# CHƯƠNG 1. BÀI TOÁN LẬP KẾ HOẠCH GIA CÔNG SẮT

## 1.1 Mô tả bài toán

### 1.1.1 ĐẦU VÀO

### 1.1.1.1 Danh sách lô hàng

bao gồm các lô hàng có cùng một loại vật liệu, với thời gian bắt đầu giao hàng (thời gian bắt đầu vận chuyển từ xưởng sản xuất) có thể khác nhau theo mỗi lô

Ví dụ:

lô hàng 1- thời gian giao hàng, ngày 2/3/2021

* (Khoản) Mục 1: 10 thanh sắt loại A chiều dài 3000mm
* Mục 2: 20 thanh sắt loại A chiều dài 2000mm

lô hàng 2- thời gian bắt đầu giao hàng, ngày 5/3/2021

* Mục 1: 10 thanh sắt loại A chiều dài 5000mm
* Mục 2: 20 thanh sắt loại A chiều dài 7000mm

### 1.1.1.2 Danh sách vật liệu

bao gồm các vật liệu có loại giống với danh sách lô hàng cần gia công, mỗi thanh sắt sẽ có ngày nhập kho hoặc ngày dự kiến nhập kho cố định để phục vụ cho việc tính toán.

* Vật liệu gốchiện tại (B1): các thanh sắt gốc trong kho với chiều dài ban đầu là 13000mm.
* Vật liệu dự định nhập kho => Vật liệu gốc dự định (B2): là các thanh vật liệu gốc trong tương lai (dự định nhập kho) có chiều dài giống vật liệu gốc.
* Vật liệu còn lại (R) => Vật liệu thừa hiện tại (R1): các thanh sắt còn thừa từ các lần gia công trước đó, có chiều dài từ khoảng 500mm đến bé hơn 13000mm.
* Vật liệu dự định (Y) => Vật liệu thừa dự định (R2): các thanh sắt còn thừa đã nằm trong kế hoạch gia công, *nhưng chưa thực sự tồn tại mà sẽ được sản sinh trong các lần gia công tiếp theo*, có chiều dài từ khoảng 500mm đến 13000mm.

### 1.1.1.3 Danh sách Máy cắt

Các máy cắt được đưa vào phải phù hợp với loại sắt được đưa vào, Mỗi máy cắt sẽ có các thuộc tính

- năng suất khác nhau tùy vào loại sắt được đưa vào cắt, được tính theo thời gian trên một nhát cắt, vd: t = 2 mins.

- trạng thái rảnh từ khi nào (vd: máy cắt A rảnh từ 9:15 AM ngày 17/2/2021).

- độ dày lưỡi cưa: mỗi máy cắt sẽ có một lưỡi cưa, lưỡi cưa sẽ có độ dày nhất định.

Ví dụ, một thanh vật liệu có chiều dài 13000mm, được đưa vào máy cắt có độ dày lưỡi cưa là 5mm, cần cắt ra 1 sản phẩm có độ dài 3000mm. Ta sẽ có được độ dài phần thừa còn lại của thanh sắt: remain = 13000 – 3000 – 5 = 9995mm.

### 1.1.2. TIÊU CHÍ

**Tiêu chí bắt buộc:**

* Đúng hạn: trước tiên phải đảm bảo điều kiện bắt buộc đó là cắt xong trước thời gian giao hàng của mỗi lô.

**Tiêu chí lựa chọn:**

rồi tiếp tục xét đến một trong hai tiêu chí bên dưới

* **tiết kiệm**: ưu tiên chọn vật liệu thừa để cắt, nếu không đủ thì sẽ dùng thêm vật liệu gốc, mục tiêu là phần thừa còn lại phải bé nhất có thể
* **nhanh**: sẽ ưu tiên chọn các thanh nguyên liệu gốc (các thanh sắt có sẵn hay dự định nhập kho), nếu các thanh vật liệu gốc không đủ thì sẽ dùng đến các thanh vật liệu thừa, mục tiêu là số lượng thanh sắt đem đi gia công ít nhất có thể

### 1.1.3. ĐẦU RA

* Kế hoạch cắt:
  + Máy A cắt x% khối lượng mục {i} lô {A} (tính ra bao nhiêu thanh sắt loại gì chiều dài bao nhiêu) bắt đầu từ thời điểm {t1}
  + Máy B cắt y% khối lượng mục {j} lô {B} bắt đầu từ thời điểm {t2}
  + …

## 1.2 Một số nghiên cứu liên quan

<đọc các bài báo về cắt thanh nguyên liệu> <tóm tắt ý tưởng chính của các bài báo, ưu điểm, nhược điểm>

# CHƯƠNG 2. THUẬT TOÁN ĐƯỢC ĐỀ XUẤT

## 3.1 Cắt nhanh

Giải thuật nhận vào các dữ liệu như đã mô tả ở phần giới thiệu, và sẽ tính toán trả về chi tiết cách gia công cho từng thanh đơn hàng. Trong đó sẽ chỉ rõ thanh đơn hàng nãy sẽ được cắt bởi thanh nguyên liệu nào, cắt bởi máy cắt số bao nhiêu và cắt tại thời điểm nào.

Kết quả trả về của thuật toán là ba mảng có số lượng phần tử bằng với số lượng danh sách thanh đơn hàng:

* Mảng lưu vị trí các thanh nguyên liệu: trong đó giá trị của các phần tử trong mảng là số thứ tự hay vị trí của thanh nguyên liệu trong danh sách các thanh nguyên liệu.
* Mảng lưu vị trí các máy cắt: trong đó giá trị của các phần tử trong mảng là số thứ tự hay vị trí của máy cắt trong danh sách các máy cắt phù hợp.
* Mảng lưu thời gian cắt các thanh đơn hàng: trong đó giá trị của các phần tử trong mảng là thời gian dự kiến cắt các thanh đơn hàng trên máy cắt.

