỉ có 14 bộ dữ liệu (tương ứng 14%) mà ở đó SA cho kết quả tốt hơn Greedy.