ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

NGUYỄN HỮU MÙI

THUẬT TOÁN VÀ CÁC BÀI TOÁN LỊCH BIỂU

LUẬN ÁN TIẾN SĨ CÔNG NGHỆ THÔNG TIN

Hà Nội – 2013

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

NGUYỄN HỮU MÙI

THUẬT TOÁN VÀ CÁC BÀI TOÁN LỊCH BIỂU

Chuyên ngành: Khoa học máy tính

Mã số: 62 48 01 01

LUẬN ÁN TIẾN SĨ CÔNG NGHỆ THÔNG TIN

NGƯỜI HƯỚNG DẪN KHOA HỌC:

1. PGS. TSKH Vũ Đình Hòa
2. PGS. TS Hoàng Xuân Huấn

Hà Nội - 2013

LỜI CẢM ƠN

Về phía cá nhân, tác giả xin bày tỏ lòng biết ơn chân thành tới PGS. TSKH Vũ Đình Hoà, PGS. TS Hoàng Xuân Huấn đã tận tình hướng dẫn tác giả trong quá trình hoàn thành luận án. Tác giả cũng chân thành cảm ơn TS Phạm Thọ Hoàn, Giám đốc Trung tâm khoa học tính toán Trường Đại học Sư phạm Hà Nội đã giúp đỡ tác giả rất nhiều trong quá trình thử nghiệm tại Trung tâm.

Về phía tập thể, tác giả xin chân thành cảm ơn Bộ môn Khoa học máy tính, Khoa Công nghệ thông tin, Trường Đại học Công nghệ; Bộ môn Khoa học máy tính, Khoa Công nghệ thông tin, Trường Đại học Sư phạm Hà Nội đã hết lòng ủng hộ và tạo điều kiện thuận lợi cho tác giả trong thời gian hoàn thành luận án.

Cuối cùng, tác giả vô cùng biết ơn các bàn bè và người thân trong gia đình vì sự cổ vũ to lớn của họ trong suốt thời gian hoàn thành luận án này.

Hà Nội, tháng 09 năm 2013

Nguyễn Hữu Mùi

LỜI CAM ĐOAN

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi. Các kết quả được viết chung với các tác giả khác đều được sự đồng ý của đồng tác giả trước khi đưa vào luận án. Các kết quả nêu trong luận án là trung thực và chưa từng được ai công bố trong các công trình nào khác.

Tác giả

Nguyễn Hữu Mùi

MỤC LỤC

DANH MỤC CÁC KÝ HIỆU VÀ TỪ VIẾT TẮT

1	ACO	Ant Colony Optimization
2	AI	Artificial Intelligence
3	AS	Ant System
4	BB	Branch and Bound
5	CPU	Central Processing Unit
6	FSP	Flow shop Scheduling Problem
7	GA	Genetic Algorithms
8	GLS	Genetic Local Search
9	GT	Giffler and Thompson
10	HTT	Hyper Threading Technology
11	IM	Iterative Improvement
12	JSP	Job shop Scheduling Problem
13	MIP	Mixed Integer linear Programming
14	MPP	Massively Parallel Processor
15	PFSP	Permutation Flow shop Scheduling Problem
16	RISC	Reduced Instructions Set Computer
17	SA	Simulated Annealing
18	SB	Shifting Bottleneck
19	TA	Threshold Acceptance
20	TS	Tabu Search

DANH MỤC CÁC BẢNG

DANH MỤC CÁC HÌNH VẼ

MỞ ĐẦU

- **Lý do chọn đề tài**

Lập lịch là một trong những chủ đề quan trọng thuộc lĩnh vực vận trù học xuất hiện từ đầu những năm 1950. Mục tiêu chính của lập lịch là phân phối tài nguyên dùng chung một cách hiệu quả nhất cho các tác vụ đồng thời trong toàn bộ thời gian xử lý. Các bài toán lập lịch rất đa dạng, chúng xuất hiện trong các lĩnh vực khác nhau như: Sản xuất, chăm sóc sức khỏe, giáo dục đào tạo, xử lý tính toán, vận tải,... Trong lĩnh vực sản xuất, các tác vụ thường được xem như là các công việc, các tài nguyên là các máy. Trong bệnh viện, các tác vụ là các bệnh nhân và các tài nguyên là các y tá, các giường bệnh, các trang thiết bị y tế được yêu cầu để điều trị các bệnh nhân. Trong giáo dục đào tạo, các tác vụ là các lớp học và các tài nguyên là các giáo viên, các phòng học, các sinh viên,... Các ví dụ khác về lập lịch bao gồm các bài toán vận chuyển (chẳng hạn như bài toán người du lịch, lập lịch hàng không, lập lịch tầu hỏa,...), các bài toán lập lịch tính toán (chẳng hạn như lập lịch CPU, lập lịch phân công,...).

Trong những năm qua, rất nhiều các công trình nghiên cứu về lập lịch với các giải pháp khác nhau đã được đề xuất, từ các tiếp cận chính xác đến các tiếp cận gần đúng và gần đây là các tiếp cận lai kết hợp đồng thời nhiều kỹ thuật với nhau. Các nhà nghiên cứu về lập lịch cũng rất đa dạng, họ hoạt động trong nhiều lĩnh vực rất khác nhau như: Các nhà nghiên cứu khoa học, các nhà khoa học quản lý và thậm chí cả các công nhân trực tiếp sản xuất. Trong những năm qua, nhiều nhà nghiên cứu thuộc các lĩnh vực tưởng chừng như không liên quan gì tới lập lịch như: Sinh học, di truyền học, thần kinh học,... cũng đã có rất nhiều đóng góp cho lý thuyết lập lịch, đặc biệt là sự

đóng góp của họ vào các phương pháp luận mới đầy triển vọng như mạng nơ và tính toán tiến hóa. Chẳng hạn như thuật toán di truyền phỏng theo học thuyết tiến hóa của Darwin được áp dụng khá rộng rãi cho các bài toán lập lịch.

Trong lĩnh vực lập lịch, một mô hình tổng quát nhất về lập lịch đó là bài toán lập lịch job shop (Job shop Scheduling Problem - JSP), bài toán này thuộc lớp *NP*-hard (*NP* là lớp các bài toán giải được bởi một thuật toán không đơn định trong thời gian đa thức) và nổi tiếng là một trong những bài toán tối ưu tổ hợp khó tính toán nhất cho tới nay. JSP cũng là một trong những bài toán được nghiên cứu nhiều nhất và là một mô hình phát triển tốt về lý thuyết lập lịch. Ngoài ra, một động lực khác thúc đẩy mạnh mẽ việc nghiên cứu JSP đó là tính ứng dụng của nó trong thực tiễn cuộc sống và sản xuất.

Ban đầu JSP được giải quyết bởi các tiếp cận tìm ra lời giải chính xác như: Các tiếp cận hiệu suất cao, các mô hình toán học, các kỹ thuật nhánh cận. Các tiếp cận này đưa ra các lời giải tối ưu thực sự cho bài toán. Về mặt lý thuyết, các tiếp cận chính xác đóng vai trò quan trọng và đã được áp dụng thành công cho một số bài toán lập lịch có kích cỡ nhỏ. Tuy nhiên, các tiếp cận này đòi hỏi khá nhiều thời gian thực thi ngay cả với các bài toán cỡ trung bình. Thậm chí, để tìm ra một lời giải thỏa mãn hoàn toàn các ràng buộc của bài toán có thể yêu cầu thời gian tính toán tăng theo hàm mũ trong khi cỡ bài toán chỉ tăng theo tuyến tính.

Trong thực tế, chúng ta thường phải giải quyết các bài toán lập lịch có kích cỡ lớn trong một khoảng thời gian khả thi với các kết quả chấp nhận được (các kết quả này không nhất thiết là phải tối ưu thực sự). Các giải pháp cho JSP đáp ứng đòi hỏi này còn được gọi là các tiếp cận gần đúng. Các tiếp

cận này thường dựa trên các tiến trình tự nhiên như: Vật lý thống kê, sự tiến hóa sinh học hay dựa trên khung cảnh trí tuệ nhân tạo. Bốn tiếp cận gần đúng đã được nghiên cứu và áp dụng phổ biến nhất cho tới nay đó là: Các luật ưu tiên nhanh, các giải thuật heuristic dựa trên nút cổ chai, trí tuệ nhân tạo, các phương pháp tìm kiếm cục bộ và meta-heuristic. Đánh giá tổng quan về các tiếp cận cho JSP sẽ được trình bày chi tiết trong chương 1 của luận án này.

Tuy nhiên, cho tới nay chưa có một tiếp cận nào đã được đề xuất có thể giải quyết triệt để bài toán lập lịch job shop tổng quát, nhất là đối với JSP có nhiều máy và nhiều công việc. Một số vấn đề chính liên quan tới việc giải quyết bài toán này còn tồn tại như sau:

1. Các chuẩn thiết kế thử nghiệm để đánh giá thuật toán mới được đề nghị.
2. Tính hội tụ của các thuật toán mới được đề xuất chưa được chứng minh dựa trên cơ sở toán học.
3. Phương pháp luận cho việc kết hợp các kỹ thuật tìm kiếm khác nhau để tạo ra một giải pháp mạnh cho JSP còn chưa được nghiên cứu một cách đầy đủ.

…

Ở nước ta, việc nghiên cứu về bài toán lập lịch job shop vẫn chưa phát triển. Trong các trường đại học, đại đa số các sinh viên, học viên cao học về công nghệ thông tin vẫn chưa biết tới bài toán này. Trong những năm gần đây đã xuất hiện một số báo cáo khoa học đề cập tới bài toán này. Tuy nhiên, kết quả đạt được còn rất khiêm tốn, chưa tương xứng với tầm quan trọng của bài toán.

Vì những lý do trên, luận án chọn đề tài "*Thuật toán và các bài toán lịch biểu*" làm đối tượng nghiên cứu. Phạm vi nghiên cứu của đề tài chủ yếu tập trung vào thuật toán di truyền và bài toán lập lịch job shop.

- **Mục tiêu của luận án**

Luận án tập trung vào giải quyết một số vấn đề chủ yếu sau đây:

1. Phân tích, đánh giá các tiếp cận đã đề xuất cho JSP để thấy được ưu điểm, nhược điểm của mỗi giải pháp. Trên cơ sở đó đề xuất một giải pháp mới cho bài toán này.

2. Đề xuất một thuật toán di truyền lai mới cho JSP và song song hóa thuật toán nhằm khắc phục độ phức tạp tính toán vốn có của các JSP cỡ lớn.

3. Chứng minh tính hội tụ của thuật toán di truyền lai với mã hóa tự nhiên áp dụng cho JSP mà luận án đề xuất.

- **Đối tượng nghiên cứu, phạm vi nghiên cứu của luận án**

+ Đối tượng nghiên cứu: Thuật toán và các bài toán lịch biểu.

+ Phạm vi nghiên cứu: Ứng dụng thuật toán di truyền giải quyết bài toán lập lịch job shop.

- **Phương pháp nghiên cứu của luận án**

Luận án sử dụng đồng thời nhiều phương pháp nghiên cứu khoa học như:

- Phương pháp nghiên cứu dựa trên tài liệu: Thu thập, phân tích, xử lý thông tin dựa trên các tài liệu như sách, báo, tạp chí,…đã in ấn hoặc công bố trên internet liên quan đến đề tài.

- Phương pháp nghiên cứu phi thực nghiệm: Tham khảo ý kiến của các chuyên gia thông qua các hội thảo trong và ngoài nước.

- Phương pháp nghiên cứu dựa trên thực nghiệm: Thông qua việc thử nghiệm trên các bài toán test chuẩn và đối sánh với các kết quả đã công bố.

- Phương pháp nghiên cứu chứng minh giả thuyết dựa trên các luận cứ khoa học: Chứng minh tính đúng đắn của giải pháp được đề xuất thông qua các luận cứ khoa học.

- **Ý nghĩa khoa học, ý nghĩa thực tiễn của đề tài**

Những đóng góp về khoa học của luận án:

1. Nghiên cứu về tổng quan của bài toán: Phân tích, đánh giá, so sánh các tiếp cận đã áp dụng cho các bài toán lập lịch job shop. Trên cơ sở đó đề xuất một số hướng nghiên cứu cho bài toán này.

2. Nghiên cứu và đề xuất một thuật toán lai mới kết hợp thuật toán di truyền với các kỹ thuật tìm kiếm khác cho bài toán lập lịch job shop. Trong thuật toán đề xuất này, có một số đổi mới trong mã hóa lời giải, toán tử đột biến và toán tử trao đổi chéo. Phương pháp đề xuất này đã được thử nghiệm trên các bài toán test chuẩn và so sánh kết quả với các giải pháp trước đó để chứng tỏ tính vượt trội của nó.

3. Song song hóa thuật toán đã đề xuất cho bài toán lập lịch job shop, thuật toán đã được cài đặt và chạy thử nghiệm cho kết quả tốt và rút ngắn được nhiều lần thời gian thực thi với cùng bộ tham số và dữ liệu vào trong thuật toán tuần tự.

4. Chứng minh tính hội tụ của thuật toán di truyền lai mới với mã hóa tự nhiên cho bài toán lập lịch job shop mà luận án đề xuất.

Ý nghĩa thực tiễn của luận án:

1. Luận án đã được sử dụng làm tư liệu giảng dạy cho môn học chuyên đề tự chọn ở bậc đại học ngành công nghệ thông tin tại Khoa Công nghệ

Thông tin, Trường Đại học Sư phạm Hà Nội.

2. Luận án có thể được sử dụng làm tài liệu tham khảo cho các sinh viên đại học và các học viên cao học ngành công nghệ thông tin làm đề tài về thuật toán di truyền và ứng dụng giải các bài toán tối ưu.

3. Nếu được đầu tư về tài chính và nhân lực, luận án có thể được hoàn thiện và áp dụng để giải quyết các bài toán trong thực tiễn về qui hoạch và tối ưu hóa.

- **Bố cục của luận án**

Ngoài phần mở đầu, kết luận và phụ lục, nội dung của luận án được bố cục thành 4 chương như sau:

CHƯƠNG 1. TỔNG QUAN VỀ THUẬT TOÁN DI TRUYỀN VÀ BÀI TOÁN LẬP LỊCH JOB SHOP

Chương này trình bày vắn tắt về thuật toán di truyền cổ điển (mã hóa nhị phân), phân tích, đánh giá các giải pháp quan trọng nhất cho JSP đã được công bố trong những năm qua. Nhận diện những khó khăn khi giải quyết bài toán lập lịch job shop mà chúng ta cần phải vượt qua trong hiện tại và tương lai. Sau khi phân tích, đánh giá, luận án đề xuất một số hướng nghiên cứu cho bài toán này.

CHƯƠNG 2: HAI BÀI TOÁN CON CỦA BÀI TOÁN LẬP LỊCH JOB SHOP

Bài toán flow shop và flow shop hoán vị là hai trường hợp riêng của bài toán job shop rất thường gặp trong thực tiễn. Chương này trình bày các khái niệm cơ bản liên quan đến hai bài toán con của JSP và thuật toán Johnson cho bài toán flow shop 2 máy và 3 máy có hạn chế điều kiện. Cuối cùng, một thuật toán di truyền mã hóa số tự nhiên được đề xuất cho hai bài toán này.

CHƯƠNG 3: MỘT THUẬT TOÁN DI TRUYỀN LAI MỚI CHO BÀI TOÁN LẬP LỊCH JOB SHOP

Bài toán lập lịch job shop là bài toán lập lịch tổng quát nhất và cũng khó giải quyết nhất. Trong chương này, luận án đề xuất một thuật toán di truyền lai mới cho JSP. Thuật toán này kết hợp thuật toán di truyền với một số kỹ thuật tìm kiếm khác như các luật ưu tiên nhanh, kỹ thuật tìm kiếm lân cận,... Thuật toán được cài đặt và chạy thử nghiệm trên các bài toán test chuẩn, các kết quả tính toán đã khẳng định tính vượt trội của nó. Để khắc phục độ phức tạp tính toán của JSP, thuật toán đã được song song hóa, cài đặt và chạy thử nghiệm trên các bài toán test chuẩn. Kết quả lời giải tối ưu thu được tương tự như thuật toán tuần tự nhưng thời gian tính toán được cải thiện nhiều lần.

CHƯƠNG 4: PHÂN TÍCH TÍNH HỘI TỤ CỦA THUẬT TOÁN DI TRUYỀN LAI MỚI CHO BÀI TOÁN LẬP LỊCH JOB SHOP

Trong chương này, luận án phân tích thuộc tính hội tụ của thuật toán đã đề xuất bằng cách áp dụng các tính chất của xích Markov. Trên cơ sở phân tích xích Markov của thuật toán di truyền, luận án đã chứng minh thuật toán được đề nghị trong chương 3 hội tụ tới tối ưu toàn cục.