Phương pháp cắt nhanh có hai hàm mục tiêu chính được sắp xếp theo thứ tự ưu tiên sau:

* Số lượng các thanh nguyên liệu đem ra gia công: mục tiêu là phải ít nhất có thể
* Tổng chiều dài các thanh nguyên liệu còn lại sau khi cắt: Chiều dài còn lại cũng phải ít nhất trong các phương án đảm bảo hàm mục tiêu trên.

Cắt nhanh được thực hiện theo chiến thuật tham lam. Các bước của giải thuật được minh họa ở *Hình 3.1* bao gồm:

1. Sắp xếp thanh nguyên liệu: tiến hành sắp xếp các thanh nguyên liệu theo chiều dài giảm dần.
2. Sắp xếp thanh đơn hàng: tương tự như bước 2, sắp xếp các thanh đơn hàng theo chiều dài giảm dần.
3. Có thanh nguyên liệu dài hơn thanh đơn hàng: kiểm tra nếu không có thanh nguyên liệu nào lớn hơn thanh đơn hàng thì chuyển đến bước 6; ngược lại chuyển đến bước 4.
4. Tìm phương án: sau khi qua được bước 3, tiến hành tìm một phương án với danh sách các thanh nguyên liệu hiện tại (chi tiết sẽ được trình bày ở mục 3.1.2).
5. Bỏ thanh nguyên liệu dài nhất: Sau khi qua bước 4, tiến hành bỏ đi một thanh nguyên liệu dài nhất trong danh sách các nguyên liệu ban đầu để tiếp tục quá trình tìm các phương án có thể cắt.
6. Chọn phương án tốt nhất: sau khi duyệt hết mảng danh sách nguyên liệu và có được các phương án cắt, tiến hành chọn ra một phương án theo tiêu chí ưu tiên là số lượng thanh nguyên liệu cần dùng là ít nhất, nếu có nhiều phương án thỏa mãn tiêu chí đầu, chọn ra phương án mà cho ra tổng chiều dài các thanh nguyên liệu thừa ít nhất.



Hình 3.1 Sơ đồ khối giải thuật cắt nhanh

Khi nhận được các thanh nguyên liệu được trình bày ở trên, tiến hành đi tìm phương án cắt cho bài toán.



Hình 3. 2 Tìm phương án cắt – cắt nhanh

Các bước của phương thức tìm phương án cắt được minh họa ở *hình 3.2* bao gồm:

1. Bước 2 – Đưa thanh nguyên liệu dài nhất lên máy: Chọn thanh dài nhất trong danh sách nguyên liệu được đưa vào để đặt lên máy cắt.
2. Bước 3 – Có thanh đơn hàng ngắn hơn: Kiểm tra lần lượt các thanh trong danh sách đơn hàng, nếu có thanh đơn hàng ngắn hơn hoặc bằng thanh trên máy cắt thì chuyển đến bước 4; ngược lại chuyển đến bước 5.
3. Bước 4 – Cắt theo thanh đơn hàng dài nhất có thể: cắt thanh nguyên liệu trên máy để tạo ra sản phầm là thanh đơn hàng đã được chọn. Sau đó tiếp tục quay lại bước 3 để tiếp tục cắt.
4. Bước 5 – Bỏ thanh nguyên liệu ra khỏi máy: Loại bỏ phần còn lại của thanh nguyên liệu trên máy cắt ra và cho nó vào danh sách vật liệu còn lại để phục vụ việc tính toán tổng chiều dài phần thừa sau khi tìm ra phương án cắt.
5. Bước 6 – Đã hết đơn hàng hoặc không còn thanh nguyên liệu đủ dài: Kiểm tra xem đơn hàng đã cắt xong chưa hoặc có còn thanh nguyên liệu nào phù hợp với tiêu chí cắt hay không, nếu đơn hàng đã cắt xong hoặc không có thanh đơn hàng thì chuyển đến bước 7; ngược lại chuyển đến bước 2.
6. Bước 7 – Ghi nhận phương án mới: nếu đơn hàng đã cắt xong thì tiến hành tính toán sẵn các hàm mục tiêu và ghi nhận phương án mới và kết thúc; ngược lại có nghĩa là không tìm được phương án cắt thì cũng kết thúc quá trình.

## 3.2 Cắt tiết kiệm

### *3.2.1 Tổng quan*

Ngược lại với giải thuật cắt nhanh, có thể tìm được phương án gần như ngay lập tức thì phương pháp cắt tiết kiệm lại không đơn giản. để tìm được một phương án cắt tiết kiệm tối ưu ta phải tốn rất nhiều thời gian và công sức, và theo một tỉ lệ thời gian nhất định thì kết quả của giải thuật cắt này sẽ tỉ lệ thuận với thời gian thực hiện thuật toán. Bài toán mà ta đặt ra ở đây có một không gian tìm kiếm rất rộng nên không thể tìm được phương án tốt nhất mà chỉ tìm được phương án *tối ưu - chấp nhận được* trong một khoảng thời gian cho phép. Giải thuật di truyền là một ứng cử viên sáng giá để có thể giải quyết bài toán đặt ra. sau đây là sơ đồ, một quy trình chung của giải thuật di truyền để giải quyết các vấn đề tìm kiếm.



