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

———————— * ————————

BÁO CÁO BÀI TẬP LỚN

TỐI ƯU LẬP KẾ HOẠCH

Đề tài 13: Exam Timetable

Đỗ Minh Đức	20200158
Vũ Anh Đức	20200181
Lê Đức Minh	20200395
Nguyễn Phương Nam	20200420

Giáo viên hướng dẫn:

TS. Bùi Quốc Trung

HÀ NỘI
Ngày 9 tháng 1 năm 2024

Mục lục

Chương 1

Giới thiệu đề tài

1.1 Bài toán

- Có N môn học $1, 2, ..., N$ cần được xếp lịch thi cuối kỳ. Mỗi môn cần được xếp thời gian và phòng thi tương ứng.
- Có M phòng trong đó phòng i có sức chứa là $c(i)$ người.
- Mỗi ngày có 4 kíp thi là 1, 2, 3, 4.
- Mỗi môn i có số sinh viên đăng ký là $d(i)(i = 1, ..., N)$.
- Giữa N môn, có K cặp môn (i, j) trong đó môn i và môn j không thể trùng kíp, ngày do có cùng sinh viên đăng ký dự thi.
- **Mục tiêu:** 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 và đầu ra

Input:

- Dòng 1: N
- Dòng 2: $d_1, d_2, ..., d_N$
- Dòng 3: M
- Dòng 4: $c_1, c_2, ..., c_M$
- Dòng 5: K

- Dòng 5 $+k(k = 1, .., K)$: ghi 2 số nguyên i, j (2 môn có chung thí sinh đăng ký, tức không thể tổ chức chung kíp)

Output: Mỗi dòng $i(i = 1, 2, ..., N)$ ghi 3 số nguyên $i, s[i]$ và $r[i]$ trong đó $s[i]$ là kíp thi và $r[i]$ là phòng thi của môn i.

1.3 Bộ dữ liệu

Bộ dữ liệu được sinh ra gồm 9 tập có kích thước khác nhau, mỗi tập có 3 test case tương ứng. Kích thước của mõi tập được định hình bới các chỉ số N, M, K. Trong đó:

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

Ở đây nhóm sinh ra bộ dữ liệu với 9 kích thước khác nhau về (N, M) là: $(10, 2), (16, 3), (20, 4), (30, 6), (40, 8), (50, 10), (60, 12), (70, 16), (80, 20)$. Với mỗi giá trị cho trước, nhóm sẽ sinh ra 1 cách xếp các môn học vào các phòng để tạo ra một lời giải chấp nhận được. Sau đó sử dụng lời giải chấp được này để sinh ra các ràng buộc.

Chương 2

Backtracking with Branch and Bound

2.1 Ý tưởng thuật toán

Ta nhận thấy nếu tồn tại lời giải cho bài toán thì số kíp thi tối đa là N với N là số môn học (trong trường hợp tệ nhất thì mỗi môn sẽ được tổ chức riêng 1 kíp từ ngày này qua ngày khác). Do đó để duyệt qua toàn bộ lời giải khả thi, mỗi môn học sẽ được gán với một phòng thi và một kíp thi trong tập $\{1, 2, ..N\}$. Giá trị của hàm mục tiêu, tức số ngày để tổ chức kì thi sẽ được tính dựa kíp thi được tổ chức muộn nhất.

2.2 Giải thuật chi tiết

Giải thuật chi tiết xem ở Algorithm 1 dưới đây.

2.3 Nhận xét

Do đây là giải thuật chính xác nên thời gian chạy khá lớn với những đầu vào có kích thước lớn.