CHƯƠNG 1. TỔNG QUAN VỀ THUẬT TOÁN DI TRUYỀN VÀ BÀI TOÁN LẬP LỊCH JOB SHOP

Thuật toán di truyền (Genetic Algorithm - GA) là chiến lược tìm kiếm được thiết kế phỏng theo các quá trình sinh học trong tự nhiên để tối ưu hóa các hàm mục tiêu. GA được đề xuất và nghiên cứu một cách có hệ thống lần đầu tiên bởi John Holland và các cộng sự tại trường đại học Michigan vào năm 1975.

Bài toán lập lịch job shop (JSP) xuất hiện từ những năm 1950. Đã có nhiều tiếp cận khác nhau được đề xuất cho bài toán này, từ các tiếp cận chính xác đến các tiếp cận gần đúng và gần đây là các tiếp cận lai kết hợp đồng thời nhiều kỹ thuật tìm kiếm với nhau. Trong chương này, luận án thống kê, phân tích, đánh giá các giải pháp quan trọng nhất cho JSP đã được công bố trong những năm qua. Cuối cùng, một số hướng nghiên cứu cho JSP và các khó khăn nổi bật khi giải quyết JSP mà chúng ta cần phải vượt qua trong hiện tại và tương lai cũng được đề cập.

1.1. Thuật toán di truyền cổ điển

Khái niệm về thuật toán di truyền đã được biết tới từ những năm 1950. Tuy nhiên, trong thời kỳ này thuật toán di truyền chưa được phát triển thành phương pháp luận mà mới chỉ được sử dụng để giải quyết các bài toán riêng rẽ xuất phát từ sinh học. Vào năm 1975, tại trường đại học Michigan ở Mỹ, John Holland và các cộng sự đã công bố một công trình mang tựa đề "*Adaptation in Natural and Artificial Systems"* [34]. Công trình này được xem như một nghiên cứu bài bản và toàn diện nhất về GA thời bấy giờ.

Hiện nay, GA đã được nghiên cứu và ứng dụng ở hầu hết các quốc gia trên thế giới và đặc biệt phát triển mạnh ở Mỹ, Trung Quốc, Nhật Bản, Hàn

Quốc,... Lý thuyết GA đã được ứng dụng thành công trong rất nhiều lĩnh vực khác nhau như sinh học, khoa học máy tính, kỹ thuật lai ghép, xử lý ảnh,...

1.1.1. Cấu trúc của thuật toán di truyền cổ điển

a. Mã hóa lời giải

Trong GA cổ điển, mỗi lời giải hay cá thể được mã hóa bởi một chuỗi các số nhị phân. Mỗi vị trí trên chuỗi được gọi là một gien và nhận một trong hai giá trị 0 hoặc 1. Như vậy, một lời giải trong GA cổ điển có dạng sau:

1	0	0	1	1	0	0	1	1	1

Hình 1.1 - Một lời giải được mã hóa nhị phân

Một trong những vấn đề cốt lõi của GA là mã hóa các lời giải sao cho phù hợp với đặc thù của bài toán. Trong GA cải tiến sau này, các nhà nghiên cứu thường mã hóa lời giải của bài toán rất đa dạng theo đặc thù của bài toán cần giải quyết. Cheng và những người khác [13] đã tổng kết và phân loại các chiến lược biểu diễn lời giải được áp dụng cho GA thành 2 nhóm bao gồm 9 loại như sau:

Nhóm 1: Mã hóa trực tiếp:

1. Dựa vào các thao tác.
2. Dựa vào các công việc.
3. Dựa vào sự liên quan giữa các cặp công việc.
4. Dựa vào thời gian hoàn thành.
5. Dùng các khóa ngẫu nhiên.

Nhóm 2: Mã hóa gián tiếp:

6. Dựa vào danh sách ưu tiên.

7. Dựa vào luật ưu tiên.

8. Dựa vào đồ thị phân biệt

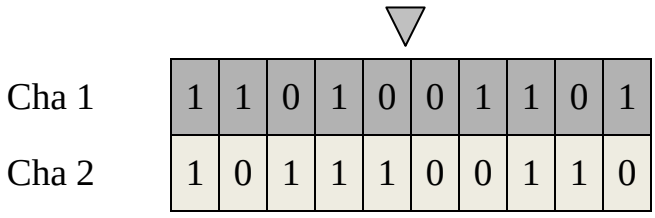
9. Dựa vào các máy.

Các tiếp cận dựa trên GA áp dụng cho JSP thường sử dụng phương pháp mã hóa trực tiếp để mã hóa các lịch biểu như là các cá thể và các toán tử di truyền được sử dụng để tiến hóa các cá thể thành các lịch biểu tốt hơn.

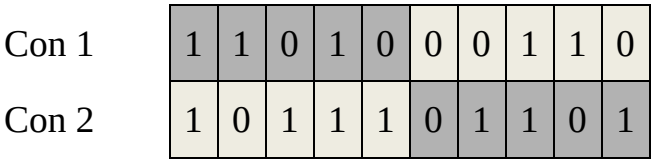
b. Toán tử trao đổi chéo

Toán tử trao đổi chéo kết hợp các đặc tính trên hai cá thể cha để tạo ra một hoặc hai cá thể con mới bằng cách hoán đổi các đoạn gien tương ứng của các cá thể cha. Có một số cách hoán đổi các gien của hai cá thể cha sau đây:

- **Trao đổi chéo một điểm**



Hình 1.2 - Hai cá thể cha cho phép trao đổi chéo



Hình 1.3 - Hai cá thể con sau phép trao đổi chéo

Theo cách này, chọn ngẫu nhiên một vị trí được gọi là điểm bắt chéo. Sau đó ghép đoạn trước điểm bắt chéo của cha 1 với đoạn sau điểm bắt chéo của cha 2 và ngược lại. Ví dụ: Cho hai cá thể cha như trong hình 1.2, phép trao đổi chéo một điểm sau gien thứ 5 cho kết quả hai cá thể con như trong hình 1.3.

- **Trao đổi chéo hai điểm**

Theo cách này, chọn ngẫu nhiên 2 vị trí trên chuỗi cá thể cha. Sau đó tráo đổi đoạn gien nằm giữa 2 điểm đó của 2 cá thể cha cho nhau. Ví dụ: Cho hai cá thể cha như trong hình 1.2, phép trao đổi chéo 2 điểm sau gien thứ 2 và sau gien thứ 8 cho kết quả hai cá thể con như trong hình 1.4.

Con 1	1	1	1	1	1	0	0	1	0	1
Con 2	1	0	0	1	0	0	1	1	1	0

Hình 1.4 - Hai cá thể con sau phép trao đổi chéo 2 điểm

- **Trao đổi chéo đồng nhất**

Phép trao đổi chéo này gieo ngẫu nhiên một đồng xu, số lần gieo bằng số gien của cá thể cha. Nếu kết quả gieo là 1 (mặt sấp) thì lấy gien từ cha 2 còn nếu kết quả gieo là 0 (mặt ngửa) thì lấy gien từ cha 1.

Ví dụ: Cho hai cá thể cha như trong hình 1.2, phép trao đổi chéo đồng nhất với kết quả gieo ngẫu nhiên đồng xu là 1010101011 cho kết quả là cá thể con trong hình 1.5.

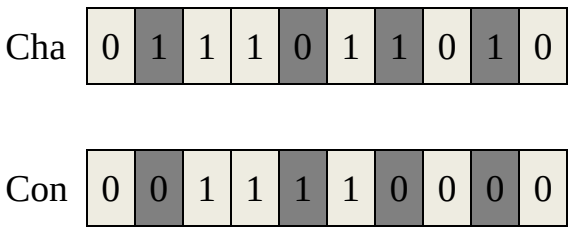
Con	1	1	1	1	1	0	0	1	1	0

Hình 1.5 - Cá thể con sau phép trao đổi chéo đồng nhất

Trong GA cải tiến sau này, phép trao đổi chéo có thể được thay đổi cho phù hợp với bài toán thực tiễn, nhưng vẫn phải đảm bảo nguyên tắc cá thể con được tái tổ hợp các gien từ hai (hoặc có thể nhiều hơn) cá thể cha.

c. Toán tử đột biến

Toán tử đột biến sửa đổi một số gien trong một cá thể cha được chọn một cách ngẫu nhiên bằng cách thay đổi các gien có giá trị 0 thành 1 và ngược lại. Ví dụ: Cho cá thể cha trong hình 1.6, giả sử các gien thứ 2, 5, 7, 9 được chọn để đột biến, chúng ta có cá thể con sau đột biến như trong hình 1.6.



Hình 1.6 - Cá thể cha và cá thể con sau phép đột biến

Trong GA cải tiến, phép đột biến có thể rất đa dạng. Tuy nhiên, vẫn phải đảm bảo nguyên tắc thực hiện sửa đổi một số gien trên một cá thể cha để có một cá thể con.

d. Toán tử chọn lọc

Toán tử chọn lọc sẽ chọn ra một quần thể cho thế hệ tiếp theo của quá trình tiến hóa. Khả năng được chọn của mỗi cá thể tùy thuộc và độ thích nghi (thường là trên cơ sở giá trị của hàm mục tiêu) của nó. Những cá thể có giá trị hàm thích nghi cao hơn sẽ có nhiều khả năng được chọn hơn để trở thành thành viên trong quần thể của thế hệ tiếp theo. Cơ chế chọn lọc này thực hiện theo nguyên lý bánh xe xổ số. Mỗi cá thể trong quần thể có một xác suất chọn lọc được tính theo công thức: p_i = eval(v_i) / F. Trong đó, eval(v_i) là giá trị của hàm thích nghi của cá thể v_i, F là tổng các giá trị thích nghi của quần thể.

Tuy nhiên, trong GA cải tiến sau này thực hiện phép chọn lọc đa dạng hơn. Phép chọn lọc có thể kết hợp cả việc ấn định với chọn lọc ngẫu nhiên theo nguyên tắc bánh xe xổ số.

1.1.2. Một thủ tục đơn giản cho thuật toán di truyền cổ điển

Một quá trình tiến hóa được thực hiện trên một quần thể gồm N cá thể (N tùy theo người dùng chọn). Mỗi cá thể được đánh giá độ tốt xấu theo hàm thích nghi. Tại thế hệ 0, một quần thể $P(0)$ được khởi tạo một cách ngẫu nhiên. Sau đó $P(0)$ được tiến hóa, quá trình tiến hóa diễn ra trong vòng lặp while, quần thể tại thế hệ thứ t được ký hiệu là $P(t) = \{x^t_1, .., x^t_n\}$. Tập lời giải của thế hệ $t + 1$ được xây dựng trong vòng lặp while bằng cách:

1. Áp dụng phép trao đổi chéo cho $p_c \times N$ cá thể và phép đột biến cho $p_m \times N$ cá thể của quần thể thế hệ thứ t, chúng ta được tập lời giải trung gian $P'(t)$. Ở đây, p_c là xác suất trao đổi chéo và p_m là xác suất đột biến.

2. Đánh giá độ thích nghi của mỗi cá thể của $P'(t)$.

3. Chọn lọc $P(t + 1)$ từ $P'(t)$.

```
Procedure GA
Begin
    t ← 0
    Khởi tạo P(t)
    Đánh giá P(t)
    While (not điều kiện dừng ) do
    Begin
        Xây dựng tập lời giải trung gian P'(t) từ P(t)
        Đánh giá P'(t)
        t ← t + 1
        Chọn lọc P(t) từ P'(t-1)
    End
End
```

1.2. Các lớp bài toán *P, NP, NPC* và *NP*-hard

1.2.1. Các lớp bài toán *P* và *NP*

Lớp *P* là lớp các bài toán giải được bởi một thuật toán đơn định trong thời gian đa thức (*P* cũng là lớp các bài toán quyết định giải được trong thời gian đa thức). Lớp *P* có thể được xem là lớp các bài toán có thể giải được trong thời gian thực tế.

Lớp *NP* là lớp các bài toán giải được bởi một thuật toán không đơn định trong thời gian đa thức (*NP* cũng là lớp các bài toán quyết định tương ứng). *NP* là lớp các bài toán không thể giải được trong thời gian thực tế.

Vấn đề đặt ra là $P = NP$? Bài toán này có thể phát biểu như sau:

Mọi bài toán giải được bởi một thuật toán không đơn định với thời gian đa thức thì cũng giải được bởi một thuật toán đơn định nào đó với thời gian đa thức?

Hiển nhiên, thuật toán đơn định là trường hợp đặc biệt của thuật toán không đơn định, vậy $P \subseteq NP$. Nhưng chúng ta không biết bao hàm thức trên có là thật sự hay không, bởi vì cho đến nay người ta vẫn chưa tìm được một bài toán thuộc lớp *NP* nhưng không thuộc lớp *P*, tức là với một bài toán *NP*, người ta chưa tìm được một thuật toán đơn định với thời gian đa thức để giải nó, nhưng cũng không chúng minh được là không tồn tại thuật toán đó. Vì vậy, câu hỏi $P \neq NP$?, hay cũng vậy câu hỏi $P = NP$? vẫn còn bỏ ngỏ.

1.2.2. Các lớp bài toán *NPC* và *NP*-hard

- **Khái niệm qui dẫn**

Cho 2 bài toán quyết định $\mathcal{P}$ và Q. Chúng ta nói $\mathcal{P}$ qui dẫn tới Q (ký hiệu là $\mathcal{P} \prec Q$) nếu tồn tại một hàm có thể tính toán trong thời gian đa thức g

chuyển đổi các thông tin vào của $\mathcal{P}$ thành các thông tin vào của Q sao cho: x là thông tin vào "yes" của $\mathcal{P}$ khi và chỉ khi $g(x)$ là thông tin vào "yes" của Q.

Bổ đề 1: Cho $\mathcal{P}$, Q, R là các bài toán quyết định. Nếu $\mathcal{P} \prec Q$ và $Q \prec R$ thì $\mathcal{P} \prec R$.

- **Bài toán *NPC***

Q được gọi là *NPC* nếu Q thuộc lớp *NP* và mọi bài toán $\mathcal{P}$ thuộc lớp *NP* đều qui dẫn tới Q.

- **Bài toán *NP*-hard**

Q được gọi là *NP*-hard nếu mọi bài toán $\mathcal{P}$ thuộc lớp *NP* đều qui dẫn tới Q.

Thường việc chỉ ra một bài toán thuộc *NP* là không quá khó khăn, vì vậy để chứng minh một bài toán Q là *NPC*, công việc chủ yếu là tìm được một bài toán $\mathcal{P}$ thuộc lớp *NPC* sao cho $\mathcal{P} \prec Q$.

Các bài toán lập lịch là các bài toán tối ưu. Người ta đã chứng minh được đa số các bài toán lập lịch đều là *NP*-hard.

1.3. Tổng quan về bài toán lập lịch job shop

1.3.1. Bài toán lập lịch job shop

JSP nổi tiếng là một trong những bài toán tối ưu tổ hợp khó tính toán nhất cho tới nay. Nó là một trong những bài toán được nghiên cứu nhiều nhất và là một mô hình điển hình về lý thuyết lập lịch, ngoài ra nó cũng được thúc đẩy mạnh mẽ bởi các nhu cầu thực tiễn. Bài toán lập lịch job shop tổng quát được phát biểu như sau:

Cho một tập n công việc $\{J_i\}_{1 \le i \le n}$, mỗi công việc bao gồm m công đoạn (hay thao tác) được xử lý ở trên một tập m máy $\{M_j\}_{1 \le j \le m}$ và thỏa mãn các ràng buộc sau đây:

1. Mỗi công việc phải được xử lý ở trên mỗi máy theo một trình tự cho trước của các thao tác. Trình tự thực hiện các thao tác của mỗi công việc lần lượt trên các máy được gọi là tuần tự công nghệ.

2. Tại một thời điểm mỗi máy chỉ có thể xử lý nhiều nhất là một công việc.

3. Mỗi máy M_j tùy ý đều có khả năng xử lý một công việc J_i nào đó, phần công việc J_i được xử lý trên máy M_j được gọi là thao tác O_{ij}.

4. Mỗi thao tác O_{ij} phải được xử lý liên tục ở trên máy M_j (từ khi bắt đầu xử lý cho tới khi xử lý xong không bị ngắt).

5. Thời gian bắt đầu xử lý và thời gian hoàn thành việc xử lý thao tác O_{ij} được ký hiệu lần lượt là s_{ij} và c_{ij}. Thời gian xử lý thao tác O_{ij} được ký hiệu là p_{ij}.