Hình 3.2 Lược đồ giải thuật di truyền

1. Thuật toán di truyền bắt đầu bằng cách khởi tạo một tập hợp các cá thể (giải pháp) và gọi chúng là một quần thể. Điều này được thực hiện ngẫu nhiên để cung cấp phạm vi phủ sóng đồng đều cho toàn bộ không gian tìm kiếm.
2. Tiếp theo, dân số được đánh giá bằng thể lực cho mỗi cá nhân trong quần thể, ở giai đoạn này thường là sẽ chú ý đến các cá thể phù hợp nhất hiện tại và thể lực trung bình của mỗi dân số.
3. Sau khi đánh giá, thuật toán sẽ quyết định xem nó nên kết thúc tìm kiếm hay không tùy thuộc vào các điều kiện kết thúc được đặt ra. Thông thường điều này sẽ là do thuật toán đã đạt đến một số thế hệ nhất định hoặc một giải pháp thích hợp đã được tìm thấy.
4. Nếu điều kiện dừng không được đáp ứng, quần thể sẽ trải qua một giai đoạn chọn lọc trong đó các cá thể trong quần thể được chọn dựa trên điểm thể lực hay còn gọi là sức khỏe của họ - thể lực càng cao, cá thể càng có cơ hội được chọn.
5. Giai đoạn tiếp theo là áp dụng trao đổi chéo và đột biến cho các cá thể đã chọn, giai đoạn này là nơi các cá thể mới được tạo ra cho thế hệ tiếp theo.
6. Tại thời điểm này, quần thể mới quay lại được tạo ra và quay lại bước đánh giá, quá trình bắt đầu lại. Và mỗi chu kì của vòng lặp này là một thế hệ.
7. Khi điều kiện kết thúc cuối cùng được đáp ứng, thuật toán sẽ thoát ra khỏi vòng lặp và trả lại kết quả tìm kiếm được.

### *3.2.2 Khởi tạo quần thể*

Đối với bài toán trên, ta sử dụng giải thuật quay lui vét cạn để tìm kiếm các cá thể và thêm chúng vào quần thể. Khi sử dụng giải thuật quay lui, thời gian chạy giải thuật khá tốn thời gian, ta không thể chờ để thuật toán vét cạn hết mọi trường hợp mà chỉ giới hạn thời gian chạy để tìm kiếm một số phương án trong một không gian tìm kiếm rộng.

Một quần thể là một phần trừu tượng của một tập hợp các cá thể. Lớp quần thể sẽ được sử dụng để thực hiện các hoạt động cấp nhóm đối với các cá thể của nó, chẳng hạn như tìm kiếm những cá thể mạnh nhất, thu thập số liệu thống kê về toàn bộ quần thể và lựa chọn các cá thể để gây đột biến hoặc lai tạo. Sức khỏe của quần thể sẽ được tính trung bình cộng của các cá thể có trong quần thể.

Một cá thể là một phần trong quần thể, đại diện cho một giải pháp, ứng viên. Phần thông tin cốt lõi của về một cá nhân là *nhiễm sắc thể* của nó, là bản mã hóa của một giải pháp khả thi cho vấn đề đang gặp phải. Nhiễm sắc thể có thể là một chuỗi, một mảng, một danh sách… Trong bài toán này, mỗi cá thể có ba nhiễm sắc thể là ba mảng số nguyên. Một vị trí riêng lẻ trong các nhiễm sắc thể được gọi là gen, và đây là những mảnh ghép có thể được điều khiển hoặc đột biến. Chiều dài của nhiễm sắc thể sẽ bằng số lượng các thanh đơn hàng. Và các gene trong đó sẽ minh họa cụ thể cho nhiễm sắc thể đó.

1. Nhiễm sắc thể - nguyên liệu: đây là nhiễm sắc thể chính, dùng để xác định sức khỏe nổi bật của một cá thể. Mỗi gene trong nhiễm sắc thể này đại diện cho vị trí thanh đơn hàng được cắt bởi thanh nguyên liệu nào. Giá trị gene ở đây chính là vị trí của thanh nguyên liệu trong danh sách nguyên liệu được đưa vào thuật toán.
2. Nhiễm sắc thể - máy cắt: đây là nhiểm sắc thể dùng để mô tả các thanh đơn hàng được cắt bởi máy cắt nào. Giá trị gene ở đây chính là vị trí của máy cắt trong danh sách máy cắt được đưa vào thuật toán.
3. Nhiễm sắc thể - thời gian cắt: đây là nhiểm sắc thể dùng để mô tả thời gian dự kiến cắt cho mỗi thanh đơn hàng. Giá trị gene ở đây chính là thời gian cắt đơn hàng tại máy cắt được định sẵn.

Mỗi cá thể sẽ có mỗi chỉ số *sức khỏe* riêng, để xác định và phân loại *thể lực* của các cá thể trong quần thể; đây là một con số thể hiện một giải pháp tốt như thế nào cho vấn đề của cá thể này. Ý nghĩa của chỉ số *thể lực* sẽ khác nhau tùy thuộc vào vấn đề đặt ra. Và trong ngữ cảnh của bài toán này, thể lực tỉ lệ nghịch với tổng chiều dài phần thừa của các nguyên liệu được đem đi gia công. Giải pháp cho mỗi cá thể cho ra *phần thừa còn lại* càng ít thì cá thể đó càng có thể lực càng cao.

Sau đây là chi tiết giải thuật quay lui:



Hình 3.2.1 Khởi tạo quần thể

1. Tạo quần thể: Khởi tạo hồ chứa hay còn gọi là quần thể rỗng.
2. Phương án hiện tại: Ban đầu tạo một phương án rỗng sau đó phương án này có thể sẽ được cập nhật sau mỗi lần đệ quy.
3. Điều kiện dừng: Kiểm tra thời gian tạo ra các cá thể đã quá thời gian cho phép chưa hoặc số lượng các cá thể được tạo ra đã đạt giới hạn tối đa của một quần thể hoặc đã tìm kiếm hết tất cả các phương án. Nếu thỏa mãn thì nhảy đến *bước 12*, ngược lại chuyển đến *bước 4*.
4. Điều kiện thêm vào quần thể: Kiểm tra phương án hiện tại đã cắt đủ cho danh sách đơn hàng chưa, nếu đã đủ thì tiến hành *bước 10*; sai thì chuyển đến *bước 5*.
5. Điều kiện Chọn nguyên liệu: Có thể hiểu như điều kiện duyệt danh sách nguyên liệu. Nếu đủ điều kiện tiến hành bước 6, ngược lại thực hiện *bước 11*
6. Chọn Nguyên liệu: lấy ra thanh nguyên liệu ở vị trí hiện tại của danh sách vật liệu
7. Điều kiện cắt: Kiểm tra nếu thanh nguyên liệu có thể cắt đc cho thanh đơn hàng tương ứng thì chuyển đến *bước 8*, ngược lại quay lại *bước 5*.
8. Cắt: Tiến hành cắt nguyên liệu cho đơn hàng, sau đó chuyển đến *bước 9.*
9. Cập nhật phương án hiện tại: Thêm vào phương án hiện tại dữ liệu mới và chuyển đến *bước 2*.
10. Thêm vào quần thể: Sau khi đã tạo ra một cá thể hoàn chỉnh thì thêm phương án đó vào quần thể rồi chuyển đến *bước 11*.
11. Xóa phần tử cuối của phương án hiện tại: Sau khi đã tìm được 1 cá thể mới và thêm vào quần thể thì tiến xóa phần tử cuối cùng của cá thể đó và quay lại trạng thái trước đó (trạng thái cận hoàn chỉnh) và tiến hành tìm kiếm một phương án mới (giải thuật quay lui); tiếp tục quay lại bước 5.

### *3.2.2 Đánh giá độ thích nghi của quần thể*

Sau khi tạo ra một quần thể, ta tiến hành đánh giá lại quần thể đó, dựa vào thể lực của các cá thể trong quần thể, độ thích nghi của của quần thể chính là tổng sức khỏe (fitness) của mỗi cá thể trên tổng dân số của quần thể.

### *3.2.3 Điều kiện dừng*

Trong bài toán này, điều kiện dừng bao gồm các ý sau:

1. Thời gian chạy thuật toán: nếu tồng thời gian chạy đạt đến ngưỡng quy định thì tiến hành dừng thuật toán và trả về kết quả.
2. Giới hạn số thế hệ (generation limited): nếu số *thế hệ* hay số *đời di tuyền* trong quần thể đạt đến ngưỡng quy định thì tiến hành dừng thuật toán và trả về kết quả.
3. Tốc độ phát triển của các đời con: sau một số thế hệ n đời di truyền nhất định, nếu các cá thể đời sau không xuất hiện nhân vật xuất chúng hơn cá thể mạnh nhất của các đời trước thì tiến hành dừng thuật toán và trả về kết quả.

Các hằng số trên sẽ được quy định sẵn trước khi thực hiện thuật toán.

### *3.2.4 Chọn lọc tự nhiên*

Sau khi đã đánh giá được quần thể, các cá thể có khả năng sinh tồn tốt hơn sẽ có cơ hội được chọn lọc và sinh sản nhiều hơn các cá thể còn lại. Ta tiến hành chọn lựa chúng để bước vào giai đoạn tiếp theo. Tuy nhiên đối với bàn toán cụ thể này, sẽ áp dụng thêm một chút khác biệt. Với mong muốn sẽ tìm thấy một số thành phần may mắn trúng với câu “Vịt hóa Thiên Nga”, trong quần thể sẽ chọn lọc một lượng cá thể cá biệt có sức khỏe kém để làm phong phú thêm các loại gene trong quần thể và phục vụ cho việc đột biến sau này.

### *3.2.5 Lai tạo*

Kế tiếp là giai đoạn sinh sản. Áp dụng phép lai chéo với quần thể, được coi là “Giao phối”, chọn lọc các cá thể cha mẹ và lai tạo chúng để tạo ra các cá thể con. Các cá thể con sẽ được kế thừa các đặc tính từ cả cha và mẹ. các cá thể con có thể sẽ thích nghi tốt hơn, hoặc kém hơn. Vì thế, để bảo tồn các gene tốt trong quần thể, trong quá trình lai tạo sẽ không tiến hành lai tạo một số lượng cá thể nhất định có sức khỏe tốt nhất, thường thì số lượng này rất ít chỉ chiếm khoảng 1%. Sau đây là chi tiết sơ đồ khối minh họa quá trình lai tạo giữa hai cá thể cha mẹ.



Hình 3.2.5 Trao đổi chéo gene giữa hai cá thể cha mẹ

1. Bước 2 – Chọn cá thể cha: chọn ngẫu nhiên một cá thể cha trong quần thể với điều kiện được nêu ở phần mô tả trên.
2. Bước 3 – Chọn cá thể mẹ: Chọn ngẫu nhiên một cá thể có sức khỏe lớn hơn sức khỏe của quần thể
3. Bước 4 - Gene của con bằng gene của cha: tiến hành sao chép tất cả các gene của cha gắn vào cá thể con. sau đó tiếp tục quá trình lai ghép.
4. Bước 5 - Tìm chuỗi gene liên tục trong con mà khác của mẹ
5. Bước 6 – Tìm thấy: nếu tìm thấy chuỗi gene liên tục trong con mà khác của mẹ thì tiến hành bước 7; ngược lại chuyển đến bước 10.
6. Bước 7 – Thử thay bằng đoạn gene tương ứng của mẹ: thay đoạn gen được tìm thấy ở bước 5 vào trong cá thể con, chuỗi gene này trong cá thể mẹ nằm ở vị trí nào thì khi chuyển sang cá thể con cũng phải chọn đúng vị trí đó để thay.
7. Bước 8 – Cá thể con lai tạo tốt hơn: kiểm tra sức khỏe của cá thể con nếu tốt hơn cá thể của cha mẹ thì đi đến bước 9; ngược lại tiếp tục quay lại bước 5.
8. Bước 9 – Ghi nhận phép lai: khi tạo ra được cá thể con có sức khỏe tốt, tiến hành thêm chúng vào quần thể.