Ngoài ra ban đầu thay vì chọn kíp thi từ tập $\{1, 2, ..N\}$ như thuật toán hiện tại, nhóm có thử thực hiện gán trực tiếp cho mỗi môn học là kíp thi trong tập $\{1, 2, 3, 4\}$. Sau đó số ngày thi sẽ được tính dựa trên các kíp thi. Ví dụ với kíp 1 sẽ là số ngày nhỏ nhất cần để tổ chức sao cho không có 2 môn

Algorithm 1 Schedule Exams

```
1: procedure ScheduleExams(n, m, d, c, conflict_table)
2:     best_objective ← (n − 1)/4 + 2
3:     scheduled_room ← [0] × n
4:     schedule_section ← [0] × n
5:     best_schedule_room ← []
6:     best_schedule_section ← []
7:     procedure IsValid(subject_index, section, room)
8:         if section/4 + 1 ≥ best_objective then
9:             return False
10:        if d[subject_index] > c[room] then
11:            return False
12:        for subject ← 0 to subject_index − 1 do
13:            if scheduled_room[subject] == room and schedule_section[subject] == section then
14:                return False
15:        for subject ← 0 to subject_index − 1 do
16:            if conflict_table(subject + 1, subject_index + 1) == 1 and schedule_section[subject] == section then
17:                return False
18:        return True
19:    procedure SolutionFound
20:        max_section ← max(schedule_section)
21:        if max_section < best_objective then
22:            best_objective ← max_section/4 + 1
23:            best_schedule_room ← [room for room in scheduled_room]
24:            best_schedule_section ← [section for section in schedule_section]
25:    procedure Backtrack(subject_index)
26:        for section ← 0 to n − 1 do
27:            for room ← 0 to m − 1 do
28:                if IsValid(subject_index, section, room) then
29:                    scheduled_room[subject_index] ← room
30:                    schedule_section[subject_index] ← section
31:                    if subject_index == n − 1 then
32:                        SolutionFound
33:                    else
34:                        Backtrack(subject_index + 1)
35:    Backtrack(0)
```

cùng ngày mà vi phạm ràng buộc. Bài toán khi đó sẽ được quy về bài toán tô màu đồ thị, là 1 bài toán NP-complete.

Chương 3

Constraints Programming

3.1 Mô hình hóa bài toán

Mô hình hóa bài toán dưới dạng một bài toán quy hoạch ràng buộc:

Từ bài toán ta rút ra được các ràng buộc

1. Hai môn học có cùng học sinh phải được xếp lịch thi ở hai timeslot khác nhau
   $s[i] \neq s[j] \quad \forall (i,j) \in C$
2. 2 môn thi có cùng timeslot phải được xếp vào hai phòng khác nhau
   $y[i][k] + y[j][k] \leq 1 \quad \forall i \neq j, i, j = 0...N-1, k = 0...M-1$
3. Môn thi phải được xếp vào phòng thi có sức chứa lớn hơn số sinh viên của môn thi đó
   $y[i][j] * d[i] \leq c[j] \quad \forall i = 0...N-1, j = 0...M-1$
4. Mỗi môn thi phải được xếp vào 1 và chỉ 1 phòng học
   $\sum_{j=0}^{M-1}(y[i][j]) = 1 \quad \forall i = 0...N-1; (a,b) \in C$
5. Các timeslot của các môn học đều không vượt quá *timeslot*
   $s[i] \leq timeslot \quad \forall i = 0...N-1$

Mục tiêu của bài toán là cực tiểu hóa giá trị của *timeslot*

3.2 Triển khai thuật toán

Sử dụng thư viện OR-Tools với bài toán đã được mô hình hóa như trên, sử dụng bộ giải cho bài toán lập lịch với thuật giải sử dụng là CP-SAT để giải bài toán.

Chương 4

Mixed Integer Programming

4.1 Mô hình hóa bài toán

Mô hình hóa bài toán dưới dạng một bài toán quy hoạch tuyến tính:

- N là số môn thi, M là số phòng thi
- $x[i][j][k]$ 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 $D(y) = 0, 1, \ldots, N-1$.
- C là tập các cặp (i, j) - môn học i và môn học j không thể thi cùng lúc.