6. Thời gian hoàn thành việc xử lý tất cả các công việc được gọi là makespan và được ký hiệu là C_{max}.

Việc giải quyết JSP là xác định một lịch biểu (thứ tự xử lý các công việc ở trên mỗi máy) sao cho makespan là nhỏ nhất có thể.

- **Một ví dụ minh họa**

Để làm rõ hơn về bài toán lập lịch job shop, một ví dụ về JSP 3 × 3 được cho trong bảng 1.1 (ví dụ này luận án tham khảo trong [77]). Dữ liệu vào bao gồm tuần tự công nghệ của các máy cho mỗi công việc và thời gian xử lý mỗi công việc ở trên mỗi máy (trong dấu ngoặc đơn).

Bảng 1.1 - JSP 3 công việc, 3 máy

Công việc	Máy (thời gian xử lý)		
1	1 (4)	2 (4)	3 (4)
2	1 (3)	3 (4)	2 (5)
3	2 (4)	1 (3)	3 (2)

Theo bảng 1.1, các thao tác của J_1 được xử lý trên các máy theo trình tự: M_1 (O_{11}), M_2 (O_{12}), M_3 (O_{13}); các thao tác của J_2 được xử lý theo trình tự: M_1 (O_{21}), M_3 (O_{23}), M_2 (O_{22}); các thao tác của J_3 được xử lý theo trình tự: M_2 (O_{32}), M_1 (O_{31}), M_3 (O_{33}).

- **Độ phức tạp của các bài toán lập lịch job shop**

Johnson [41] đã chứng minh rằng bài toán flow shop 2 máy, có thể được giải trong thời gian $O(n\log n)$. Cũng chính Johnson cùng với Garey và Sethi [26] đã chứng minh bài toán flow shop 3 máy là *NP*-hard. Như vậy, việc tìm lịch biểu tối ưu cho bài toán flow shop từ 3 máy trở lên cũng là *NP*-hard. Tuy nhiên, nếu bài toán flow shop 3 máy thỏa mãn điều kiện: $\max p_{i2} \leq \max\{\min p_{i1}, \min p_{i3}\}$ thì nó có thể được giải trong thời gian đa thức [41].

Chỉ có một số ít các trường hợp hạn chế của JSP có thể giải được trong thời gian đa thức như dưới đây:

1. JSP 2 máy với số công đoạn của các công việc không quá 2 ($J2|op \leq 2$), được giải trong thời gian $O(nlogn)$. Thuật giải bài toán cho trường hợp này do Jackson [38] đề xuất bằng cách áp dụng có mở rộng thuật toán của Johnson cho bài toán lập lịch flow shop 2 máy.

2. JSP 2 máy với thời gian xử lý các công đoạn $p_{ij} = 1$ ($J2|\ p_{ij} = 1$) có thể giải được trong thời gian $O(n)$ bởi thuật toán do Hefetz và Adiri [33] đề xuất.

3. Akers [4] đã chứng minh rằng JSP chỉ có 2 công việc ($J|\ n = 2$) có thể được xem như là một bài toán đường đi ngắn nhất và vì thế có thuật giải thời gian đa thức.

4. Dựa trên ý tưởng của Kravchenko và Sotskov [44], Brucker [11] đưa ra thuật toán thời gian đa thức cho JSP 2 máy k công việc ($J2|n = k$) với k là hằng số.

Ngoài các trường hợp đặc biệt đã nêu ở trên, các trường còn lại của JSP đều thuộc lớp *NP*-hard [66].

1.3.2. Các tiếp cận chính xác

Các tiếp cận chính xác tìm ra lời giải tối ưu thực sự của bài toán. Tuy nhiên, thời gian tính toán sẽ tăng theo hàm số mũ hoặc một đa thức bậc cao khi cỡ bài toán chỉ tăng theo tuyến tính. Ba cách tiếp cận chính xác quan trọng nhất đã được áp dụng rộng rãi từ rất sớm cho JSP đó là: Các tiếp cận hiệu suất cao, các mô hình toán học và các kỹ thuật nhánh cận.

a. Các tiếp cận hiệu suất cao

Các tiếp cận hiệu suất cao tìm lời giải tối ưu thực sự cho JSP bằng cách tuân theo một tập các qui tắc để xác định chính xác trình tự xử lý các công việc. Johnson [41] là người đầu tiên đề xuất tiếp cận hiệu suất cao cho bài toán flow shop 2 máy và 3 máy có hạn chế điều kiện. Sau Johnson là các đề xuất của Akers [4], Jackson [38], Hefetz và Adiri [33],…

- **Phân tích, đánh giá**

Các tiếp cận hiệu suất cao tìm ra lời giải tối ưu thực sự cho JSP. Tuy nhiên, với một JSP có n công việc và m máy thì nó có tới $(n!)^m$ lời giải. Do đó, việc liệt kê tất cả các lời giải có thể để tìm ra lời giải tối ưu thực sự là điều không thực tế. Cho tới nay, các tiếp cận hiệu suất cao cho JSP với $n \geq 3$ và $m \geq 3$ vẫn còn bỏ ngỏ. French [25] đã tiên đoán rằng không có các thuật toán hiệu xuất cao cho hầu hết các bài toán lập lịch job shop. Lý do chủ yếu cho vấn đề này là vì các tiếp cận hiệu xuất cao tập trung vào liệt kê tất cả các lời giải có thể để tìm ra lời giải tối ưu thực sự của bài toán.

b. Các mô hình toán học

Một trong các mô hình toán học tiêu biểu và xuất hiện sớm nhất cho JSP là mô hình MIP (Mixed Integer linear Programming) của Manne [48]. MIP bao gồm một chương trình tuyến tính, một tập các ràng buộc tuyến tính, một hàm mục tiêu tuyến tính đơn và một số biến quyết định nguyên. Ở đây các biến nguyên được sử dụng để thi hành các ràng buộc.

Sau MIP là mô hình toán học LR (Lagrangian Relaxation) do Fisher [23] đề xuất lần đầu tiên. Sau đó Van De Velde [70], Della Croce và những người khác [18],... đã có công cải tiến mô hình này. Trong mô hình LR, quyền ưu tiên và các ràng buộc sử dụng các số nhân Lagrangian không âm, hàm mục tiêu sử dụng các điều khoản phạt.

Sau LR là mô hình toán học phân rã do Ashour [7] đề xuất lần đầu tiên. Sau đó, mô hình toán học phân rã được cải tiến bởi một số người khác. Trong mô hình này, bài toán gốc được phân hoạch thành một tập các bài toán con, sau đó các bài toán con được giải tối ưu.

Trong những năm gần đây, các mô hình toán học thường được kết hợp với các giải pháp khác để tạo ra một giải pháp lai cho JSP. Chẳng hạn như kết

hợp mô hình toán học với phương pháp heuristic [49], kết hợp mô hình toán học với các luật ưu tiên [50],...

- **Phân tích, đánh giá**

Mô hình MIP có ưu điểm là đơn giản. Tuy nhiên, số các ràng buộc thường khá lớn dẫn đến độ phức tạp tính toán cao, cho nên không khả thi về thời gian tính toán cho JSP [9]. Fisher và những người khác [22] cũng đã phân tích đánh giá và rút ra kết luận các mô hình LR và mô hình phân rã áp dụng cho JSP cũng thực thi rất tồi. Họ đã chứng tỏ điều này thông qua thử nghiệm trên các bài toán chuẩn của Muth và Thompson [52]. Các phân tích còn rút ra kết luận là ngay cả khi các mô hình toán học được kết hợp với các phương pháp khác hoặc được cải tiến thì vẫn thực thi không tốt và không phù hợp với bài toán khó như là JSP.

c. Các kỹ thuật nhánh cận

G. H. Brooks và C. R. White [10] là những người đầu tiên đề xuất kỹ thuật nhánh cận (Branch and Bound - BB) cho JSP, tiếp sau đó kỹ thuật này đã được phát triển bởi nhiều nhà nghiên cứu khác. Các nhà nghiên cứu thường dùng các bài toán chuẩn của Muth và Thompson [52] để thử nghiệm và chứng minh cho tính hiệu quả của thuật toán mà họ đề nghị. Năm 1985, Carlier và Pinson [12] đã tìm ra lời giải tối ưu thực sự cho bài toán *mt*10 bằng một thuật toán nhánh cận. Sau đó, kỹ thuật này còn được cải tiến bởi nhiều nhà nghiên cứu khác, các cải tiến tập trung vào chiến lược phân nhánh và xây dựng hàm đánh giá cận dưới.

Các kỹ thuật nhánh cận sử dụng cấu trúc cây để biểu diễn không gian lời giải cho bài toán. Việc tìm kiếm các lời giải bắt đầu tại nút gốc của cây và kết thúc khi nút lá được thăm. Mỗi nút tại mức *p* trong cây tìm kiếm biểu diễn một lời giải bộ phận gồm *p* thao tác. Tại mỗi nút này hoạt động phân nhánh

quyết định tập các nút có thể tiếp theo để phát triển cây tìm kiếm. Thủ tục cận tính toán cận dưới và dựa vào cận trên tốt nhất đã biết ở thời điểm hiện tại để quyết định có phát triển tiếp tại nút này hay không. Nếu cận dưới được tính toán lớn hơn cận trên tốt nhất hiện tại thì dừng phát triển nút này. Khi một nút lá hay nút có cận dưới lớn hơn cận trên được thăm thì sự tìm kiếm quay trở lại theo lối cũ tới nút chưa được thăm cao nhất, việc tìm kiếm được tiếp tục từ nút này. Thuật toán dừng khi cây tìm kiếm không thể phát triển được nữa.

Như vậy, trong các kỹ thuật nhánh cận, chúng ta không cần phải duyệt toàn bộ cây không gian trạng thái để tìm ra lời giải tốt nhất. Bằng cách đánh giá cận dưới của các nút sẽ được phát triển tiếp theo, chúng ta có thể cắt bỏ các nhánh không khả thi. Vì thế không gian tìm kiếm được thu hẹp và việc tìm kiếm sẽ nhanh hơn. Cái khó nhất của phương pháp này là việc xây dựng "*hàm đánh giá cận dưới*". Nếu hàm này được xây dựng tốt, sẽ giúp chúng ta cắt bỏ được nhiều nhánh không khả thi, khi đó phương pháp nhánh cận sẽ cải thiện đáng kể so với tiếp cận hiệu suất cao.

- **Phân tích, đánh giá**

Để đánh giá các ưu điểm, nhược điểm của các kỹ thuật nhánh cận, các nhà phân tích đã thông qua các kết quả thử nghiệm trên các bài toán chuẩn của Muth và Thompson [52]. Các phân tích đều đưa ra kết luận chung là các kỹ thuật này thường yêu cầu thời gian tính toán của máy tính rất lớn. Glover và Greenberg [29] đã phân tích, đánh giá và kết luận các kỹ thuật nhánh cận không phù hợp với các bài toán tối ưu tổ hợp khó. Chính vì lý do này mà các kỹ thuật nhánh cận không thích hợp với bài toán có độ phức tạp lớn như JSP.

1.3.3. Các tiếp cận gần đúng

Trong thực tế, chúng ta cần phải giải quyết các bài toán cấp độ lớn trong một khoảng thời gian khả thi với các kết quả chấp nhận được (các kết

quả này không nhất thiết là phải tối ưu thực sự). Các tiếp cận cho JSP đáp ứng đòi hỏi này còn được gọi là các tiếp cận xấp xỉ hay gần đúng. Các phương pháp này thường dựa trên các tiến trình tự nhiên như là vật lý thống kê và sự tiến hóa sinh học hay dựa trên khung cảnh trí tuệ nhân tạo. B. Giffler và Thompson [28] là những người đầu tiên đề xuất tiếp cận gần đúng cho JSP. Trong một bài báo có tựa đề "*Algorithms for Solving Production Scheduling Problems",* các tác giả này đã đề cập tới vấn đề không cần thiết phải tìm kiếm một lịch biểu tối ưu trong toàn bộ không gian các lịch biểu có thể mà chỉ cần tìm kiếm trong một tập con các lịch biểu khả thi. Bài báo của họ còn quan trọng ở chỗ lần đầu tiên các phương pháp gần đúng được áp dụng dựa trên các luật ưu tiên và các lịch biểu tích cực.

Mặc dù các phương pháp gần đúng không bảo đảm chắc chắn tìm được các lời giải tối ưu thực sự, nhưng đổi lại chúng có thể đạt được các lời giải gần tối ưu trong một khoảng thời gian tính toán hợp lý cho các bài toán có độ phức tạp tính toán lớn. Vì vậy, các phương pháp gần đúng là thích hợp nhất để giải quyết các bài toán có độ phức tạp tính toán lớn với điều kiện là phải kết hợp nhuần nhuyễn giữa vận trù học và trí tuệ nhân tạo. Bốn tiếp cận gần đúng đã được nghiên cứu và áp dụng phổ biến nhất cho tới nay đó là: Các luật ưu tiên, các heuristic dựa trên nút cổ chai, trí tuệ nhân tạo và các phương pháp tìm kiếm cục bộ và meta-heuristic.

a. Các luật ưu tiên

Tiếp cận gần đúng được phát triển sớm nhất cho JSP dựa trên các luật ưu tiên. Các luật ưu tiên được đề xuất nhằm mục đích làm giảm các yêu cầu về thời gian tính toán. Trong tiếp cận này, tại mỗi bước, tất cả các thao tác sẵn sàng được lập lịch được gán một quyền ưu tiên nhất định, thao tác có quyền ưu tiên cao nhất sẽ được chọn để lập lịch.

Jackson [37], Giffler và Thompson [28] là những người đề xuất sớm nhất về luật ưu tiên. Trong các đề xuất sớm nhất này, nổi bật hơn cả là các luật ưu tiên của Giffler và Thompson (còn được gọi ngắn gọn là thuật toán GT). Cho tới nay, thuật toán GT vẫn được xem như là nền tảng cho các luật ưu tiên khác. Tầm quan trọng của nó xuất phát từ thực tế đó là nó sinh ra các lịch biểu tích cực.

Tiếp sau đó phải kể đến công trình nghiên cứu nổi tiếng và toàn diện nhất về các luật ưu tiên trong lập lịch của Panwalker và Iskander [56]. Trong công trình này, 113 luật ưu tiên được trình bày, xem xét lại và phân loại. Theo hướng nghiên cứu khác, một số đề xuất tập trung vào việc nghiên cứu kết hợp nhiều luật ưu tiên với nhau [73]; kết hợp các luật ưu tiên với các kỹ thuật tìm kiếm khác như: Kết hợp các luật ưu tiên với tìm kiếm cục bộ [32], kết hợp các luật ưu tiên với giải thuật đàn kiến [58], kết hợp các luật ưu tiên với tìm kiếm tabu và khai phá dữ liệu [8], kết hợp các luật ưu tiên với giải thuật di truyền [42],...

- **Phân tích, đánh giá**

Qua xem xét các công trình về luật ưu tiên đã được công bố trong những năm qua chúng ta có thể rút ra một số kết luận chung sau đây:

1. Các kết quả tìm kiếm đạt được rất nhanh khi dùng các luật ưu tiên một cách riêng lẻ nhưng chất lượng lời giải rất tồi, sự chênh lệch so với các kết quả tối ưu thực sự khá lớn. Các kết quả sẽ tốt hơn khi các luật được kết hợp với nhau, tuy nhiên thời gian tính toán yêu cầu nhiều hơn.

2. Khi kết hợp các luật ưu tiên với các kỹ thuật nhánh cận kết quả thực thi khá tốt nhưng thời gian tính toán so với chỉ dùng các luật ưu tiên lớn hơn gấp ba lần.

3. Các luật ưu tiên phù hợp khi một kỹ thuật giải khởi đầu cho JSP và khi chúng được kết hợp với các kỹ thuật tìm kiếm gần đúng khác (nhận xét này đã được luận án áp dụng cho thuật toán di truyền lai mới cho JSP trong chương 3).

b. Các heuristic dựa trên nút cổ chai

Các heuristic dựa trên nút cổ chai (Shifting Bottleneck - SB) là tiếp cận gần đúng xuất hiện muộn hơn các luật ưu tiên. Tiếp cận SB có thể khắc phục các điểm yếu của các luật ưu tiên đã phân tích ở trên và các tiếp cận chính xác yêu cầu thời gian tính toán quá cao. Adams và những người khác [3] là những người đầu tiên đề xuất phương pháp SB cho các bài toán lập lịch. Sau đó Applegate và Cook [6] đã kết hợp phương pháp SB với các kỹ thuật nhánh cận để tạo ra một phương pháp lai hiệu quả tìm được lời giải tối ưu cho bài toán *mt*10 là một thách thức tính toán thời bấy giờ. Gần đây hơn, nhiều công trình đã đưa ra các phương pháp nút cổ chai sửa đổi (Shifting Bottleneck Modification - SBM) để tăng tính hiệu quả cho các bài toán cỡ lớn. Chẳng hạn như các đề xuất của Zhi Huang [82], Karimi Gavareshki và Fazel Zarandi [43]. Ngoài ra, các nhà nghiên cứu còn tập trung vào việc kết hợp phương pháp SB với các kỹ thuật tìm kiếm khác để tạo ra các phương pháp mới hiệu quả hơn cho JSP như: Kết hợp SB với tìm kiếm giả luyện thép và thuật toán di truyền [61], kết hợp SB với thuật toán di truyền và lý thuyết tập mờ [62],...