### *3.2.6 Đột biến*

Dễ nhận thấy rằng, nếu chỉ bằng việc sinh ngẫu nhiên và lai tạo sẽ rất khó để tìm được nghiệm. Trừ khi cá thể khởi tạo phù hợp luôn với yêu cầu đề, tức là có đáp án ngay từ đầu – không gian tìm kiếm nhỏ. Đột biến chính là nguyên liệu của *chọn lọc tự nhiên*, bằng việc lựa chọn ngẫu nhiên các vị trí và thay thế bằng một kí tự ngẫu nhiên nào đó – *đột biến điểm*.

Ứng dụng đột biến vào quần thể. Đột biến ảnh hưởng đến các cá thể hơn là quần thể. Ta xem xét từng cá nhân trong quần thể, và nếu họ đủ may mắn (hoặc không may mắn) ta sẽ áp dụng một số biến đổi ngẫu nhiên vào nhiễm sắc thể của chúng. giống như ở mục *lai tạo,* loại đột biến được áp dụng phụ thuộc vào vấn để cụ thể mà ta đang giải quyết. Trong trường hợp này, chúng ta phải tìm kiếm các vị trí gene chưa tốt và thay thế chúng bởi các gene khác để cường hóa sức mạnh của cá thể đó. Sau đây là các bước của phương thức đột biến trên một cá thể được mô phỏng bằng sơ đồ khối bên dưới.



Hình 3.2.6 Đột biến cá thể

1. Bước 2 - Chọn cá thể đột biến: như đã mô tả ở phần đầu, ta sẽ có hai nhóm lựa chọn chính đó là những cá thể tốt, và những cá thể kém (tồi tệ nhất) để tiến hành đột biến.
2. Bước 3 – Tính lượng nguyên liệu còn lại: với mỗi cá thể đại diện cho một giải pháp cắt sắt. ta tiến hình tính toán phần thừa nguyên liệu còn lại khi đi theo giải pháp này. Có thể là tận dụng các thanh đã bị cắt trong phương pháp hoặc những thanh chưa được cắt nhưng có ưu thế hơn về phần thừa nếu được lựa chọn.
3. Bước 4 – Vị trí gene = 1: Khởi tạo giá trị biến lặp nhiễm sắc thể, đánh dấu vị trí(index) trong nhiểm sắc thể
4. Bước 5 – Vị trí thanh nguyên liệu = 1: Khởi tạo giá trị biến lặp nguyên liệu, đánh dấu vị trí(index) trong danh sách nguyên liệu còn lại.
5. Bước 6 – Tạo phương án mới bằng cách thay thế nguyên liệu vào vị trí gene hiện tại: dùng thanh nguyên liệu hiện tại để cắt thanh đơn hàng tại vị trí đơn hàng chính là vị trí gene. Có thể ta sẽ được phương án hợp lệ mới hoặc không.
6. Bước 7- Đã đến vị trí thanh nguyên liệu cuối: nếu sai chuyển đến *bước 8*; ngược lại chuyển đến *bước 9*.
7. Bước 8 – Tăng vị trí thanh nguyên liệu: bước nhảy là một đơn vị cho mỗi lần tăng, để duyệt hết tất cả các thanh nguyên liệu còn lại. sau đó lại nối tiếp đến *bước 6*.
8. Bước 9 – Đã đến vị trí gene cuối: nếu sai chuyển đến bước 10; ngược lại chuyển đến *bước 11*.
9. Bước 10 – Tăng vị trí gene: bước nhảy là một đơn vị cho mỗi lần tăng, để duyệt hết tất cả các gene trong nhiểm sắc thể. sau đó lại nối tiếp đến *bước 5*.
10. Bước 11 – Chọn phương án thay thế tốt nhất: sau khi duyệt hết nhiễm sắc thể cũng như danh sách nguyên liệu còn lại ta có được một danh sách các phương án thay thế. Ta sẽ chọn một phương án tốt nhất trong đó (mỗi phương án là một cá thể mới, chọn cá thể có sức khỏe tốt nhất) để đặt làm kết quả đột biến – cá thể mới.

# CHƯƠNG 4. THỬ NGHIỆM VÀ ĐÁNH GIÁ

## 4.1 Cài đặt các thuật toán đã đề xuất

### 4.1.1 Thuật toán tham lam

Thuật toán tham lam được cài đặt như sau:

public static String getMessageFromFastCut(String order, String stock) {  
 if (stock.trim().equals(*BLANK*)) return *BLANK*;

// mảng lưu các phương án của bài toán  
 List<FastCutBean> listArn = new ArrayList<>();

int[] arrOrderInit = Arrays.*stream*(order.split(*COMMA*))  
 .map(s -> Integer.*parseInt*(s.trim()))  
 .mapToInt(Integer::intValue)  
 .toArray();  
  
 // sắp xếp mảng order theo thứ tự từ cao đến thấp  
 int[] arrOrder = IntStream.*of*(arrOrderInit)  
 .boxed()  
 .sorted(Comparator.*reverseOrder*())  
 .mapToInt(i -> i)  
 .toArray();  
  
 // sắp xếp lại chỉ số index trong mảng order ban đầu theo thứ tự từ cao đến thấp

int[] sortedIndicesOrder = IntStream.*range*(0, arrOrderInit.length)  
 .boxed()  
 .sorted(Comparator.*comparing*(i -> -arrOrderInit[i]))  
 .mapToInt(ele -> ele)  
 .toArray();  
  
 int[] arrStockInit = Arrays.*stream*(stock.split(*COMMA*))  
 .map(s -> Integer.*parseInt*(s.trim()))  
 .mapToInt(Integer::intValue)  
 .toArray();  
  
