**ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



**BÁO CÁO MINI-PROJECT**

**Học phần: Tối ưu lập kế hoạch**

**Đề tài: Xếp lịch thi học kỳ**

Giáo viên hướng dẫn: TS. Bùi Quốc Trung

Nhóm 1

Sinh viên thực hiện:

| 1. | Lê Huy Dương | 20194032 |
| --- | --- | --- |
| 2. | Phan Đức Anh | 20193891 |
| 3. | Ngô Hùng Mạnh | 20194112 |
| 4. | Nguyễn Đỗ Tú | 20194200 |

Hà Nội, 07-2022

**MỤC LỤC**

[**1.**](#_heading=h.1t3h5sf) **Tổng quan 3**

[**2.**](#_heading=h.4d34og8) **Phương pháp 3**

[**2.1.**](#_heading=h.2s8eyo1) **Mô hình của Tú 3**

[**2.2.**](#_heading=h.17dp8vu) **Mô hình của Phan Anh 4**

[**2.2.1. Linear Integer Programming 4**](#_heading=h.3rdcrjn)

[**2.2.2. Constraint Programming (CP) 5**](#_heading=h.26in1rg)

[**2.3.**](#_heading=h.lnxbz9) **Mô hình Genetic Algorithm (GA) 6**

[**3.**](#_heading=h.35nkun2) **Thực nghiệm 11**

[**4.**](#_heading=h.1ksv4uv) **Kết luận 11**

# Tổng quan

Hôm nay là thứ hai

# Phương pháp

## Mô hình của Tú

**Đặt biến:**

N là số lượng môn học,

M là số phòng thi,

c là vector 1 chiều chứa số lượng chỗ ngồi ở mỗi phòng,

d là vector 1 chiều chứa số lượng học sinh tham gia thi mỗi môn,

w là vector 2 chiều chứa các cặp môn học bị xung đột với nhau,

k là số lượng kíp thi (biến cần tối ưu),

p là biến vector, dùng để gán kíp thi cho mỗi môn học, khởi tạo tất cả phần tử bằng 0.

p[subject] = slot,

m là biến vector 2 chiều, để gán môn học cho phòng học tại 1 kíp thi, khởi tạo tất cả các phần tử bằng 0.

m[slot][room] = subject

**Mô hình quay lui:**

Hàm TRY(u, kip) với u là biến đếm chỉ số lượng môn học đã được sắp xếp, kip là số lượng kíp cần dùng tại thời điểm đó.

* Đặt điều kiện dừng:
* nếu số lượng môn học đã được sắp xếp hết thì dừng lại.
* nếu số kíp “*kip*” lớn hơn giá trị tối ưu hiện tại k thì dừng lại.
* Thêm hàm kiếm tra: check(u, kip)

Nếu môn *u* thi ở kíp *kip* được thì trả về TRUE, còn lại thì trả về FALSE.

* Bắt đầu quay lui với việc gán môn học cho các phòng tại mỗi kíp, và sau khi hết các trường hợp có thể của kíp thứ *kip*, ta tiếp tục đệ quy hàm TRY(u, kip + 1) để tìm các phương án có thể trong kíp sau.

## Mô hình của Phan Anh

### 2.2.1. Linear Integer Programming

N môn, d[i] là số lượng học sinh thi môn i, M là số phòng thi, c[i] là số chỗ ngồi của phòng i. S[i][j]=1 là môn i và j không được cùng kíp.

Để tổng số ngày thi diễn ra n môn là nhỏ nhất (1 ngày 4 kíp) thì số kíp cần là nhỏ nhất. Số kíp tối đa cần là n.

Đổi biến p về x:

Một môn chỉ thi trong một kíp:

Hai môn không chung một kíp:

Số kíp cần luôn nhỏ hơn biến k:

Tổng số chỗ ngồi của các phòng dành cho môn i phải lớn hơn tổng số sinh viên thi môn đó:

Hai môn thi cùng một kíp không được cùng một phòng:

Hàm mục tiêu:

Mô hình Linear Programming được đề xuất có số lượng constraint là O(n4). Đưa mô hình vào OR Tools để giải cho kết quả chạy được với n, m <= 20.

Nhận xét: mô hình không phù hợp để giải bài toán thực tế.

### 2.2.2. Constraint Programming (CP)

Mô hình CP tương tự với mô hình Linear Programming, điểm khác biệt ở constraint cuối cùng ta sẽ biến đổi về số lượng constraint là O(n3). Ý tưởng là ta sẽ thử từng giá trị k xem có một feasible solution hay không. Tuy nhiên việc thử từng k sẽ chạy rất lâu, ta có nhận xét là nếu với k1 mà không đưa ra được feasible solution thì với mọi k’ < k1 cũng sẽ không có feasible solution. Từ đây ta sẽ chặt nhị phân theo k, với mỗi k tìm xem có feasible solution hay không:

N môn, d[i] là số lượng học sinh thi môn i, M là số phòng thi, c[i] là số chỗ ngồi của phòng i. s[i][j]=1 là môn i và j không được cùng kíp.