- **Phân tích, đánh giá**

Các heuristic dựa trên nút cổ chai là một phương pháp mạnh được phân tích, thiết kế và thực hiện tốt. Nhược điểm cơ bản của phương pháp này là sự phức tạp khi cài đặt chương trình máy tính và vấn đề xác định cỡ của bài toán con. Một khó khăn khác của phương pháp dựa trên SB là vấn đề thiết đặt các tham số sao cho phù hợp để có thể thu được lời giải tốt nhất. Các phương

pháp SB sửa đổi và các phương pháp lai kết hợp SB với các phương pháp khác đã tạo ra các phương pháp mạnh tốt hơn cho JSP, đặc biệt là đối với các bài toán cỡ lớn. Đây cũng là xu hướng mà các nhà nghiên cứu đang tập trung vào nghiên cứu nhiều nhất hiện nay.

c. Các tiếp cận trí tuệ nhân tạo

Tiếp cận trí tuệ nhân tạo (Artificial Intelligence - AI) là một tiếp cận liên quan tới sự kết hợp giữa sinh học và các khả năng xử lý của máy tính. Có nhiều kỹ thuật trí tuệ nhân tạo khác nhau đã được phát triển cho bài toán lập lịch job shop. Trong phần này, luận án trình bày ba nhóm chính được nghiên cứu nhiều nhất cho JSP đó là: Phương pháp thỏa mãn ràng buộc, mạng nơ ron và các phương pháp trí tuệ nhân tạo còn lại.

c1. Phương pháp thỏa mãn ràng buộc

Phương pháp thỏa mãn ràng buộc nhằm mục đích làm giảm bớt kích cỡ của không gian tìm kiếm bằng cách áp dụng các ràng buộc hạn chế bậc của các biến được chọn và các giá trị được gán cho mỗi biến. Sau khi các biến đã được gán giá trị có thể xuất hiện các mâu thuẫn, tiến trình khắc phục các giá trị mâu thuẫn được gọi là việc kiểm tra tính nhất quán, và phương pháp tháo gỡ được gọi là phương pháp quay lui. Bài toán thỏa mãn ràng buộc được giải quyết khi vị trí các biến được xác định hoàn toàn và không vi phạm các ràng buộc của bài toán. Phương pháp này thuộc lĩnh vực trí tuệ nhân tạo, tuy nhiên việc áp dụng phương pháp này cho các bài toán lập lịch sử dụng mô hình cây tìm kiếm có liên hệ mật thiết với các kỹ thuật nhánh cận. Ban đầu phương pháp thỏa mãn ràng buộc được phát triển để giải quyết các bài lập lịch trong công nghiệp, sau đó nhiều công trình được đề xuất cho bài toán lập lịch job shop. Phương pháp thỏa mãn ràng buộc sớm nhất cho JSP là của Erschler và những người khác [21]. Các công trình tiếp theo phải kể đến: Các cách tiếp

cận thỏa mãn ràng buộc cho các bài toán lập lịch khó theo mức độ tăng dần của Fox và Sadeh [23], [64], phương pháp thỏa mãn ràng buộc kết hợp với các kỹ thuật nhánh cận để giải quyết một cách tối ưu các bài toán con được sinh ra của Pesch và Tetzlaff [57], Garrido và những người khác [27],…

- **Phân tích, đánh giá**

Đánh giá về phương pháp thỏa mãn ràng buộc trên cơ sở xem xét các công trình về các kỹ thuật thỏa mãn ràng buộc khác nhau có thể rút ra kết luận sau đây:

1. Các kết quả tìm kiếm tối ưu đạt được tương đối tốt và các phương pháp thỏa mãn ràng buộc mới được đề xuất gần đây cho kết quả tốt hơn. Tuy nhiên thời gian tính toán lại yêu cầu cao hơn.

2. Các phương pháp lai kết hợp phương pháp thỏa mãn ràng buộc với các tiếp cận chính xác cho kết quả tối ưu tốt hơn trong thực tế. Tuy nhiên, vì chúng tương tự như các kỹ thuật nhánh cận nên độ phức tạp thời gian tính toán khá cao.

3. Các phương pháp thỏa mãn ràng buộc cho JSP vẫn còn khá nhiều tồn tại. Một trong các lý do cho vấn đề này đó là các phương pháp được đề nghị này còn khá chung chung và có thể áp dụng được cho nhiều bài toán lập lịch khác nhau. Để áp dụng hiệu quả phương pháp này cho JSP, cần phải chú ý đến các thông tin đặc thù của bài toán hơn, khi đó các vùng hứa hẹn của không gian tìm kiếm mới có thể được xác định rõ ràng và thời gian tính toán mới được cải thiện.

c2. Mạng nơ ron

Mạng nơ ron được cấu trúc dựa trên cấu trúc bộ não của các thực thể sống đơn giản. Việc xử lý thông tin theo phương pháp này được thực hiện

thông qua một hệ thống các đơn vị xử lý song song có mối liên hệ với nhau. Mạng nơ ron là một phương pháp luận phổ biến và được ứng dụng nhiều trong thực tiễn. Có nhiều mô hình mạng nơ ron được áp dụng cho các bài toán lập lịch như: Mạng tìm kiếm Hop-field, mạng sửa lỗi Multi - Layer Perceptron, mạng xác suất Boltzmann machine, mạng cạnh tranh và mạng tự tổ chức. Tuy nhiên vì các ứng dụng mạng nơ ron cho JSP được thực thi chủ yếu trong hai mô hình đầu, nên luận án chỉ tập trung trình bày về hai loại mạng này. Bạn đọc có thể tham khảo sự phân tích và đánh giá về ứng dụng của mạng nơ ron trong lập lịch một cách đầy đủ hơn trong công trình của Wang và Brunn [74].

Mạng tìm kiếm Hop-field là mạng không tuyến tính kết hợp tự động để cực tiểu hóa năng lực của hệ thống. Trong các phương pháp dựa trên mạng Hop-field, mô hình toán học MIP thường được áp dụng để ánh xạ JSP vào mạng nơ ron.

Các mạng sửa lỗi Multi - Layer Perceptron được huấn luyện trên các mẫu lấy từ ánh xạ: $f: S \subset R^n \longrightarrow R^m$, từ một tập con S được giới hạn nào đó của không gian Ơclit n chiều tới không gian Ơclit m chiều. Khi một mẫu được áp dụng cho mạng, luật sửa lỗi điều chỉnh các trọng số sao cho phù hợp với ánh xạ ở trên. Các trọng số được điều chỉnh sao cho sự đáp ứng của mạng trên thực tế di chuyển tới gần sự đáp ứng mong muốn.

Trong những năm gần đây, nhiều đề xuất có xu hướng kết hợp mạng nơ ron với các phương pháp tìm kiếm khác nhằm tạo ra các phương pháp hiệu quả hơn cho JSP. Chẳng hạn như phương pháp lai kết hợp mạng nơ ron với thuật toán di truyền của Ye Li và Yan Chen [78] áp dụng cho JSP. Qua thử nghiệm phương pháp này đã được đánh giá là khá thành công trong tìm kiếm các lời giải tối ưu cho JSP.

- **Phân tích, đánh giá**

Các mạng tìm kiếm Hop-field có sử dụng mô hình toán học nên số các ràng buộc, các biến và mối quan hệ giữa các biến thường trở nên quá mức. Do vậy, mô hình này chỉ phù hợp với các bài toán cỡ nhỏ. Ngoài ra, mô hình này thường đòi hỏi phải được thiết kế đặc thù cho mỗi bài toán cần giải quyết. Hệ thống có thể gặp sự cố nếu như áp dụng mô hình không được thiết kế cho nó.

Đối với mô hình mạng sửa lỗi thường có sự đòi hỏi quá mức về các nơ ron và sự huấn luyện từ các dữ liệu không tối ưu được yêu cầu bởi các chuyên gia, các thông tin hiện tại hay các luật ưu tiên. Phần lớn các mô hình kết hợp mạng sửa lỗi với các phương pháp khác, các kết quả tìm kiếm tối ưu thường do các kỹ thuật khác mang lại.

Osman và Kelly [55] đã kết luận rằng các mạng nơ ron không thể cạnh tranh được với các phương pháp khác và không phải là giải pháp tốt cho JSP.

c3. Các tiếp cận AI còn lại

Các tiếp cận AI còn lại bao gồm:

1. Mô hình phân bổ ngẫu nhiên song song của Lo và Hsu [47], tiếp cận này có nhiều đặc điểm giống với mạng nơ ron Hop-field.

2. Phương pháp tiềm năng cân bằng phỏng theo vật lí động lực học và thống kê của Yokoi và những người khác [80], ở đó tiềm năng vật lí của mỗi công việc là đối tượng quan sát như là năng lượng tương tác giữa các hoạt động ở trên các máy.

3. Phương pháp tối ưu hệ kiến (Ant System - AS) cho JSP của Colorni và những người khác [15]. AS là một kỹ thuật tối ưu ngẫu nhiên dựa trên quần thể được phát triển bởi Denebourg, Pasteels và Verhaeghe [20]. Nó mô phỏng theo hành vi của các đàn kiến tìm đường đi ngắn nhất tới nguồn thức

ăn của nó. Thủ tục đàn kiến đánh dấu vị trí ở trên các nút của một đồ thị không liên thông và theo một nguyên lý Monte Carlo kết hợp với tìm kiếm tham lam để tìm ra sự quyết định nút liền kề nào sẽ di chuyển tới.

Trong 3 tiếp cận kể trên, phương pháp AS được nghiên cứu và áp dụng nhiều nhất cho JSP. Đã có nhiều đề xuất dựa trên AS như: Phương pháp tối ưu đàn kiến dựa trên tri thức của Lining Xing và những người khác [46], phương pháp tối ưu đàn kiến theo hai giai đoạn của Omar Castrillon và những người khác [54]. Một số đề xuất kết hợp AS với các kỹ thuật tìm kiếm khác: Kết hợp các luật ưu tiên với thuật toán đàn kiến được đề xuất bởi Ramezanali Mahdavinejad [58], kết hợp thuật toán di truyền với thuật toán đàn kiến được đề xuất bởi Surekha và Sumathi [67],...

- **Phân tích, đánh giá**

Qua xem xét các công trình liên quan đến 3 kỹ thuật AI nêu trên, các kết quả thu được qua thử nghiệm của các phương pháp được đề xuất cho thấy rằng các phương pháp này nói chung thực thi không tốt trong khi nỗ lực tính toán được yêu cầu lại cao. Vì vậy, chúng không phải là giải pháp tốt cho JSP.

d. Các phương pháp cục bộ và Meta-heuristics

Để tìm lời giải tối ưu cho một bài toán tối ưu tổ hợp, thông thường cần phải xác định một cấu hình bao gồm: Một tập hữu hạn các lời giải S, một hàm định giá cần được tối ưu và một cơ chế sinh lời giải mới. Đó là một cấu hình đơn giản nhất được dùng để tạo ra một sự chuyển đổi từ một trạng thái này sang một trạng thái khác của không gian trạng thái các lời giải. Các phương pháp dựa trên nguyên lý tổng quát trên được xem như là tìm kiếm cục bộ hay tìm kiếm vùng lân cận. Trong kỹ thuật này, cơ chế sinh phác thảo một vùng lân cận cho mỗi cấu hình. Một vùng lân cận $N(x)$ là một hàm xác định một sự chuyển đổi từ một lời giải x sang một lời giải x' khác bằng cách gây ra một sự

thay đổi nhỏ. Mỗi lời giải $x' \in N(x)$ có được từ lời giải x bằng một phép biến đổi đơn giản được xác định trước và được gọi là một sự di chuyển, x được nói là di chuyển tới x'. Trong hầu hết các kỹ thuật tìm kiếm vùng lân cận, người ta công nhận một cấu trúc đối xứng đó là: x' là một lân cận của x khi và chỉ khi x là lân cận của x'. Mục tiêu của chiến lược tìm kiếm này là thông qua một chuỗi các phép di chuyển hướng tới việc tìm kiếm một lời giải tối ưu. Việc chọn một lân cận trong một phép di chuyển phụ thuộc và tiêu chuẩn chọn, một số thủ tục chọn phổ biến bao gồm:

1. Chọn lân cận có hàm đánh giá tốt hơn được tìm thấy đầu tiên. Thủ tục này thường được áp dụng trong các thuật toán ngưỡng.

2. Chọn lân cận có hàm đánh giá tốt nhất trong toàn bộ vùng lân cận. Thủ tục này thường được áp dụng trong các heuristic nút cổ chai.

3. Chọn lân cận có hàm đánh giá tốt nhất của một mẫu các lân cận. Thủ tục này thường được áp dụng trong các kỹ thuật tìm kiếm tabu.

Một số các phương pháp tìm kiếm cục bộ được đề xuất phỏng theo các tiến trình trong tự nhiên như là vật lý thống kê, sự tiến hóa của sinh học,... Chúng thường hoạt động như là một tiến trình chủ được lặp đi lặp lại nhiều lần, vì thế chúng còn được gọi là meta-heuristics. Hầu hết các kỹ thuật meta-heuristics được phát triển trong những năm gần đây theo hướng các phương pháp tìm kiếm gần đúng để giải các bài toán tối ưu phức tạp. Trong đó có một số phương pháp meta-heuristics được phát triển cho JSP nổi bật sau đây: Thuật toán cải thiện lặp, thuật toán giả luyện thép, thuật toán chấp nhận ngưỡng, thuật toán tìm kiếm tabu và thuật toán di truyền.

d1. Thuật toán cải thiện lặp (Iterative Improvement - IM)

Thuật toán cải thiện lặp có thể được mô tả vắn tắt như sau:

Từ một lịch biểu được sinh ngẫu nhiên ban đầu, thuật toán điều khiển quá trình tìm kiếm để thu được một lời giải tối ưu là lời giải tốt nhất trong vùng lân cận của nó. IM là lớp đơn giản nhất trong các kỹ thuật tìm kiếm cục bộ, thủ tục IM được đặc tả như dưới đây:

Chọn một lời giải ban đầu $s \in S$;
REPEAT
    Sinh lời giải $s' \in N(s)$;
    IF $c(s') < c(s)$ *THEN* $s := s'$;
UNTIL $c(s') \geq c(s)$

Thuật toán IM sẽ được lặp đi lặp lại cho tới khi gặp một lời giải s^* nào đó. Nói chung, s^* chỉ là một lời giải tối ưu cục bộ đối với vùng lân cận N, vì vậy rất có thể không phải là lời giải tối ưu toàn cục. Để khắc phục hạn chế này, Aarts và những người khác [2] đã đề xuất một thuật toán IM đa khởi tạo. Thuật toán khởi đầu từ một lời giải được sinh ngẫu nhiên, sau đó thuật toán IM tìm kiếm lời giải tốt nhất trong vùng lân cận của nó. Thuật toán lại bắt đầu từ một lời giải sinh ngẫu nhiên khác, tiến trình này được lặp đi lặp lại cho tới khi tiêu chuẩn dừng thỏa mãn và lời giải tốt nhất trong các lần lặp được chọn.