 /\*  
 \* sắp xếp mảng stock theo thứ tự từ cao đến thấp  
 \*/  
 int[] arrStock = IntStream.*of*(arrStockInit)  
 .boxed().sorted(Comparator.*reverseOrder*())  
 .mapToInt(i -> i)  
 .toArray();  
  
 /\*  
 \* chỉ số index trong mảng stock theo thứ tự từ cao đến thấp  
 \*/  
 int[] sortedIndicesStock = IntStream.*range*(0, arrStockInit.length)  
 .boxed()  
 .sorted(Comparator.*comparing*(i -> -arrStockInit[i]))  
 .mapToInt(ele -> ele)  
 .toArray();  
  
 int[] arrCheckMaterialCanBeCut = new int[arrStock.length];  
 int numberMaterialRemoved = 0;  
 Arrays.*fill*(arrCheckMaterialCanBeCut, 1);  
  
 *fastCutMain*(listArn, arrCheckMaterialCanBeCut, numberMaterialRemoved, arrStock, arrOrder);  
  
 listArn = listArn.stream()  
 .sorted(Comparator.*comparing*(FastCutBean::getNumberMaterial)  
 .thenComparing(FastCutBean::getRemain))  
 .collect(Collectors.*toList*());  
  
 if (!listArn.isEmpty()) {  
 return best;

} else {  
 return *BLANK*;  
 }  
}

fastCutMain(List<FastCutBean> listArn, int[] arrCheckMaterialCanBeCut,  
 int numberMaterialRemoved, int[] arrStock, int[] arrOrder) {  
 int unit = Integer.*parseInt*(*STEEL\_BLADE\_THICKNESS*);  
 while (true) {  
 int[] arrRemain = new int[arrStock.length];  
 OptionalInt minOder = Arrays.*stream*(arrOrder).min();  
 /\*  
 \* mảng lưu cách cắt toriai  
 \*/  
 int[] arrIndexStockUsed = new int[arrOrder.length];  
 Arrays.*fill*(arrIndexStockUsed, -1);  
 for (int i = 0; i < arrStock.length; i++) {  
 if (arrCheckMaterialCanBeCut[i] == 1) {  
 int remaining = arrStock[i];  
 for (int j = 0; j < arrOrder.length; j++) {  
 if (remaining < minOder.getAsInt()) break;  
 if (remaining >= arrOrder[j] && arrIndexStockUsed[j] == -1) {  
 arrIndexStockUsed[j] = i;  
 remaining -= (arrOrder[j] + unit);  
 if (remaining < 0) {  
 remaining = 0;  
 }  
 }  
 }  
 arrRemain[i] = remaining;  
 }  
 }  
 /\*  
 \* kiểm tra nếu không thể cắt được nữa thì dừng lại  
 \*/  
 if (Arrays.*stream*(arrIndexStockUsed).anyMatch(item -> item == -1)) {  
 break;  
 }  
 /\*  
 \* số lượng thanh sắt cần dùng  
 \*/  
 int numberMaterial = Arrays.*stream*(arrIndexStockUsed).distinct().count();  
 /\*  
 \* tổng phần thừa với cách cắt tương ứng  
 \*/  
 int remain = IntStream.*range*(0, arrRemain.length)

.filter(i -> arrRemain[i] != arrStock[i])

.mapToObj(i -> arrRemain[i]).mapToInt(Integer::intValue).sum();

listArn.add(new FastCutBean(numberMaterial, remain, arrIndexStockUsed));

arrCheckMaterialCanBeCut[numberMaterialRemoved++] = 0;

*fastCutMain*(listArn, arrCheckMaterialCanBeCut,

numberMaterialRemoved, arrStock, arrOrder);  
 }  
}

### 4.1.2 Thuật toán di truyền

các hàm chính của thuật toán:

1. Khởi tạo quần thể
2. void generatePopulation(List<Integer> currentChromosome, List<Integer> stockState) {  
    // Population limit reached!  
    if (Boolean.*TRUE*.equals(isFinishedGen)) {  
    return;  
    }  
    long duration = Duration.*between*(startGen, Instant.*now*()).getSeconds();  
    if (duration > this.generationLimit) {  
    isFinishedGen = true;  
    return;  
    }  
     
    // Get a complete ARN  
    if (currentChromosome.size() == orders.size()) {  
    Individual individual = new Individual(new ArrayList<>(currentChromosome));  
    int weight = getFitnessOfChromosome(currentChromosome, stocks, orders);  
    individual.setFitness(weight);  
    this.population.add(individual);  
    this.realPopulationSize++;  
    return;  
    }  
    // Next Node  
    for (int i = 0; i < stockState.size(); i++) {  
    duration = Duration.*between*(startGen, Instant.*now*()).getSeconds();  
    if (duration > this.generationLimit) {  
    isFinishedGen = true;  
    return;  
    }  
     
    int idx = currentChromosome.size();  
    if (orders.get(idx).equals(stockState.get(i))) {  
    List<Integer> stocksTemp = new ArrayList<>(stockState);  
    // update stocks  
    stocksTemp.set(i, stocksTemp.get(i) - orders.get(idx));  
     
    currentChromosome.add(i);  
     
    generatePopulation(currentChromosome, stocksTemp);  
    currentChromosome.remove(currentChromosome.size() - 1);  
     
    duration = Duration.*between*(startGen, Instant.*now*()).getSeconds();  
    if (duration > this.generationLimit) {  
    isFinishedGen = true;  
    return;  
    }  
    } else {  
     
    if (orders.get(idx) + this.cutWidth <= stockState.get(i)) {  
    List<Integer> stocksTemp = new ArrayList<>(stockState);  
    // update stocks state  
    stocksTemp.set(i, stocksTemp.get(i) - orders.get(idx) - this.cutWidth);  
     