Từ bài toán ta rút ra được các ràng buộc

1. Trong 1 kíp thi, mỗi phòng thi chỉ có thể được xếp tối đa 1 môn thi
   $\sum_{i=0}^{N-1} x[i][j][k] \leq 1 \quad \forall k = 0...N-1; j = 0...M-1$
2. Mỗi môn thi đều phải được xếp lịch thi vào duy nhất 1 kíp và 1 phòng thi
   $\sum_{k=0}^{N-1} \sum_{j=0}^{M-1} x[i][j][k] = 1 \quad \forall i = 0...N-1$
3. Môn thi phải được xếp vào phòng thi có sức chứa lớn hơn số sinh viên của môn thi đó
   $\sum_{k=0}^{N-1} x[i][j][k] * d[i] \leq c[j] \quad \forall i = 0 \ldots N-1; j = 0 \ldots M-1$
4. 2 môn thi conflict với nhau thì không thể thi cùng thời gian với nhau
   $\sum_{j=0}^{M-1} (x[a][j][k] + x[b][j][k]) \leq 1 \quad \forall k = 0...N-1; (a, b) \in C$

5. Số kíp thi là y
   $x[i][j][k] * k \leq y \forall i = 0.j = 0...M-1; k = 0...N-1;$

Mục tiêu của bài toán là cực tiểu hóa giá trị của y

4.2 Triển khai thuật toán

Sử dụng thư viện OR-Tools với bài toán đã được mô hình hóa như trên, sử dụng bộ giải cho bài toán lập lịch với thuật giải sử dụng là CBC Mixed Integer Programming để giải bài toán.

Chương 5

Greedy Search

5.1 Chi tiết giải thuật

Giải thuật Greedy Search được nhóm xây dựng theo quy tắc max-matching Sắp xếp lại số sinh viên các môn học và số lượng chỗ ngồi của các phòng đều theo thứ tự giảm dần.

Thuật toán được xây dựng lần lượt từ các môn học có số lượng sinh viên lớn nhất chưa được xếp lớp, cho đến khi tất cả các môn học đã được xếp kíp thi. Cụ thể như sau:

Tại mỗi vòng lặp, thuật toán duyệt lần lượt qua các môn học chưa được sắp xếp phòng thi. Một môn học sẽ được thêm vào kíp hiện tại nếu:

- Môn học đó không có conflict với các môn học khác đang được xếp trong kíp hiện tại
- Còn phòng chưa được sử dụng, có đủ chỗ cho môn học đó và số lượng chỗ ngồi lớn nhất

Thỏa mãn 2 điều kiện trên, môn học sẽ được thêm vào kíp hiện tại, phòng học được sử dụng cho môn đó sẽ bị đánh dấu là đã sử dụng. Vi phạm 1 trong 2, bỏ qua môn học đó và xét đến môn học tiếp theo. Khi tất cả các phòng học mỗi kíp đều đã bị sử dụng, quay lại duyệt các môn học từ đầu.

5.2 Nhận xét

Nhóm đưa đề ứng dụng giải thuật Greedy thường thấy bằng cách ứng dụng ý tưởng max-matching, sắp xếp các môn học có số lượng sinh viên nhiều

nhất vào những phòng có nhiều chỗ trống nhất và khéo léo đưa vào các ràng buộc để đưa ra lời giải chính xác.

Đánh giá:

- Kết quả: Đối với những bộ dữ liệu không quá lớn kết quả tương đối tốt, với những bộ dữ liệu cực lớn kết quả không kém nhiều so với các giải thuật khác
- Thời gian chạy: Nhanh (khoảng 6-8s cho bộ dữ liệu 1000)
- Cài đặt: Ý tưởng đơn giản, dễ hiểu, dễ cài đặt

Chương 6

Local Search

6.1 Giải thuật SA

Nhóm sử dụng giải thuật SA (Simulated Annealing) để chuyển từ bài toán tối ưu (Optimization) sang bài toán tìm kiếm thỏa mãn ràng buộc (Find feasible solution)

6.1.1 Convert "Optimization" to "Find feasible solution"

Thay vì cố gắng tối thiểu hóa số kíp thi k, số kíp thi k sẽ được cho trước, mục tiêu của giải thuật là kiểm tra xem có thể xếp lịch các môn học thỏa mãn các ràng buộc đã cho và giới hạn trong k kíp hay không.