- **Phân tích, đánh giá**

Các kết quả thử nghiệm từ các công trình được đề xuất về thuật toán IM, ngay cả trong trường hợp đã được cải tiến cho thấy thường chỉ đạt được tối ưu cục bộ. Lý do cho vấn đề này là vì IM chỉ thực hiện tìm kiếm trong một phạm vi giới hạn (một vùng lân cận) của không gian các lời giải. Vì vậy, có thể rút ra kết luận IM không phải là giải pháp tốt cho JSP.

d2. Thuật toán giả luyện thép (Simulated Annealing - SA)

SA là một trong những phương pháp tìm kiếm cục bộ điển hình, phương pháp này phỏng theo một tiến trình vật lý trong luyện kim. Thủ tục SA có thể được đặc tả như dưới đây:

i := 0;

Khởi tạo một lời giải s ∈ S;

best := c(s);

s := s;*

REPEAT

Sinh ngẫu nhiên một lời giải s' ∈ N(s);

IF random[0, 1] < min{1, exp(- $\frac{c(s')-c(s)}{c_i}$ *)} THEN s := s';*

IF c(s') < best THEN

BEGIN

s := s';*

best := c(s');

END;

c_{i+1} *:= g(*c_i*);*

i := i + 1;

UNTIL tiêu chuẩn dừng thỏa mãn

Trong bước thứ *i*, *s*' được chấp nhận với xác suất được xác định bởi min {1, exp(- $\frac{c(s')-c(s)}{c_i}$)}, ở đây c_i là một chuỗi các tham số điều khiển dương với lim c_i = 0 khi $i \longrightarrow \infty$.

Điều này có thể được giải thích rõ hơn như sau: nếu $c(s') \leq c(s)$, thì *s* được thay thế bởi *s*'với xác suất là 1. Nếu $c(s') > c(s)$, thì *s* được thay thế bởi *s*' với một xác suất nào đó, xác suất này sẽ giảm dần theo sự tăng dần của biến *i*. Để hiểu chi tiết hơn về vấn đề này, có thể tham khảo bài viết của Van Laarhoven và những người khác [72]. Một trong những đóng góp quan trọng

trong việc phát triển phương pháp SA cho JSP của Van Laarhoven và những người khác [71] là xác định vùng lân cận, trên cơ sở vùng lân cận này họ đã xây dựng một thuật toán SA mạnh cho JSP, thuật toán này có ưu điểm là đơn giản và dễ thực hiện. Trong các đề xuất sau này, nổi bật là đề xuất của Yamada và Nakano [76], đề xuất này đã cải tiến thuật toán SA bằng việc kết hợp với thuật toán GT của Giffler và Thompson [28] để sinh các lịch biểu tích cực.

Trong những năm gần đây, các nhà nghiên cứu có xu hướng phát triển các thuật toán SA lai với các kỹ thuật tìm kiếm khác nhằm tạo ra các giải pháp mạnh cho JSP như: Kết hợp SA với tìm kiếm tabu [81], kết hợp SA với thuật toán memetic dựa trên sự chọn lọc vô tính [39], kết hợp SA với thuật toán di truyền, thuật toán dựa trên nút cổ chai và lý thuyết mờ [63],...

- **Phân tích, đánh giá**

Các kết quả thử nghiệm cho thấy các hạn chế chính của SA đó là:

1. Thời gian tính toán được yêu cầu quá cao để có thể đạt được các lời giải tốt (có trường hợp lên tới 44 giờ tính toán với hơn 375 triệu lần lặp).

2. Kết quả thực thi phụ thuộc rất nhiều vào việc lựa chọn các tham số cho thuật toán, vì vậy các tham số phải được chọn rất cẩn thận, điều này là rất khó.

Sở dĩ có các hạn chế trên bởi vì thuật toán SA phải trải qua rất nhiều lần lặp mới tới được lời giải tốt. Ngoài ra, các phương pháp SA được đề xuất còn khá chung chung, chưa tập trung vào thông tin đặc thù của bài toán JSP.

Chính vì các hạn chế trên mà các giải pháp SA sau này đã tập trung vào cải tiến nhằm tăng tốc độ thực thi của SA. Chẳng hạn như: Cải tiến cách tính xác suất chấp nhận bằng một bảng tra cứu của Johnson và những người khác

[40], đề xuất tăng tốc SA bằng cách cho phép thực hiện bước nhẩy dài của Szu và Hartely [68],... và gần đây hơn là các đề xuất kết hợp SA với các phương pháp mạnh khác để cải thiện tốc độ và chất lượng của lời giải như đã nêu ở trên. Các kết quả thử nghiệm đã cho thấy ưu điểm của các phương pháp SA lai và các phương pháp SA lai hiện vẫn đang là hướng nghiên cứu để tạo ra các giải pháp tốt hơn cho JSP.

d3. Thuật toán chấp nhận ngưỡng (Threshold Acceptance - TA)

Một biến thể của thuật toán giả luyện thép là thuật toán chấp nhận ngưỡng (TA). TA khác SA ở chỗ chấp nhận *s*': Trong thuật toán TA, *s*' được chấp nhận nếu *c*(*s*') - *c*(*s*) nhỏ hơn một ngưỡng *t* nào đó, *t* là một tham số điều khiển dương giảm dần. Thủ tục chấp nhận ngưỡng được đặc tả như sau:

i := 0;
Khởi tạo một lời giải s ∈ S;
best := c(s);
s := s;*
REPEAT
    Sinh ngẫu nhiên một lời giải s' ∈ N(s);
    IF c(s') - c(s) < t_i THEN s := s';
    IF c(s') < best THEN
        BEGIN
            s := s';*
            best := c(s');
        END;
    t_{i+1} := g(t_i);
    i := i + 1;
UNTIL tiêu chuẩn dừng thỏa mãn

g là một hàm không âm và *g*(*t*) < *t* với mọi *t*.

- **Phân tích, đánh giá**

Phương pháp TA có ưu điểm hơn phương pháp IM là có thể tránh được tối ưu cục bộ. Tuy nhiên, TA cũng có những hạn chế tương tự như phương pháp SA là có thể quay trở lại các lời giải đã thăm và làm tăng thời gian tính toán. Chính vì vậy, phương pháp TA đơn cũng không phải là giải pháp tốt cho JSP, nó nên được dùng kết hợp với một số phương pháp khác.

d4. Thuật toán tìm kiếm tabu (Tabu Search - TS)

Để tránh trường hợp quay trở lại những lời giải đã thăm của các phương pháp SA và TA, phương pháp TS lưu trữ các lời giải đã thăm trong một danh sách được gọi là tabu. Tập các ứng cử viên cho lời giải của bước lặp tiếp theo không chấp nhận các lời giải chứa trong danh sách tabu. Tuy nhiên, việc lưu trữ tất cả các lời giải đã thăm trong danh sách tabu và việc kiểm tra xem một lời giải ứng cử viên có thuộc danh sách này hay không sẽ gây tốn kém tài nguyên bộ nhớ và tăng thời gian thực thi của thuật toán. Để phần nào hạn chế điều này, trong thực tế người ta chỉ lưu trữ một số nhất định các lời giải mới được thăm gần đây nhất trong danh sách tabu. Tùy theo mục đích tìm kiếm, số lời giải được lưu trữ trong danh sách tabu được giới hạn bởi một số *t* nào đó. Khi số lời giải lưu trữ trong danh sách tabu vượt quá *t* thì lời giải được lưu trữ sớm nhất trong danh sách sẽ bị loại bỏ để nhường chỗ cho lời giải mới được thăm. Trong phương pháp TS, một tiêu chuẩn được gọi là "*khát vọng*" có thể được đưa vào. Khi một lời giải *s* di chuyển tới một lời giải *s'* thuộc danh sách tabu, nó sẽ được chấp nhận nếu thỏa mãn tiêu chuẩn khát vọng.

Thủ tục TS có thể được đặc tả như sau:

Khởi tạo lời giải ban đầu $s \in S$*;*

best := c(s);

s := s;*

Tabu list := ∅;

REPEAT

Cand(s) := {s' ∈ N(s) / s' không thuộc danh sách tabu hay s' thuộc danh sách tabu nhưng thỏa mãn tiêu chuẩn khát vọng}

Sinh ngẫu nhiên một lời giải $\bar{s} \in Cand(s)$;

Cập nhật danh sách tabu;

$s := \bar{s}$

IF c(s) < best THEN

BEGIN

s := s;*

best := c(s);

END;

UNTIL tiêu chuẩn dừng thỏa mãn

Glover [30] là người đầu tiên đề xuất phương pháp TS, tuy nhiên Laguna và những người khác [45] mới là những người đầu tiên sử dụng tiếp cận TS cho các bài toán lập lịch. Sau đó phương pháp TS đã được cải tiến bởi nhiều nhà nghiên cứu khác nhằm cải thiện sự thực thi của nó cho JSP. Gần đây, các nhà nghiên cứu đưa ra các đề xuất kết hợp TS với các kỹ thuật tìm kiếm khác để tạo ra các phương pháp lai hiệu quả cho JSP như: Kết hợp TS với SA của Zhang và những người khác [81], kết hợp TS với thuật toán di truyền để tạo ra một phương pháp lai được gọi là "*Genetic Local Search* - GLS" của Moraglio và những người khác [51],...

- **Phân tích, đánh giá**

Từ các công trình đã được công bố và các kết quả thử nghiệm của chúng trên các bài toán chuẩn, các kết quả thử nghiệm thu được đã cho thấy

về mặt tổng thể các phương pháp dựa trên TS cho kết quả tốt nhất trong các phương pháp tìm kiếm cục bộ đã trình bày ở trên. Các phương pháp TS không những cho các lời giải tốt hơn mà thời gian tính toán lại không yêu cầu quá cao. Các phương được đưa ra gần đây thường là các phương lai kết hợp TS và các kỹ thuật tìm kiếm khác [35], [51],... Các phương pháp lai chiếm ưu thế hơn, đặc biệt là với các bài toán có kích cỡ lớn và có độ phức tạp như JSP.

Hạn chế đáng kể nhất của các kỹ thuật tìm kiếm dựa trên TS là yêu cầu các tham số phải được chọn rất cẩn thận và phải được điều chỉnh sao cho phù hợp với đặc thù của mỗi bài toán. Các quyết định như thế thường là rất khó thực hiện.

d5. Các thuật toán di truyền (Genetic Algorithms - GA)

Các thuật toán di truyền phỏng theo các quá trình sinh học để tối ưu hóa các hàm mục tiêu. John Holland [34] và các cộng sự của ông tại trường đại học Michigan, Mỹ là những người đầu tiên đề xuất phương pháp GA. Sau đó, GA được rất nhiều người nghiên cứu, trong đó phải kể đến người học trò xuất sắc của Holland là D. E. Goldberg. Goldberg đã có công đóng góp rất nhiều trong việc phát triển lý thuyết và ứng dụng thuật toán di truyền. Ông đã thành công trong việc áp dụng lý thuyết GA để phát triển một hệ thống điều khiển sự truyền dẫn khí gas trong một thành phố, ông cũng rất nổi tiếng với cuốn sách mang tựa đề "*Genetic algorithms in search, optimization and machine learning*" [31].

Tuy xuất hiện muộn hơn các phương pháp khác nhưng GA được phát triển khá nhanh cả về phương diện lý thuyết lẫn ứng dụng trong thực tiễn. Các chuyên gia nghiên cứu về GA hiện nay đã trải rộng hầu hết các quốc gia trên thế giới và đặc biệt đông đảo ở một số quốc gia như Mỹ, Trung Quốc, Nhật Bản, Hàn Quốc,... Lý thuyết GA đã được ứng dụng thành công trong rất nhiều

lĩnh vực khác nhau trong đó có lĩnh vực khoa học máy tính bao gồm: Học máy, lý thuyết điều khiển, tối ưu tổ hợp,... Đặc trưng chung nhất của GA là: Chiến lược tìm kiếm dựa trên quần thể cùng với ba toán tử: Chọn lọc, đột biến và trao đổi chéo. Những người đi tiên phong áp dụng GA cho JSP phải kể đến Nakano và Yamada [53], họ cũng là những người đã có nhiều đóng góp trong việc phát triển và cải tiến các phương pháp dựa trên GA cho JSP.

- **Phân tích, đánh giá**

Từ một số công trình áp dụng GA cho JSP, các kết quả thử nghiệm cho thấy rằng GA không phù hợp cho việc điều chỉnh các lời giải khi rất gần với lời giải tối ưu vì toán tử trao đổi chéo thường phá vỡ sự điều chỉnh này. Để khắc phục nhược điểm đó, các nhà nghiên cứu đã tìm kiếm các giải pháp lai kết hợp GA với các phương pháp tìm kiếm khác. Người đi tiên phong trong việc tìm kiếm các phương pháp lai là Ulder và những người khác [69]. Họ đã đề xuất một phương pháp lai kết hợp GA với tìm kiếm cục bộ cho ra một giải pháp có tên "*Genetic local search* - GLS". Trong phương pháp này, phép trao đổi chéo được thực hiện trên hai cá thể để sinh ra một lời giải mới, lời giải mới này được dùng như là lời giải khởi đầu cho việc tìm kiếm cục bộ tiếp theo. Phương pháp GLS đã kết hợp kỹ thuật tìm kiếm cục bộ để cải thiện lời giải và toán tử trao đổi chéo của GA để cung cấp sự trao đổi thông tin giữa các cá thể. Trong những năm gần đây, nhiều phương pháp lai GA với các kỹ thuật tìm kiếm khác đã được đề xuất như: Kết hợp GA và lý thuyết mờ của Deming Lei [19], kết hợp GA với các luật ưu tiên của Kamrul Hasan [42], kết hợp GA với giả luyện thép [61], kết hợp GA với tìm kiếm dựa trên nút cổ chai và lý thuyết tập mờ [62], kết hợp GA với mạng nơ ron của Ye Li và Yan Chen [79],... Các kết quả thử nghiệm trên các bài toán chuẩn từ các công trình áp dụng GA cho JSP đã được công bố cho thấy các kỹ thuật lai chiếm ưu thế hơn các kỹ thuật sử dụng GA đơn thuần, đổi lại chúng lại yêu cầu thời gian

tính toán nhiều hơn. Tuy nhiên, đối với các bài toán cỡ lớn đa số các phương pháp đã đề xuất vẫn chưa tìm được lời giải tối ưu thực sự trong khoảng thời gian tính toán chấp nhận được. Trong tương lai, các giải pháp dựa trên GA vẫn đang là đối tượng nghiên cứu để cho ra các phương pháp mạnh hiệu quả hơn khi giải quyết các bài toán cỡ lớn.

1.3.4. Tổng kết đánh giá chung về các tiếp cận cho JSP

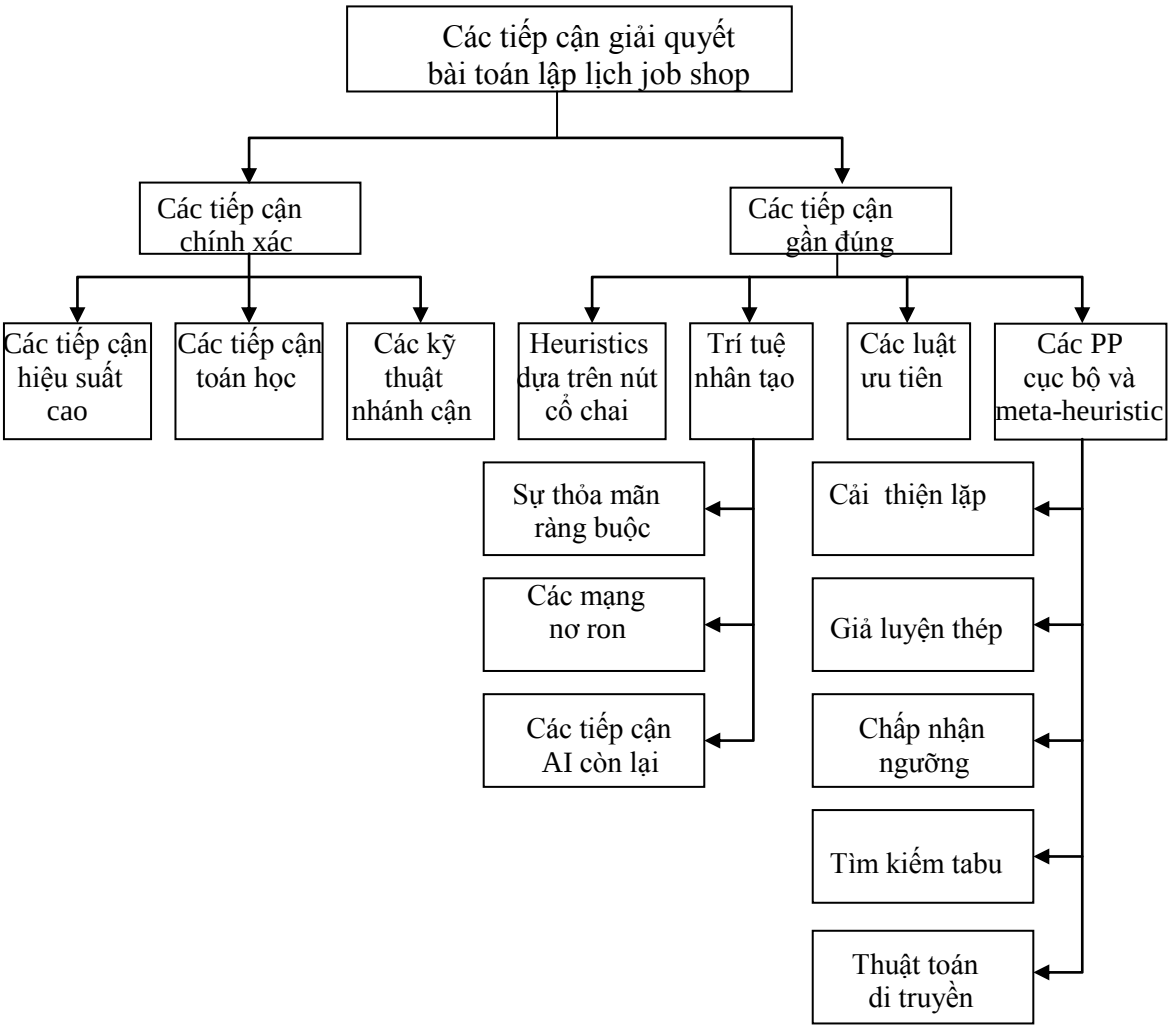
Sơ đồ tổng kết các tiếp cận chủ yếu đã được áp dụng cho bài toán lập lịch job shop được trình bày trong hình 1.7.