Đổi biến p về x:

Một môn chỉ thi trong một kíp:

Hai môn không được chung một kíp:

Được sử dụng tối đa k kíp:

Tổng số chỗ ngồi của các phòng dành cho môn i phải lớn hơn tổng số sinh viên thi môn đó:

Hai môn thi cùng một kíp không được cùng một phòng:

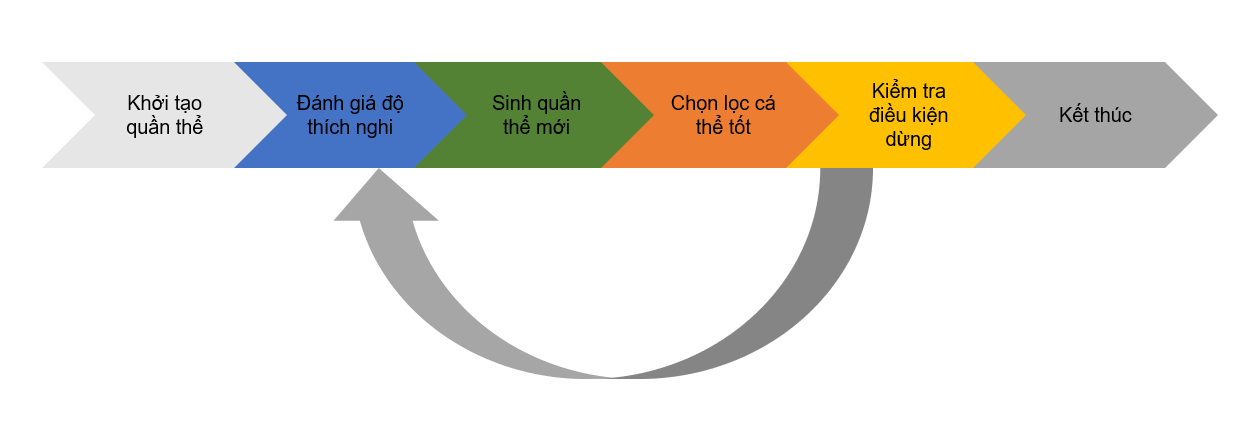
Mô hình khi đưa vào OR Tools được kết quả: chạy tốt với n,m <= 50. Tuy nhiên, với n > 50, mô hình kém hiệu quả và chạy rất lâu. Phần lớn với mỗi k đều trả về UNKNOWN (tức là không biết có lời giải tối ưu hay không).

## Mô hình Genetic Algorithm (GA)

**Tổng quan về GA**

* Bắt đầu được nghiên cứu từ những năm 70 bởi J. Holland, K. DeJong, D. Goldberg.
* Là thuật toán tìm kiếm heuristic, nằm trong các thuật toán tiến hoá (evolutionary algorithm).
* GA mô phỏng quá trình tiến hoá của tự nhiên bằng cách lai ghép sản sinh ra các cá thể mới và chọn lọc các cá thể tốt nhất.
* Các thuật toán GAs có các toán tử: mã hoá cá thể; lai ghép – đột biến; cơ chế chọn lọc sinh tồn.

Mô hình GA được minh hoạ dưới đây



**Mô hình GA\_Duong**

* Mã hoá cá thể

Mỗi cá thể có số phần tử bằng số môn thi: . Giá trị của phần tử thứ k: là kíp của môn thi k.

| 2 | 1 | 1 | 0 | 0 | 1 | 3 | 2 |
| --- | --- | --- | --- | --- | --- | --- | --- |

* Giải mã cá thể

Duyệt từ trái sang phải, các phần tử có giá trị bằng nhau nằm trong cùng một kíp.

Ví dụ: Kíp 0: môn số 3, môn số 4; Kíp 1: môn số 1, môn số 2, môn số 5; Kíp 2: môn số 0, môn số 7; Kíp 3: môn số 6.

| 2 | 1 | 1 | 0 | 0 | 1 | 3 | 2 |
| --- | --- | --- | --- | --- | --- | --- | --- |