Số kíp k sẽ được khởi tạo bằng giải thuật Greedy như ở trên và giảm đi 1 (với mục đích tìm xem có thể tìm được lời giải tối ưu hơn Greedy không). Mã giả việc chuyển đổi như sau:

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 6.1: Convert "Optimization" to "Find feasible solution"

Thực hiện thuật toán theo phương pháp Max-matching như giải thuật Greedy đến kíp thứ k thì dừng lại, những môn chưa được sắp xếp sẽ được nhóm lại vào tập $UNASSIGN$. Thuật toán sau đó dựa trên việc tìm kiếm các lân cận để tìm ra các lời giải hợp lệ.

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 6.2: "Find feasible solution"

Hai thành phần chính của thuật toán bao gồm:

- **SA based local search**: Tìm kiếm lời giải lân cận và xem xét dựa trên ý tưởng thuật toán SA
- **swap two section**: Đổi chỗ các môn học giữa 2 kíp thi với mục tiêu giúp giải thuật vượt qua điểm tối ưu cục bộ

6.1.2 SA based local search

Mục tiêu của bài toán sau khi có ràng buộc k kíp sẽ trở thành cố gắng tối thiểu hóa số lượng phần tử của tập $UNASSIGN$, khi số lượng tập $UNASSIGN$ trở về 0 tức là đã tìm được 1 lời giải với k tốt hơn số kíp k tìm được ở giải thuật Greedy.

Dưới đây là mã giả của thành phần **SA based local search**:

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 6.3: "Find feasible solution"

Trong đó, cấu trúc hàng xóm lân cận **Neighbor structure** 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 6.4: "Neighbor structure"

6.1.3 Swap two section

Mục tiêu là đổi chỗ các môn học trong các kíp trong lời giải hiện tại, vẫn thỏa mãn ràng buộc. Sau đó sắp xếp lại các môn học trong tập $UNASSIGN$ vào các kíp còn phòng trống để tối thiểu hóa số môn học chưa được sắp xếp (tập $UNASSIGN$)

Pseudo-code Swap two section

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

```
3.  keep a copy of Copy_X, Copy_Y, Copy_UNASSIGN
4.  Random an element 'e' in section X
5.  Remove 'e' from X
6.  Con(e, Y) <- set of events in Y that have conflict with e
7.  X = X ∪ Con(e, Y) \ {e}
8.  Y = Y ∪ {e} \ Con(e, Y)
9.  if !conflict(X):
10.   X_Accept, X_Reject = Greedy(X)
11.   Y_Accept, Y_Reject = Greedy(Y)
12.   if len(X_Reject) > 0 or len(X_Reject) > 0:
13.     return
14.   else:
15.     for sub in UNASSIGN:
16.       if Con(sub, X) = 0:
17.         move 'sub' from UNASSIGN to X
18.       else if Con(sub, Y) = 0:
19.         move 'sub' from UNASSIGN to Y
20.   X_Accept, X_Reject = Greedy(X)
21.   Y_Accept, Y_Reject = Greedy(Y)
22.   X = X_Accept, Y = Y_Accept, UNASSIGN += X_Reject + Y_Reject
23.   if len(UNASSIGN) < len(Copy_UNASSIGN):
24.       return X, Y, UNASSIGN
```

Hình 6.5: "Swap two section"