Trong các tiếp cận chính xác cho JSP, các kỹ thuật nhánh cận chiếm ưu thế hơn cả. Tuy nhiên, tiếp cận này chỉ có thể áp dụng cho các bài toán cỡ nhỏ vì chúng thường gây ra tràn bộ nhớ do sự phát triển quá nhanh của cây tìm kiếm. Vì vậy, khi kết hợp kỹ thuật BB với các kỹ thuật khác cũng cần phải hết sức thận trọng để hạn chế nhược điểm này.

Trong nhóm các tiếp cận gần đúng: Các luật ưu tiên, phương pháp cải thiện lặp và phương pháp chấp nhận ngưỡng cho kết quả rất tồi và được kết luận là không phải các giải pháp tốt cho JSP. Trong các tiếp cận này, các luật ưu tiên thường được dùng để hỗ trợ các tiếp cận tìm kiếm chủ công khác như là các tiếp cận tìm kiếm cục bộ và meta-heuristic.

Nếu không được kết hợp với các kỹ thuật tìm kiếm cục bộ khác, các heuristic dựa trên nút cổ chai được kết luận là không tốt hơn các luật ưu tiên nhanh trong khi đòi hỏi thời gian tính toán lại nhiều hơn.

Các tiếp cận trí tuệ nhân tạo được xem là còn rất nhiều tiềm năng. Tuy nhiên, chúng cần phải được nghiên cứu thêm, đặc biệt là vấn đề liên quan tới việc điều chỉnh các giá trị tham số và việc sử dụng các thông tin đặc thù của bài toán. Mạng nơ ron và hệ kiến vẫn cần phải có các nghiên cứu và thử nghiệm thêm để có thể đánh giá đúng bản chất của các tiếp cận này.



Hình 1.7 - Các tiếp cận chủ yếu giải quyết JSP

Các tiếp cận tìm kiếm dựa trên SA chưa phải là các kỹ thuật tìm kiếm mạnh cho JSP, tuy nhiên chúng có thể là đối thủ để cạnh tranh với các kỹ thuật mạnh khác. Đặc biệt, khi SA được kết hợp một cách hợp lý với các kỹ thuật tìm kiếm khác đã cho các giải pháp hiệu quả đối với các JSP cỡ lớn. Tuy nhiên, để đạt được các lời giải tối ưu các tiếp cận lai này lại thường đòi hỏi thời gian tính toán quá lớn.

Các tiếp cận dựa trên TS cho kết quả tốt nhất trong các kỹ thuật tìm kiếm cục bộ cả về chất lượng lời giải lẫn thời gian tính toán. Các kỹ thuật lai

TS và các kỹ thuật tìm kiếm khác đều cho các lời giải chất lượng cao và thời gian tính toán hợp lý, đặc biệt là với các bài toán có kích cỡ lớn và có độ phức tạp như JSP.

Các kỹ thuật tìm kiếm dựa trên GA cho JSP nếu chỉ áp dụng đơn thuần thì kết quả tìm kiếm kém hơn so với các kỹ thuật tìm kiếm cục bộ khác. Tuy nhiên, nếu GA được kết hợp với các kỹ thuật tìm kiếm khác sẽ tạo ra các phương pháp lai khá hiệu quả cho JSP. Trong các công trình được công bố trong những năm gần đây về các giải pháp cho JSP, có lẽ số các công trình có liên quan tới áp dụng GA là nhiều nhất.

Cuối cùng có thể đưa ra kết luận chung rằng các kết quả tốt nhất thu được cả về thời gian chạy máy lẫn chất lượng lời giải thuộc về các tiếp cận gần đúng lai như là SB lai, SA lai, TS lai và GA lai. Các phương pháp lai này cần phải hết sức mềm dẻo trong việc kết hợp các kỹ thuật tìm kiếm trên cơ sở các đặc thù riêng của bài toán job shop. Tuy nhiên, cho tới nay phương pháp luận cho việc kết hợp các kỹ thuật tìm kiếm khác nhau để tạo ra một giải pháp mạnh cho JSP vẫn còn là vấn đề cần phải nghiên cứu.

1.3.5. Một số tồn tại và các đề xuất

- **Một số tồn tại**

Qua phân tích, đánh giá các tiếp cận đã đề xuất cho JSP, có một số vấn đề cần phải được thảo luận chi tiết và đầy đủ hơn sau đây:

1. Vấn đề thứ nhất liên quan tới heuristic, hiện nay việc chọn một heuristic cho một bài toán cụ thể cần giải quyết là chưa rõ ràng và nó phụ thuộc vào nhiều nhân tố khác nhau. Cho nên việc đánh giá kết quả của các giải pháp khác nhau đòi hỏi phải chính xác, đặc biệt các kết quả có liên quan tới heuristic. Cho tới nay có rất ít các chỉ dẫn cụ thể về vấn đề này được đưa ra và vì vậy không có các thủ tục tiêu chuẩn. Chính vì thế trong rất nhiều

trường hợp trên cơ sở các kết quả thử nghiệm thu được không thể giải thích rõ ràng cái gì nên làm và cái gì không. Nhiều nhà phân tích cho rằng vấn đề thiết kế thử nghiệm còn nhiều bất cập, các thiết kế thử nghiệm cần phải nghiêm ngặt hơn, các bài toán chuẩn cần phải nhiều hơn và đa dạng hơn. Các thiết kế thử nghiệm cũng cần phải chỉ rõ những cái gì cần được đo và đo như thế nào.

2. Một vấn đề thứ hai cần phải được làm rõ đó là xác định các thuộc tính hội tụ của các tiếp cận gần đúng, khả năng đảm bảo tìm ra lời giải tối ưu của các giải pháp gần đúng mới cho JSP.

3. Vấn đề thứ ba đó là chưa có một phương pháp hình thức được đề xuất cho việc kết hợp hiệu quả các kỹ thuật tìm kiếm với nhau. Các nghiên cứu nên tập trung vào việc đưa ra cách thức kết hợp như thế nào để có một tiếp cận mới mạnh hơn.

4. Vấn đề thứ tư là sự phức tạp của các bài toán job shop cỡ lớn với không gian lời giải quá lớn của nó. Do vậy, nên tập trung vào các kỹ thuật tìm kiếm song song trên nhiều vùng lân cận của không gian lời giải. Theo hướng này phương pháp mạng nơ ron nếu được khai thác đúng tiềm năng thực sự của nó và có sự kết hợp hợp lý với các chiến lược tìm kiếm khác sẽ có thể giải quyết được vấn đề này.

- **Một số đề xuất**

Sau khi phân tích, đánh giá các giải pháp đã được áp dụng cho JSP, kết hợp với các kết luận của các công trình được đề xuất gần đây, luận án có một số đề xuất sau đây:

1. Các phương pháp mới cho JSP nên tập trung vào các kỹ thuật gần đúng để tránh thời gian tăng theo hàm số mũ khi cỡ bài toán tăng theo tuyến tính và nên là các giải pháp lai pha trộn một số kỹ thuật tìm kiếm khác nhau.

2. Các phương pháp mới này nên tích hợp một cách hợp lý các ưu điểm nổi trội của mỗi phương pháp thành phần sao cho phù hợp với đặc thù của mỗi bài toán cần giải quyết.

3. Các phương pháp mới nên tập trung vào giải quyết các vấn đề còn chưa được làm rõ đã nêu ở trên, đặc biệt là khai thác tiềm năng của một số tiếp cận giàu tiềm năng mà chưa được khai thác.

4. Trong tìm kiếm cục bộ, nên sử dụng khối tới hạn trong các cấu trúc vùng lân cận để tạo ra các di chuyển một cách hiệu quả nhất. Trong giai đoạn đầu của quá trình tìm kiếm nên áp dụng phương pháp dựa trên nút cổ chai để đạt được các lời giải chất lượng cao một cách nhanh chóng.

5. Các phương pháp mới nên áp dụng các luật sinh lịch biểu tích cực để hạn chế không gian tìm kiếm và dẫn dắt tới các lời giải tốt hơn.

CHƯƠNG 2. HAI BÀI TOÁN CON CỦA BÀI TOÁN LẬP LỊCH JOB SHOP

Trong thực tiễn, chúng ta thường gặp trường hợp các bài toán cần giải quyết chỉ thỏa mãn một số ràng buộc của bài toán lập lịch job shop. Đối với các bài toán con này, cách giải quyết đơn giản hơn bài toán lập lịch job shop rất nhiều. Trong chương này, luận án trình bày hai bài toán con của bài toán lập lịch job shop thường gặp trong thực tiễn sản xuất và đề xuất một thuật toán di truyền với mã hóa tự nhiên cho chúng.

2.1. Bài toán lập lịch flow shop hoán vị

Chúng ta xét một ví dụ trong thực tiễn sản xuất trước khi mô tả hình thức bài toán flow shop hoán vị:

Giả sử cần gia công n chi tiết C_1, C_2,…, C_n trên m máy M_1, M_2,…, M_m, mỗi chi tiết đều phải được gia công trên tất cả các máy theo thứ tự 1, 2,…, m. Vấn đề đặt ra là phải tìm ra một trình tự gia công các chi tiết trên các máy sao cho thời gian hoàn thành ngắn nhất. Đây là một ví dụ thực tiễn cho bài toán lập lịch flow shop hoán vị.

2.1.1. Mô tả bài toán

Bài toán lập lịch flow shop hoán vị (Permutation Flow shop Scheduling Problem - PFSP) được mô tả một cách hình thức như sau:

Cho n công việc (J_1, J_2, ..., J_n) được xử lý trên m máy (M_1, M_2, ..., M_m) và có các đặc trưng sau đây:

1. Mỗi công việc J_i (i = 1, ..., n) có m thao tác, thao tác thứ j phải được xử lý ở trên máy M_j (j = 1, ..., m). Như vậy, một công việc chỉ có thể bắt đầu được xử lý ở trên máy M_j nếu nó được hoàn thành việc xử lý ở trên máy M_{j-1}

và máy M_j đang rỗi. Tất cả các công việc phải được xử lý một cách liên tục từ khi bắt đầu cho tới khi kết thúc, không có khoảng thời gian dừng khi chuyển từ máy này sang máy khác.

2. Trình tự xử lý các công việc ở trên tất cả các máy là như nhau. Tức là, nếu một công việc có thứ tự xử lí thứ *i* ở trên máy M_1 thì công việc đó cũng có thứ tự xử lý thứ *i* ở trên các máy còn lại.

3. Thao tác của công việc J_i được xử lý ở trên máy M_j được ký hiệu là O_{ij} và có thời gian xử lý cho trước là p_{ij}.

4. Khoảng thời gian kể từ khi bắt đầu xử lý các công việc cho tới khi hoàn thành việc xử lý tất cả các công việc được gọi là makespan của bài toán và được ký hiệu là C_{max}.

Việc giải quyết PFSP là xác định một lịch biểu (một thứ tự xử lý các công việc ở trên mỗi máy) sao cho makespan là nhỏ nhất.

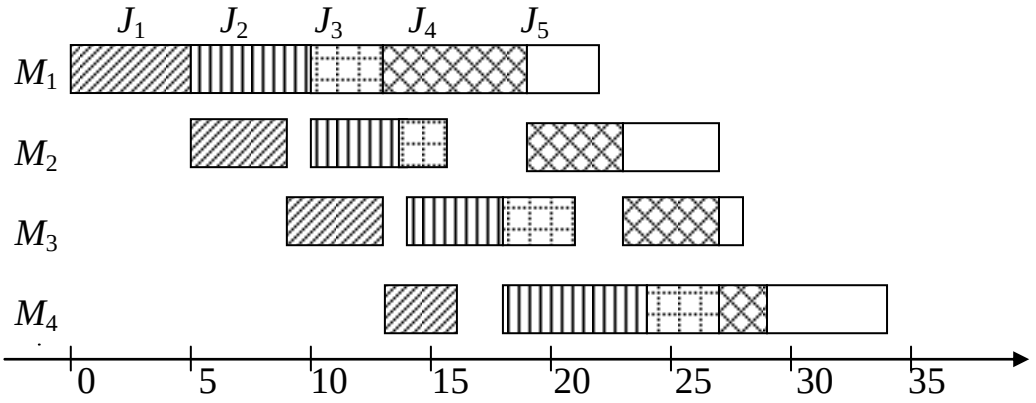
- **Một ví dụ minh họa**

Ví dụ, PFSP 5 công việc 4 máy được cho trong bảng 2.1. Dữ liệu vào bao gồm tuần tự công nghệ của các công việc và thời gian xử lí mỗi công việc ở trên mỗi máy.

Bảng 2.1 - PFSP 5 công việc 4 máy

	J_1	J_2	J_3	J_4	J_5
M_1	5	5	3	6	3
M_2	4	4	2	4	4
M_3	4	4	3	4	1
M_4	3	6	3	2	5

Để biểu diễn trực quan một lời giải (lịch biểu), người ta thường dùng biểu đồ Grant. Giả sử rằng các công việc được lập lịch theo thứ tự xử lý: J_1, J_2, J_3, J_4, J_5, biểu đồ Grant minh họa lịch biểu hoán vị tương ứng như được trình bày trong hình 2.1 với makespan = 34.



Hình 2.1 - Biểu đồ Grant biểu diễn một lời giải của PFSP 5 công việc 4 máy

2.1.2. Cách tính thời gian hoàn thành trong một lịch biểu hoán vị

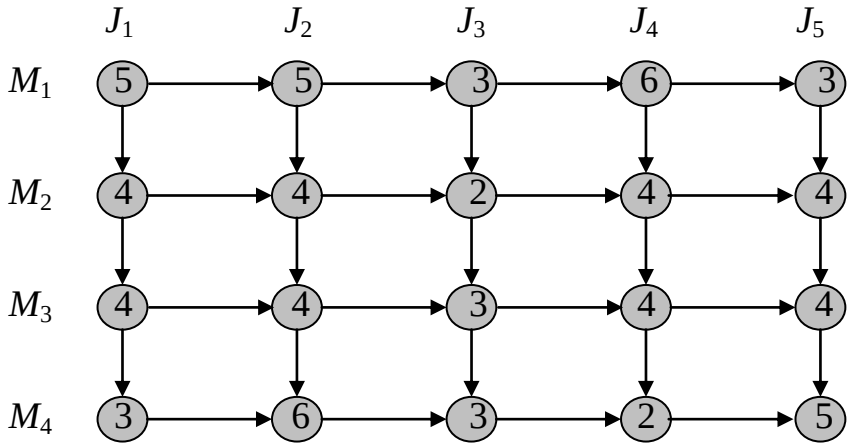
Gọi j_1, j_2, j_3, j_4, j_5 là một hoán vị của J_1, J_2, J_3, J_4, J_5 và C_{jj_k} là thời gian hoàn thành thao tác thứ j của công việc j_k. Chúng ta có công thức tính C_{jj_k} như sau:

$$C_{jj_k} = \begin{cases} \sum_{i=1}^{j} p_{ij_k} & \text{nếu } j_k = j_1 \quad \{j_k \text{ được xử lý đầu tiên}\} \\ \sum_{i=1}^{k} p_{ij_k} & \text{nếu } j = 1 \quad \{\text{các thao tác được xử lý trên máy } M_1\} \\ \max(C_{j-1j_k}, C_{jj_{k-1}}) + p_{jj_k} & \{\text{các trường hợp còn lại}\} \end{cases}$$

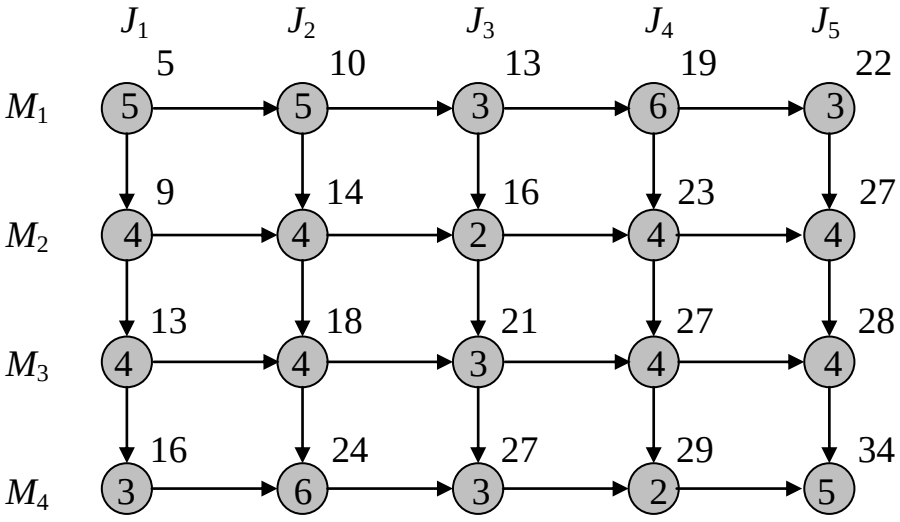
- **Đồ thị không liên thông của một lịch biểu hoán vị**

Để giải thích dễ hiểu hơn về cách tính thời gian hoàn thành của một lịch biểu hoán vị, chúng ta có thể dùng đồ thị không liên thông để biểu diễn.