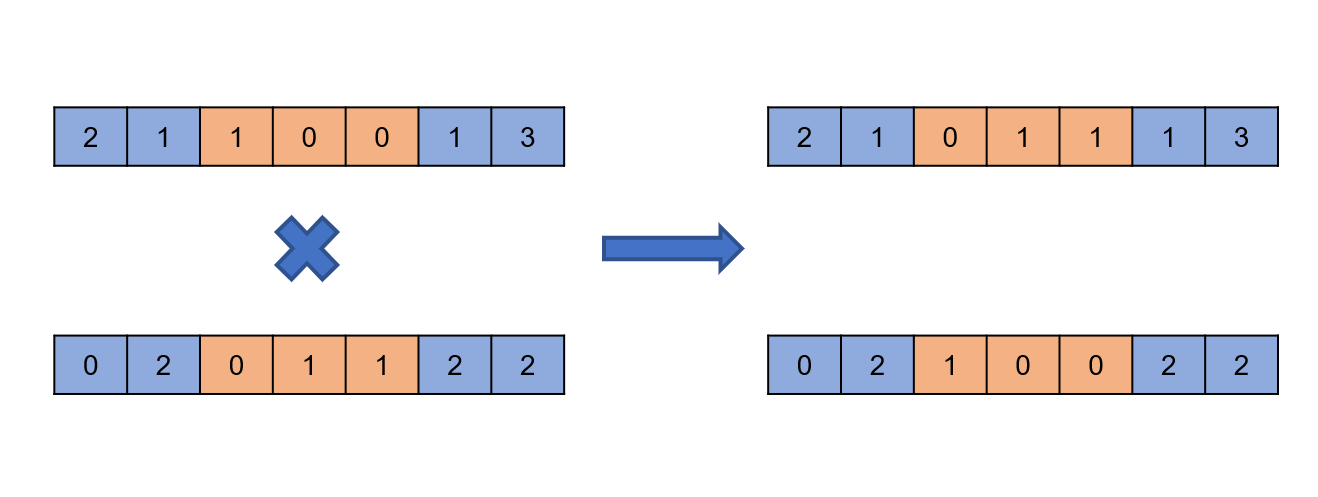
* Đánh giá độ thích nghi

Với mỗi cá thể , giá trị độ thích nghi được tính theo công thức:

Trong đó là tổng số kíp của cá thể, với mỗi ràng buộc bị vi phạm thì hàm sẽ tăng thêm một đơn vị.

* Lai ghép:

Sử dụng lai ghép hai điểm cắt (two point crossover)



* Đột biến

Với mỗi cá thể ựa chọn 2% các phần tử để đột biến.

Đột biến bằng cách thay đổi giá trị của từng phần tử trong cá thể:

Ảnh có chứa văn bản, đồng hồ

Mô tả được tạo tự động

* Chọn lọc

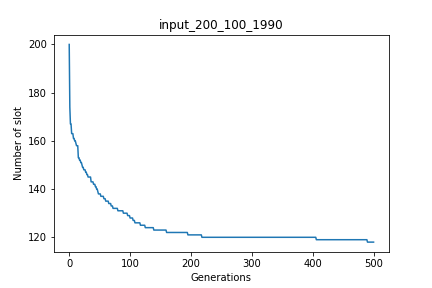
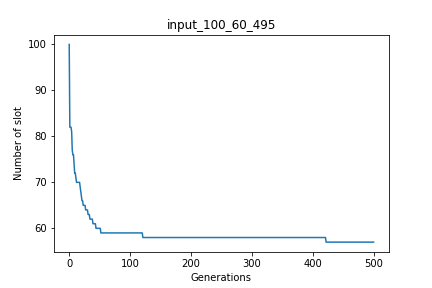
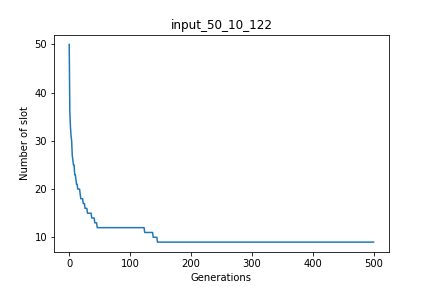
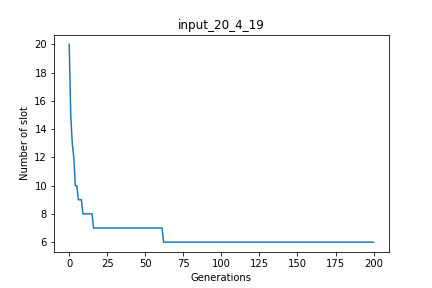
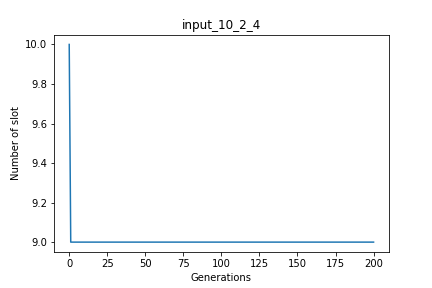
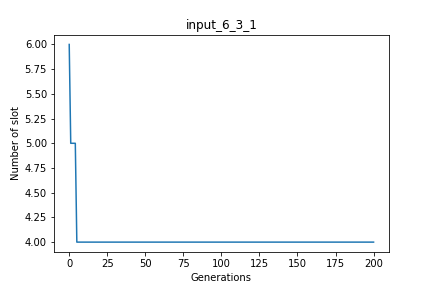
Chọn lọc thế hệ: sử dụng phương pháp chọn lọc các cá thể tốt nhất (elitism) (chọn ra các cá thể có độ thích nghi tốt nhất sinh tồn ở thế hệ tiếp theo).