    // add to ARN  
    currentChromosome.add(i);  
     
    generatePopulation(currentChromosome, stocksTemp);  
    // redo add to ARN  
    currentChromosome.remove(currentChromosome.size() - 1);  
     
    if (Boolean.*TRUE*.equals(isFinishedGen)) {  
    return;  
    }  
    duration = Duration.*between*(startGen, Instant.*now*()).getSeconds();  
    if (duration > this.generationLimit) {  
    isFinishedGen = true;  
    return;  
    }  
    }  
    }  
    }  
   }
3. lai tạo

public Population crossoverPopulation(Population population) {  
 // Create new population  
 Population newPopulation = new Population(population);  
  
 List<Integer> orders = newPopulation.getOrders();  
 List<Integer> stocks = newPopulation.getStocks();  
  
 // Loop over current population by fitness  
 for (int populationIndex = this.elitismCount; populationIndex < population.size() \* (1 - this.worstRate); populationIndex++) {  
 Individual parent1 = population.getFittest(populationIndex);  
  
 // Apply crossover to this individual?  
 if (this.crossoverRate > Math.*random*()) {  
 // Initialize offspring  
 Individual offspring = new Individual(parent1);  
 List<Integer> stockRemain = newPopulation.getStockRemain(parent1.getChromosome(), stocks, orders);  
  
 // Find second parent  
 Individual parent2 = selectParent(population);  
  
 // Loop over genome  
 for (int geneIndex = 0; geneIndex < parent1.getChromosomeLength(); geneIndex++) {  
 // Use half of parent1's genes and half of parent2's genes  
 int gene1 = offspring.getGene(geneIndex);  
 int gene2 = parent2.getGene(geneIndex);  
 if (gene1 != gene2 && stockRemain.get(gene2) >= orders.get(geneIndex)) {  
 if (stockRemain.get(gene2).equals(orders.get(geneIndex))) {  
 stockRemain.set(gene2, 0);  
 offspring.setGene(geneIndex, gene2);  
 }  
 int cutWidth = newPopulation.getCutWidth();  
 if (stockRemain.get(gene2) >= orders.get(geneIndex) + cutWidth) {  
 stockRemain.set(gene2, stockRemain.get(gene2) - orders.get(geneIndex) - cutWidth);  
 offspring.setGene(geneIndex, gene2);  
 }  
 }  
 }  
  
 double currentFitness = newPopulation.getFitnessOfChromosome(offspring.getChromosome(), stocks, orders);  
  
 if (currentFitness < offspring.getFitness()) {  
 offspring.setFitness(currentFitness);  
 // Add offspring to new population  
 newPopulation.setIndividual(populationIndex, offspring);  
 }  
 } else {  
 // Add individual to new population without applying crossover  
 newPopulation.setIndividual(populationIndex, parent1);  
 }  
 }  
 return newPopulation;  
}

1. đột biến

void mutate(Population newPopulation, List<Integer> chromosome, int worstPosition, int worstValue, int currentFitness) {  
 List<Integer> orders = newPopulation.getOrders();  
 List<Integer> stocks = newPopulation.getStocks();  
  
 List<Integer> stockTemp = newPopulation.getStockRemain(chromosome, stocks, orders);  
 int bestGapOfAll = Integer.*MAX\_VALUE*;  
 int finalMoveTo = -1;  
 int finalFromPosition = -1;  
  
 for (int i = 0; i < orders.size(); ++i) {  
 int bestGap = Integer.*MAX\_VALUE*;  
 int moveTo = -1;  
 int fromPosition = -1;  
 for (int j = 0; j < stocks.size(); ++j) {  
 if (j == chromosome.get(i)) {  
 continue;  
 }  
 if (stockTemp.get(j) >= orders.get(i)) {  
 List<Integer> chromosomeTemp = new ArrayList<>(chromosome);  
 chromosomeTemp.set(i, j);  
 int sumRemain = newPopulation.getFitnessOfChromosome(chromosomeTemp, stocks, orders);  
 if (stockTemp.get(j).equals(orders.get(i)) && bestGap > sumRemain) {  
 bestGap = sumRemain;  
 moveTo = j;  
 fromPosition = i;  
 }  
  
 int cutWidth = newPopulation.getCutWidth();  
 if (stockTemp.get(j) >= orders.get(i) + cutWidth && bestGap > sumRemain) {  
 bestGap = sumRemain;  
 moveTo = j;  
 fromPosition = i;  
 }  
 }  
 }  
  
 if (bestGapOfAll > bestGap) {  
 bestGapOfAll = bestGap;  
 finalMoveTo = moveTo;  
 finalFromPosition = fromPosition;  
 }  
 }  
  
 if (finalFromPosition < 0 || finalMoveTo < 0) {  
 return;  
 }  
  
 chromosome.set(finalFromPosition, finalMoveTo);  
  
 // mutant individual  
 Individual mutant = new Individual(chromosome);  
 mutant.setFitness(newPopulation.getFitnessOfChromosome(chromosome, stocks, orders));  
  
 if (mutant.getFitness() < worstValue && mutant.getFitness() < currentFitness) {  
 newPopulation.getIndividuals().set(worstPosition, mutant);  
 }  
}

## 4.2 Cài đặt thuật toán quy hoạch tuyến tính

Để có thể so sánh và đánh giá kết quả so với thuật toán tham lam và thuật toán di truyền, chúng tôi sử dụng thuật toán quy hoạch tuyến tính bằng phương pháp đơn hình được nghiên cứu trong bài báo [1]. Phương pháp này đòi hỏi đầu vào đồng nhất, nghĩa là các thanh nguyên liệu đầu vào phải có cùng chiều dài

Công thức quy hoạch tuyến tính như sau:

*(formula)*

thuật toán này có thể cài đặt dễ dàng trong excel.