Hình 2.2 là đồ thị không liên thông của một lịch biểu hoán vị theo thứ tự xử lý: J_1, J_2, J_3, J_4, J_5. Mỗi nút trên đồ thị biểu diễn một thao tác, con số ghi trên mỗi nút là thời gian xử lý của thao tác đó trên máy M_j.



Hình 2.2 - Đồ thị không liên thông biểu diễn một lời giải của PFSP

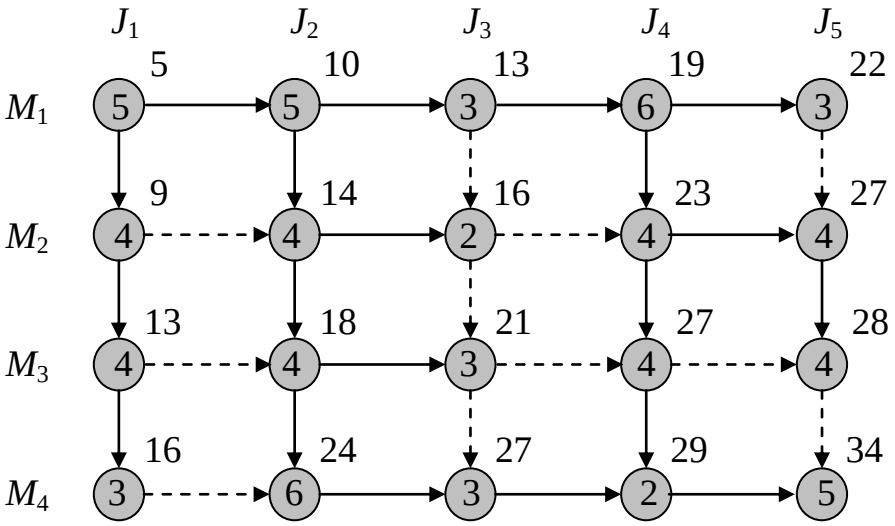


Hình 2.3 - Cách tính thời gian hoàn thành trong đồ thị không liên thông

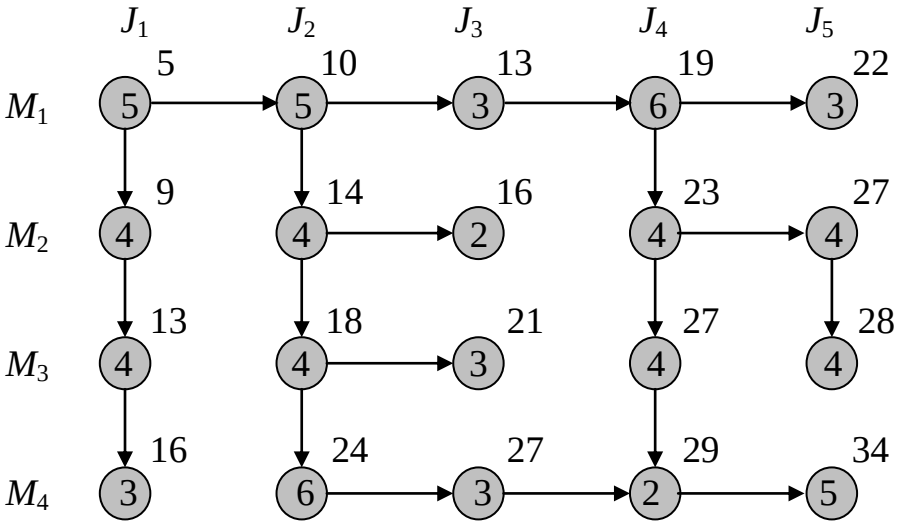
Thời gian hoàn thành của mỗi nút trong đồ thị không liên thông bằng thời gian xử lý của nút (con số trong ô tròn) đó cộng với max {thời gian hoàn thành của nút ngay bên trái, thời gian hoàn thành của nút ngay phía trên}. Hình 2.3 minh họa cho việc tính thời gian hoàn thành của các nút trong đồ thị

không liên thông (thời gian hoàn thành của mỗi nút được biểu diễn bởi các con số phía trên mỗi nút).

- **Cạnh tới hạn, đường tới hạn**



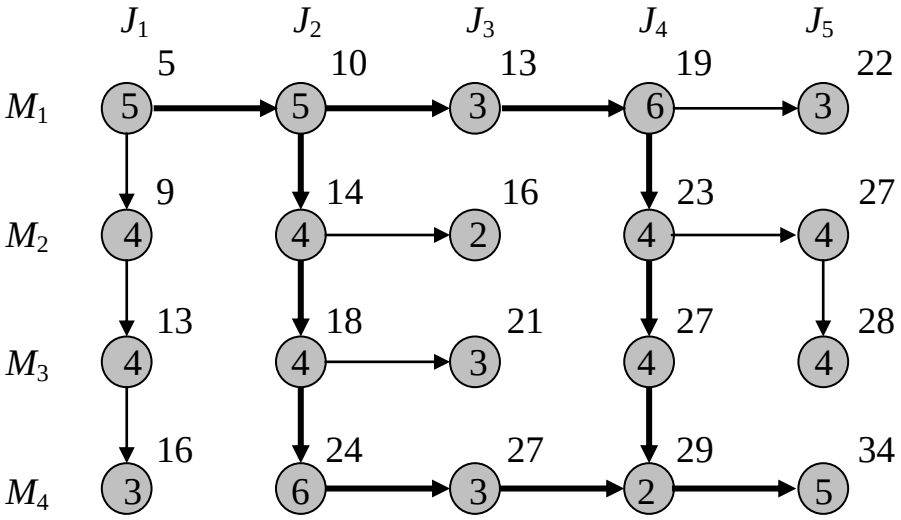
Hình 2.4 - Các cạnh tới hạn của đồ thị không liên thông



Hình 2.5 - Đồ thị cạnh tới hạn

Một cạnh giữa nút (j, j_k) và nút (j', $j_{k'}$) được gọi là cạnh tới hạn nếu nó được dùng để tính thời gian hoàn thành của nút (j', $j_{k'}$). Trên hình 2.4 các cạnh nét liền là các cạnh tới hạn. Nếu loại bỏ các cạnh không phải là cạnh tới hạn trên hình 2.4, chúng ta có đồ thị chỉ với các cạnh tới hạn như trong hình 2.5.

Đường tới hạn là đường đi từ nút bắt đầu tới nút kết thúc (hình 2.6). Tổng độ dài của đường tới hạn chính là thời gian hoàn thành tất cả các công việc của một lịch biểu. Như vậy, đối với lịch biểu J_1, J_2, J_3, J_4, J_5 có 2 đường tới hạn khác nhau (đường bao gồm các mũi tên tô đậm).



Hình 2.6 - Đồ thị đường tới hạn

2.1.3. Thuật toán Johnson cho PFSP 2 máy và PFSP 3 máy

Bài toán lập lịch flow shop hoán vị 2 máy và 3 máy thỏa mãn một số điều kiện nhất định, có thể áp dụng thuật toán Johnson [41] được đề xuất vào năm 1954 để tìm ra lời giải tối ưu thực sự. Trong mục này luận án trình bày phương pháp áp dụng thuật toán Johnson để tìm lịch biểu tối ưu thực sự cho PFSP 2 máy và PFSP 3 máy có hạn chế điều kiện.

a. Thuật toán Johnson cho PFSP 2 máy

- **Phát biểu bài toán**

Cho n công việc $J_1, J_2, ..., J_n$, mỗi công việc đều có 2 công đoạn được xử lý theo thứ tự lần lượt trên 2 máy A và B có các đặc trưng sau:

1. Không có thời gian dừng khi chuyển từ máy A sang máy B.

2. Tại một thời điểm, mỗi máy chỉ có thể xử lý một công việc.

3. Trình tự xử lý các công việc trên 2 máy là như nhau.

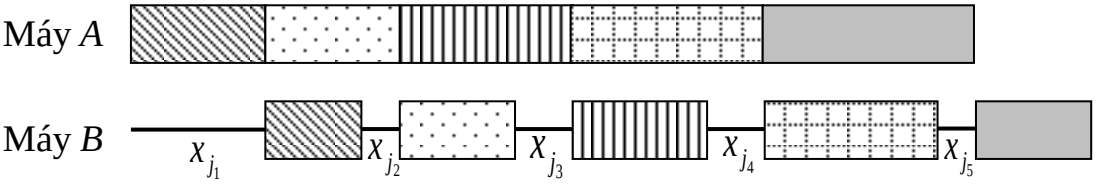
4. Thời gian công việc J_i ($i = 1, ..., n$) trên máy A là a_i, trên máy B là b_i.

Bài toán đặt ra là tìm một trình tự xử lý các công việc sao cho thời gian hoàn thành việc xử lý tất cả các công việc trên cả 2 máy là nhỏ nhất.

Mỗi qui trình xử lý các công việc trên 2 máy A, B tương ứng với một hoán vị $\pi = (j_1, j_2, ..., j_n)$ của n công việc đã cho. Thời gian hoàn thành việc xử lý tất cả các công việc theo qui trình π là:

$$t(\pi) = \sum_{k=1}^{n} x_{j_k} + \sum_{k=1}^{n} b_k \qquad (1)$$

Trong đó, x_{j_k} là thời gian từ khi máy B xử lý xong công việc j_{k-1} đến khi máy B bắt đầu xử lý công việc j_k. Ký hiệu tổng thứ nhất của (1) là $t_B(\pi)$. Vì tổng thứ 2 là hằng số với mọi π, việc giải bài toán được qui về tìm một hoán vị π sao cho $t_B(\pi)$ là nhỏ nhất: $\min\{t_B(\pi): \pi \in P\}$.



Hình 2.7 - Makespan của PFSP 2 máy

- **Thuật toán Johnson**

Bổ đề 1: Giả sử $\pi = (j_1, ..., j_{k-1}, j_k, j_{k+1}, ..., j_n)$ là một hoán vị, π' là một hoán vị nhận được từ π bằng cách đổi chỗ 2 phần tử j_k và j_{k+1}:

$$\pi' = (j_1, ..., j_{k-1}, j_{k+1}, j_k, ..., j_n).$$

Nếu $\min(a_{j_k}, b_{j_{k+1}}) \le \min(a_{j_{k+1}}, b_{j_k})$ thì $t_B(\pi) \le t_B(\pi')$.

Bổ đề 2: Nếu p, q, r là 3 chỉ số thoả mãn:

$\min(a_p, b_q) \le \min(a_q, b_p)$, $\min(a_q, b_r) \le \min(a_r, b_q)$, và nếu $a_q \ne b_q$ thì $\min(a_p, b_r) \le \min(a_r, b_p)$.

Từ bổ đề 1 và bổ đề 2, định lý Johnson được phát biểu như sau:

Đính lý Johnson: $t_B(\pi)$ đạt giá trị nhỏ nhất khi hoán vị $\pi = (j_1, j_2, ..., j_n)$ thỏa mãn: $\min(a_{j_k}, b_{j_{k+1}}) \le \min(a_{j_{k+1}}, b_{j_k})$, với mọi $k = 1, 2, ..., n - 1$.

- **Ví dụ minh họa**

Xét PFSP 4 công việc 2 máy, thời gian xử lý các công việc trên mỗi máy được cho trong bảng 2.2.

Bảng 2.2 - PFSP 4 công việc 2 máy

	J_1	J_2	J_3	J_4
A	1	5	3	4
B	2	3	2	5

Thuật toán Johnson trong thực tế được tiến hành theo các bước sau:

Bước 1: Chọn số nhỏ nhất trong bảng 2.2, nếu số đó thuộc hàng máy A thì công việc tương ứng được lập lịch xử lý đầu tiên. Nếu số đó thuộc hàng

máy *B* thì công việc tương ứng được lập lịch xử lý sau cùng.

Bước 2: Xóa cột tương ứng với công việc vừa được lập lịch.

Bước 3: Nếu mọi công việc đã được lập lịch thì dừng, hoán vị thu được là một lời giải tối ưu. Ngược lại, quay lên bước 1.

Chú ý: Trong trường hợp có hơn một giá trị nhỏ nhất, chúng ta có thể chọn một số bất kỳ trong các số nhỏ nhất đó.

Trở lại bài toán đang được xét:

+ Trong bảng 2.2 số nhỏ nhất là 1 nằm trên hàng máy *A*, vì vậy công việc J_1 được lập lịch đầu tiên.

+ Xóa cột J_1 trong bảng 1 chúng ta được bảng 2.3:

Bảng 2.3 - Các công việc chưa được lập lịch

	J_2	J_3	J_4
A	5	3	4
B	3	2	5

+ Trong bảng 2.3, số nhỏ nhất là 2 nằm trên hàng máy *B*, vì vậy công việc J_3 được lập lịch sau cùng (vị trí thứ 4).

Bảng 2.4 - Các công việc chưa được lập lịch

	J_2	J_4
A	5	4
B	3	5

+ Xóa cột J_3 trong bảng 2.3 chúng ta được bảng 2.4.

+ Trong bảng 2.4, số nhỏ nhất là 3 nằm trên hàng máy *B*, vì vậy công việc J_2 được lập lịch sau cùng trong số các vị trí còn lại (vị trí thứ 3).

+ Xóa cột J_2 trong bảng 2.4 chúng ta có bảng 2.5:

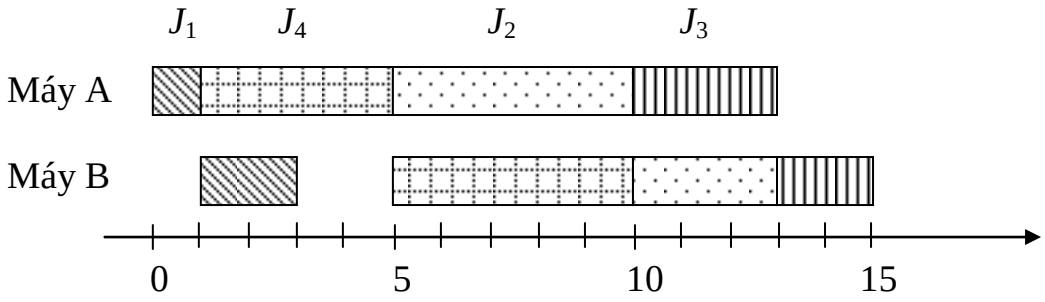
Bảng 2.5 - Các công việc chưa được lập lịch

	J_4
A	*4*
B	*5*

Công việc J_4 được lập lịch vào vị trí còn lại (vị trí thứ 2).

Chúng ta có hoán vị thu được là một lịch biểu tối ưu sau: J_1, J_4, J_2, J_3.

Biểu đồ Grant và thời gian hoàn thành của lịch biểu tối ưu trên được trình bày trong hình 2.8.