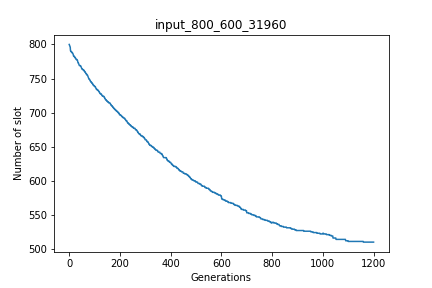
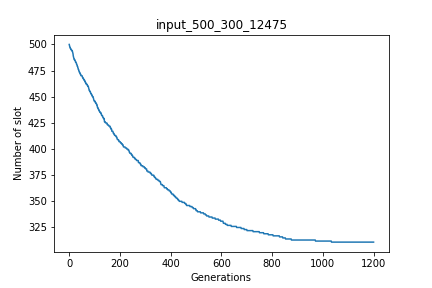
* Thực nghiệm

Tham số cài đặt

| **Dữ liệu** | **nb\_inds** | **nb\_generations** |  |  |
| --- | --- | --- | --- | --- |
| input\_6\_3\_1 | 50 | 200 | 0.8 | 0.6 |
| input\_10\_2\_4 | 50 | 200 | 0.8 | 0.6 |
| input\_20\_4\_19 | 50 | 200 | 0.8 | 0.6 |
| input\_50\_10\_122 | 200 | 500 | 0.8 | 0.6 |
| input\_100\_60\_495 | 200 | 500 | 0.8 | 0.6 |
| input\_200\_100\_1990 | 200 | 500 | 0.8 | 0.6 |
| input\_500\_300\_12475 | 200 | 700 | 0.8 | 0.6 |
| input\_800\_600\_31960 | 200 | 700 | 0.8 | 0.6 |

Kết quả





**Mô hình GA\_Manh**

* Mã hoá cá thể

Mỗi cá thể đại diện cho một lịch thi, bao gồm danh sách các exam (kíp thi, môn thi, phòng thi)

Mỗi exam được mã hóa thành chuỗi nhị phân

| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

* Đánh giá độ thích nghi

Với mỗi cá thể , giá trị độ thích nghi được tính theo công thức:

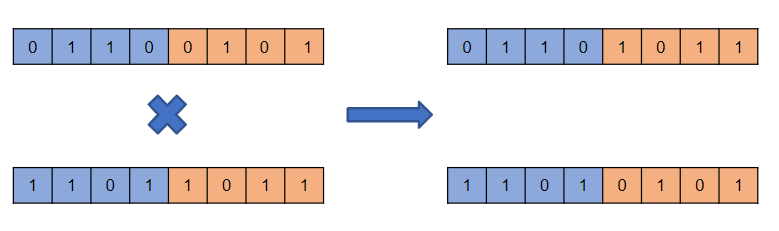
Trong đó là tổng số kíp của cá thể, là tổng số ràng buộc bị vi phạm .

* Chọn lọc

Chọn lọc thế hệ: sử dụng phương pháp chọn lọc các cá thể có độ đo fitness tốt nhất (chọn ra các cá thể có độ thích nghi tốt nhất sinh tồn ở thế hệ tiếp theo).

* Lai ghép:

Sử dụng lai ghép một điểm cắt (one point crossover)



* Đột biến

Với mỗi cá thể lựa chọn 20% các phần tử để đột biến.

Đột biến bằng cách thay đổi giá trị của từng phần tử trong cá thể:

* Thực nghiệm

Tham số cài đặt

| **Dữ liệu** | **nb\_inds** | **nb\_generations** | (xs đột biến) | (xs lai ghép) |
| --- | --- | --- | --- | --- |
| input\_6\_3\_1 | 50 | 300 | 0.2 | 0.8 |
| input\_10\_2\_4 | 50 | 300 | 0.2 | 0.8 |
| input\_20\_4\_19 | 50 | 300 | 0.2 | 0.8 |
| input\_50\_10\_122 | 100 | 500 | 0.2 | 0.8 |
| input\_100\_60\_495 | 100 | 500 | 0.2 | 0.8 |

# Thực nghiệm

# Kết luận