6.2 Nhận xét

So với giải thuật Greedy Search, giải thuật trên cho ra kết quả tốt hơn ở một số bộ test có kích thước lớn, tuy nhiên thời gian chạy rất lâu và tương đối phức tạp trong việc căn chỉnh các hyperparameter và cài đặt

Chương 7

Kết quả thử nghiệm

7.1 Các giải thuật chính xác

Bảng 7.1: Kết quả các giải thuật chính xác

Test Case	N	M	K	Value	CP	MIP
data_10_2_(0)	10	2	24	2	0.21	1.69
data_10_2_(1)	10	2	5	3	0.11	0.39
data_10_2_(2)	10	2	37	2	0.11	0.70
data_16_3_(0)	16	3	3	2	61.70	7.30
data_16_3_(1)	16	3	70	3	NA	6.91
data_16_3_(2)	16	3	3	2	NA	5.06
data_20_4_(0)	20	4	153	2	0.63	21.84
data_20_4_(1)	20	4	54	2	3.14	13.40
data_20_4_(2)	20	4	181	2	0.52	NA

7.2 Các giải thuật heuristic

Nhóm thực hiện thí nghiêm trên 10 bộ dữ liệu nhỏ và vừa (kích thước dưới 100), và 12 bộ dữ liệu lớn (kích thước trên 100, với kích thước như Bảng 7.2. Có thể đưa ra nhận xét như sau:

Nhận xét:

- Tổng quan: Do bước khởi tạo của SA chính là giải thuật GS, dựa vào đó để tìm kiếm các giá trị k tối ưu, do đó kết quả của SA luôn tốt hơn GS, và tất nhiên thời gian chạy của SA luôn lớn hơn GS

Bảng 7.2: So sánh giải thuật SA và greedy search

Test Case	**N**	**M**	**K**	**GS Val**	**GS Time**	**SA Val**	**SA Time**
data_16_3_(1)	16	3	70	11	0.002	11	2.78
data_16_3_(2)	16	3	3	6	0.001	6	2.04
data_20_4_(1)	20	4	54	6	0.001	6	1.16
data_20_4_(2)	20	4	181	7	0.001	7	1.73
data_30_6_(1)	30	6	379	7	0.001	7	2.83
data_30_6_(2)	30	6	94	8	0.001	8	2.89
data_60_12_(1)	60	12	1722	8	0.003	8	8.39
data_60_12_(2)	60	12	227	7	0.001	7	8.67
data_80_20_(1)	80	20	2107	7	0.004	7	16.7
data_80_20_(2)	80	20	1936	7	0.004	7	15.4
data_100_10_(0)	100	10	1174	17	0.004	17	31.92
data_100_10_(1)	100	10	3601	20	0.008	**18**	32.6
data_100_10_(2)	100	10	2072	28	0.005	28	40.1
data_200_15_(0)	200	15	7507	26	0.017	26	110
data_200_15_(1)	200	15	227	26	0.016	**25**	126
data_200_15_(2)	200	15	5805	19	0.015	19	105
data_500_20_(0)	500	20	12370	37	0.061	37	622
data_500_20_(1)	500	20	73584	41	0.143	**40**	861
data_500_20_(2)	500	20	19069	41	0.061	41	806
data_500_30_(0)	500	30	47855	33	0.137	33	560

- Với các bộ dữ liệu nhỏ và vừa, 2 giải thuật SA (Simulated Annealing) và GS (Greedy Search) đều đưa ra kết quả giống nhau, thời gian chạy của GS nhanh hơn rất nhiều so với SA do SA không tìm được lời giải với những giá trị k tối ưu hơn.
- Với các bộ dữ liệu lớn, một số test case SA đã đưa ra được lời giải tốt hơn so với GS tuy nhiên do tính phức tạp của giải thuật, làm mất rất nhiều thời gian để tìm ra kết quả. Hơn thế nữa, kết quả của GS đưa ra không chêch lệch quá nhiều so với SA (chỉ từ 1 - 2 kíp)