## 4.3 So sánh kết quả chạy các thuật toán

So sánh hai tiêu chí của của các thuật toán.

1. Số lượng thanh đem đi gia công là ít nhất (trực tiếp liên quan đến thời gian)
2. Tỉ lệ dư thừa = Tổng chiều dài các thanh dư thừa / Tổng chiều dài các thanh nguyên liệu sử dụng

### 4.3.1 Trường hợp thử nghiệm thứ nhất

Dữ liệu đầu vào bao gồm:

* Lô hàng một cần cắt 300 thanh 1,250m, 200 thanh 1,2m, 400 thanh 1m.
* Trong kho có 600 thanh nguyên liệu có chiều dài là 11,7m; 100 thanh 5,623m; 100 thanh 1,009m; 200 thanh 1,64m.
* Có ba máy cắt, độ dày lưỡi cắt thép là 5mm.

Kết quả của các thuật toán như sau:

1. Cắt nhanh (giải thuật tham lam)

* Số lượng thanh đem đi gia công: 100 thanh
* Tỉ lệ dư thừa: 13%

1. Cắt tiết kiệm (giải thuật di truyền)

* Số lượng thanh đem đi gia công: 1 thanh
* Tỉ lệ dư thừa: 13%

1. Giải thuật đơn hình – quy hoạch tuyến tính (không khả thi)

### 4.3.2 Trường hợp thử nghiệm thứ hai

Dữ liệu đầu vào bao gồm:

1. Lô hàng một cần cắt 2000 thanh 2m.
2. Lô hàng hai cần cắt 1000 thanh 3m
3. Trong kho có 10000 thanh nguyên liệu có chiều dài là 11,7m
4. Có ba máy cắt, độ dày lưỡi cắt thép là 5mm.

Kết quả của các thuật toán như sau:

1. Cắt nhanh (giải thuật tham lam)

* Số lượng thanh đem đi gia công: 7 thanh
* Tỉ lệ dư thừa: 15%

1. Cắt tiết kiệm (giải thuật di truyền)

* Số lượng thanh đem đi gia công: 7 thanh
* Tỉ lệ dư thừa: 15%

1. Giải thuật đơn hình – quy hoạch tuyến tính

* Số lượng thanh đem đi gia công: 7 thanh
* Tỉ lệ dư thừa: 15%

### 4.3.3 Trường hợp thử nghiệm thứ ba

Dữ liệu đầu vào bao gồm:

1. Lô hàng một cần cắt 1000 thanh 3m, 2000 thanh 2m, 3000 thanh 5m.
2. Lô hàng hai cần cắt 4000 thanh 5m, 5000 thanh 7m, 2000 thanh 3m.
3. Trong kho có 20000 thanh nguyên liệu có chiều dài là 11,7m; 1500 thanh; 3000 thanh 7,995m.
4. Có ba máy cắt có các tham số sau: mất một phút để cắt một nhát cắt, thời gian rảnh từ 13h ngày 2021/02/20; độ dày lưỡi cắt thép là 0mm.

Kết quả của các thuật toán như sau:

1. Cắt nhanh (giải thuật tham lam)
   * Số lượng thanh đem đi gia công: 80 thanh
   * Tỉ lệ dư thừa: 17%
2. Cắt tiết kiệm (giải thuật di truyền)
   * Số lượng thanh đem đi gia công: 92 thanh
   * Tỉ lệ dư thừa: 13%
3. Giải thuật đơn hình – quy hoạch tuyến tính (không khả thi)

### 4.3.4 Trường hợp thử nghiệm thứ bốn

Dữ liệu đầu vào bao gồm:

1. Lô hàng một cần cắt 1000 thanh 3m, 2000 thanh 2m, 3000 thanh 5m, 4000 thanh 7m
2. Trong kho có 100 thanh nguyên liệu có chiều dài là 11,7m
3. Có ba máy cắt, độ dày lưỡi cắt thép là 5mm.

Kết quả của các thuật toán như sau:

1. Cắt nhanh (giải thuật tham lam)
   * Số lượng thanh đem đi gia công: 55 thanh
   * Tỉ lệ dư thừa: 22%
2. Cắt tiết kiệm (giải thuật di truyền)
   * Số lượng thanh đem đi gia công: 62 thanh
   * Tỉ lệ dư thừa: 31%
3. Giải thuật đơn hình – quy hoạch tuyến tính
   * Số lượng thanh đem đi gia công: 55 thanh
   * Tỉ lệ dư thừa: 22%

|  |  |  |
| --- | --- | --- |
| Thuật toán | Số lượng thanh sử dụng | Tỉ lệ dư thừa |
| Đơn hình |  |  |
| Cắt nhanh  Cắt tiết kiệm |  |  |

### 4.3.5 Nhận xét

Từ các trường hợp kiểm thử ở trên, nhận thấy rằng có hai trường hợp xảy ra:

* Chỉ sử dụng các thanh nguyên liệu mới: thuật toán tham lam và thuật toán đơn hình cho cùng một kết quả tối ưu và kết quả đó là tối ưu nhất. Thuật toán di truyển sẽ cho kết quả xấu hơn hai kết quả trên nếu dữ liệu đầu vào lớn.
* Sử dụng cả các thanh nguyên liệu mới và các thanh nguyên liệu thừa từ các lần gia công trước: trường hợp này không thể áp dụng giải thuật đơn hình. giải thuật di truyền đã chiếm ưu thế hơn so với giải thuật tham lam đối với tiêu chí tiết kiệm.

Từ những nhận xét trên, chúng tôi đưa ra khuyến nghị, nếu cần tận dụng các thanh dư thừa thì nên áp dụng giải thuật cắt tiết kiệm; ngược lại chỉ sử dụng các thanh nguyên liệu mới thì áp dụng giải thuật cắt nhanh.