Hình 2.8 - Biểu đồ Grant của lịch biểu tối ưu bài toán 2 máy

Như vậy, thời gian ngắn nhất để hoàn thành việc xử lý 4 công việc trên 2 máy của bài toán đã cho là 15 đơn vị thời gian.

b. Thuật toán Johnson cho PFSP 3 máy

- **Phát biểu bài toán**

Cho n công việc J_1, J_2, ..., J_n, mỗi công việc đều có 3 công đoạn được xử lý theo thứ tự lần lượt trên 3 máy A, B và C có các đặc trưng sau:

1. Không có thời gian dừng khi chuyển từ máy này sang máy kia.

2. Tại một thời điểm, mỗi máy chỉ có thể xử lý một công việc.

3. Trình tự xử lý các công việc trên 3 máy là như nhau.

4. Thời gian công việc J_i (i = 1, ..., n) trên máy A là a_i, trên máy B là b_i và trên máy C là c_i.

Bài toán đặt ra là tìm một trình tự xử lý các công việc sao cho thời gian hoàn thành việc xử lý tất cả các công việc trên cả 3 máy là nhỏ nhất.

Thuật toán Johnson áp dụng được cho PFSP 3 máy nếu như một trong 2 điều kiện hoặc cả 2 điều kiện sau đây thỏa mãn:

$$1)\ \max b_i \le \min a_i \qquad 2)\ \max b_i \le \min c_i \qquad (2)$$

Thuật toán Johnson áp dụng cho PFSP 3 máy thỏa mãn ràng buộc (2) như sau:

Bước 1: Lập 2 máy giả là G và H với thời gian xử lý công việc J_i trên máy G là $a_i + b_i$ và trên máy H là $b_i + c_i$.

Bước 2: Áp dụng thuật toán Johnson tìm lịch biểu tối ưu trên 2 máy G và H. Lịch biểu tối ưu thu được cũng chính là lịch biểu tối ưu trên 3 máy A, B, và C.

- **Ví dụ minh họa**

Xét PFSP 5 công việc 3 máy, thời gian xử lý các công việc trên mỗi máy được cho trong bảng 2.6.

Bảng 2.6 - PFSP 5 công việc 3 máy

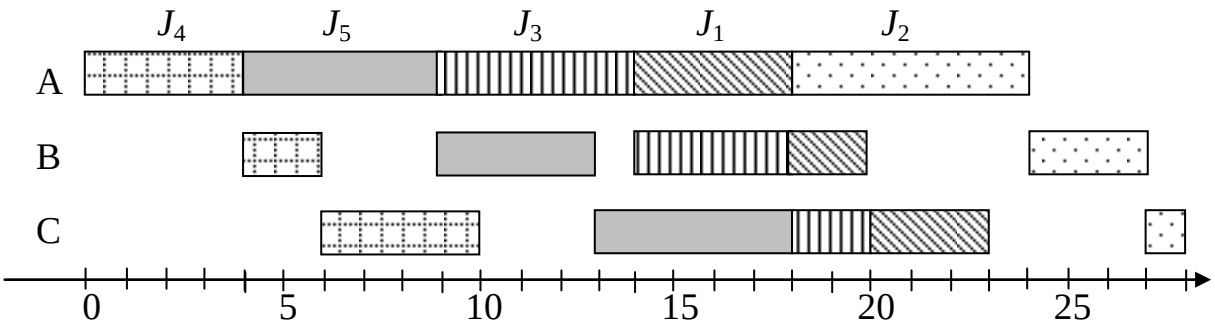
	J_1	J_2	J_3	J_4	J_5
A	4	6	5	4	5
B	2	3	4	2	4
C	3	1	2	4	5

Từ bảng 2.6, chúng ta thấy min $a_i = 4$, max $b_i = 4$, min $c_i = 1$, ràng buộc (2) thỏa mãn (max $b_i \leq$ min a_i). Áp dụng thuật toán Johnson cho bài toán 3 máy bằng cách xây dựng 2 máy giả *G* và *H* như trong bảng 2.7.

Bảng 2.7 - Thời gian xử lý các công việc trên 2 máy *G* và *H*

	J_1	J_2	J_3	J_4	J_5
G	6	9	9	6	9
H	5	4	6	6	9

Áp dụng thuật toán Johnson cho PFSP 2 máy, chúng ta thu được lịch biểu tối ưu trên 2 máy *G* và *H* sau: J_4, J_5, J_3, J_1, J_2. Đây cũng chính là lịch biểu tối ưu trên 3 máy thật *A*, *B*, và *C*. Biểu đồ Grant và thời gian hoàn thành của lịch biểu tối ưu trên được trình bày trong hình 2.9.



Hình 2.9 - Biểu đồ Grant của lịch biểu tối ưu bài toán 3 máy

Như vậy, thời gian ngắn nhất để hoàn thành việc xử lý 5 công việc trên 3 máy của bài toán đã cho là 28 đơn vị thời gian.

2.1.4. Một thuật toán di truyền mã hóa tự nhiên cho bài toán lập lịch flow shop hoán vị tổng quát

Thuật toán Johnson chỉ áp dụng được cho các PFSP 2 máy hoặc 3 máy thỏa mãn ràng buộc (2). Như vậy, với các PFSP tổng quát không thể giải được bằng thuật toán Johnson [41]. Đối với các bài toán này, chúng ta phải dùng các phương pháp gần đúng để giải quyết chúng, một trong số các phương pháp gần đúng được áp dụng có hiệu quả nhất cho PFSP tổng quát là thuật toán di truyền. Trong mục này luận án đề xuất một thuật toán di truyền mới áp dụng cho PFSP.

- **Mã hóa lời giải**

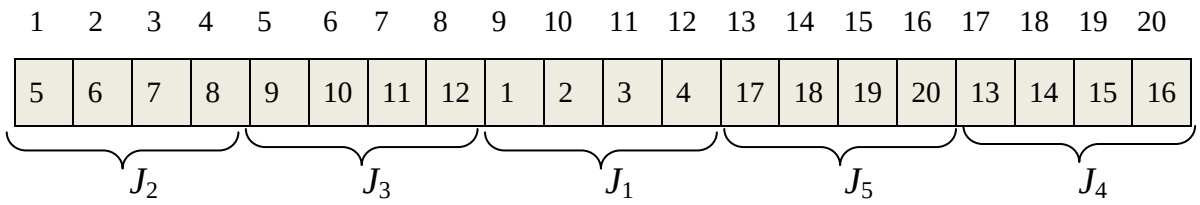
Bảng 2.8 - Mã hóa lời giải theo số tự nhiên

Công việc	Mã hóa thao tác				
J_1	1	2	3	4	5
J_2	6	7	8	9	10
J_3	11	12	13	14	15
J_4	16	17	18	19	20

Với PFSP n công việc m máy, tổng số các thao tác cần phải được xử lý trong một qui trình là $l = n \cdot m$. Chúng ta mã hóa l thao tác cần được xử lý theo số tự nhiên như sau:

Đánh số các thao tác của công việc J_1 từ 1 đến m, của công việc J_2 từ m + 1 đến 2m, ..., của công việc J_n từ $(n - 1)m + 1$ đến $n \cdot m$. Như vậy, một lời giải sẽ là một hoán vị nào đó của dãy số tự nhiên {1, 2, ..., l}. Tuy nhiên, chỉ những hoán vị tuân thủ thứ tự thao tác của tất cả các công việc mới là lời giải của bài toán.

Ví dụ, bài toán 4 công việc 5 máy, các thao tác sẽ được mã hóa như trong bảng 2.8. Một lời giải hợp lệ của bài toán có thể được biểu diễn dưới dạng một vectơ mà các phần tử là một hoán vị của các số tự nhiên trong bảng 2.8 như sau:



Hình 2.10 - Một lời giải hợp lệ cho PFSP 4 công việc 5 máy

- **Hàm đánh giá độ thích nghi**

Hàm đánh giá độ thích nghi của mỗi lời giải được ký hiệu là fitness được xây dựng theo công thức sau:

$$\text{fitness} = M - g(v)$$

Trong đó, $g(v)$ là thời gian hoàn thành của mỗi lời giải v, M là tham số dương được đưa vào sao cho $M - g(v) > 0$ với mọi v. Bài toán tìm min được chuyển đổi thành bài toán tìm max để tiện nghiên cứu hơn. Như vậy, nhiệm vụ của bài toán là phải tìm một lịch biểu v sao cho fitness = $M - g(v)$ đạt giá trị lớn nhất (tương đương với $g(v)$ nhỏ nhất).

- **Khởi tạo tập lời giải ban đầu**

Để khởi tạo pop_size (pop_size là cỡ quần thể) lời giải cho thế hệ đầu tiên, chúng ta tiến hành theo các bước sau:

Bước 1:

+ Xây dựng một mảng một chiều *n*_job[1..*n*] để lưu số thao tác của mỗi công việc đã được lập lịch. Mảng này gồm *n* phần tử tương ứng với *n* công việc, ban đầu các giá trị của các phần tử của mảng này đều bằng 0.

+ Xây dựng một danh sách *d* lưu các công việc chưa được lập lịch xong. Ban đầu *d* gồm *n* phần tử (tương ứng với *n* công việc).

Bước 2:

Lặp lại qui trình sau cho đến khi $d = \varnothing$.

1. Chọn ngẫu nhiên một số tự nhiên là một phần tử của *d* (chọn một công việc để lập lịch).

Ví dụ, số ngẫu nhiên là 2, tức là J_2 được chọn để lập lịch.

2. Lập lịch cho tất cả các thao tác của công việc vừa được chọn.

3. Xóa công việc này khỏi danh sách *d*. Quay lên bước 1.

- **Các toán tử di truyền**

Toán tử chọn lọc

Toán tử chọn lọc nhằm mục đích chọn các lời giải tốt cho thế hệ sau dựa trên giá trị của hàm thích nghi. Về cơ bản, cơ chế hoạt động của toán tử này tương tự như toán tử chọn lọc trong thuật toán di truyền cổ điển đã trình bày ở chương 1. Tức là chọn lọc một cách ngẫu nhiên theo nguyên tắc bánh xe xổ số.

Toán tử đột biến

Toán tử đột biến thực hiện trên một cá thể cha, số các cá thể cha được chọn để đột biến theo một tỷ lệ nhất định tùy thuộc vào xác suất đột biến p_m. Phép đột biến được tiến hành theo các bước sau:

1. Chọn ngẫu nhiên một thao tác (ký hiệu là *ope*1) trong cá thể cha. Xác định công việc chứa thao tác đó (ký hiệu là *job*1) và vị trí của thao tác đó (ký hiệu là *pos*1).

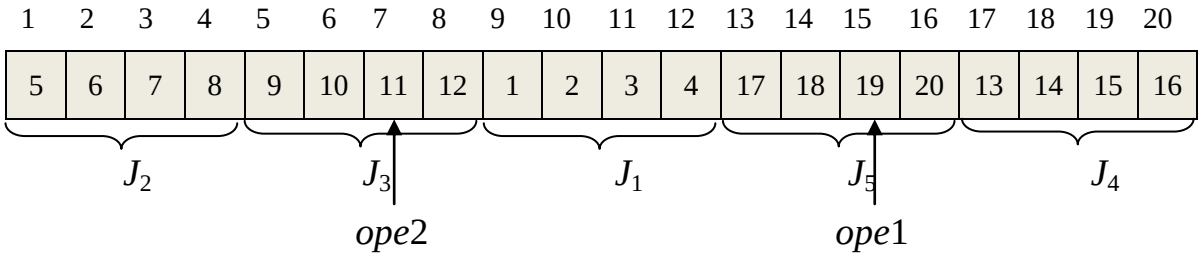
2. Chọn ngẫu nhiên một thao tác (*ope*2) trong cá thể cha. Xác định công việc chứa thao tác đó (*job*2) và vị trí của thao tác đó (*pos*2).

3. Nếu *job*1 ≠ *job*2 thì tiến hành đột biến bằng cách chèn các thao tác của job1 vào vị trí các thao tác của *job*2 (hoặc hoán đổi vị trí của chúng cho nhau), chúng ta được cá thể con. Việc chèn sẽ xẩy ra 2 trường hợp sau đây:

- Trường hợp 1: *pos*1 > *pos*2, dồn các thao tác của *job*2 sang phải *m* vị trí để lấy chỗ cho các thao tác của *job*1.

- Trường hợp 2: *pos*1 < *pos*2, dồn các thao tác của *job*2 sang trái *m* vị trí để lấy chỗ cho các thao tác của *job*1.

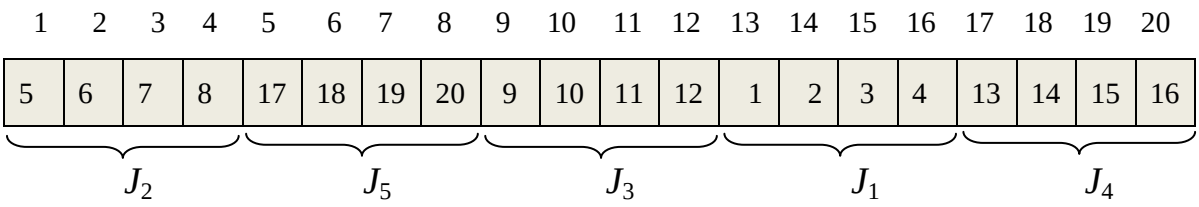
Ví dụ, cá thể cha được chọn để đột biến như hình 2.11:



Hình 2.11 - Cá thể cha

+ Chẳng hạn, *ope*1 = 19 → *job*1 = J_5 và *pos*1 = 15, *ope*2 = 11 → job2 = J_3 và pos2 = 7.

+ $job1 \neq job2$ và $pos1 > pos2$ → chèn các thao tác của J_5 vào vị trí các thao tác của J_3 (dồn các thao tác của J_3 sang phải 4 vị trí), chúng ta được cá thể con sau đột biến như trong hình 2.12.



Hình 2.12 - Cá thể con sau phép đột biến

Toán tử trao đổi chéo

Toán tử trao đổi chéo được thực hiện trên 2 cá thể cha. Các gien trong cá thể con được kết hợp từ các gien trong 2 cá thể cha theo một qui luật nào đó. Toán tử lai ghép được tiến hành theo các bước sau:

1. Tung ngẫu nhiên đồng xu, nếu kết quả là mặt sấp (tương ứng với 1) thì chọn công việc từ cha thứ 2 đưa vào cá thể con. Ngược lại chọn từ cha thứ nhất. Việc tung đồng xu được tiến hành *n* lần tương ứng với *n* công việc.

Việc chọn các công việc từ mỗi các cá thể cha đưa vào cá thể con được thực hiện như sau: Chúng ta kiểm tra cá thể cha từ trái sang phải, nếu gặp công việc chưa có trong cá thể con thì chọn để đưa vào cá thể con. Tiến trình kết thúc khi số công việc trong cá thể con là *n*.

2. Từ dãy các công việc vừa được chọn cho cá thể con, chúng ta chuyển thành một dãy bao gồm tập các thao tác của mỗi công việc để có được cá thể con thực sự có mã hóa các thao tác là các số tự nhiên.

Ví dụ, hai cá thể cha được chọn để lai ghép như trong hình 2.13.

1	2	3	4	5	6	7	8	9	10	11	12	13	15	15	16	17	18	19	20
5	6	7	8	9	10	11	12	1	2	3	4	17	18	19	20	13	14	15	16

parent1

1	2	3	4	5	6	7	8	9	10	11	12	13	15	15	16	17	18	19	20
13	14	15	16	5	6	7	8	9	10	11	12	1	2	3	4	17	18	19	20

parent2

Hình 2.13 - Các cá thể cha tham gia trao đổi chéo

Giả sử, các kết quả tung đồng xu là: 0, 0, 1, 0, 1. Khi đó cá thể con có các công việc được chọn từ parent1 và parent2 là: J_2, J_3 (từ parent1), J_4 (từ parent2), J_1 (từ parent1) và J_5 (từ parent2). Cá thể con sau lai ghép như được biểu diễn trong hình 2.14.

1	2	3	4	5	6	7	8	9	10	11	12	13	15	15	16	17	18	19	20
5	6	7	8	9	10	11	12	13	14	15	16	1	2	3	4	17	18	19	20

Hình 2.14 - Cá thể con sau phép trao đổi chéo

- **Thủ tục tiến hóa**

Thủ tục tiến hóa cho bài toán lập lịch flow shop hoán vị được đặc tả vắn tắt như sau:

Procedure GA_PFSP

Begin

t ← 0

Khởi tạo P(t)

Đánh giá P(t)

While (not điều kiện dừng) do

Begin

Xây dựng tập lời giải trung gian P'(t):

+ Áp dụng toán tử đột biến với P(t) được $P_1(t)$

+ Áp dụng toán tử trao đổi chéo với P(t) được $P_2(t)$

+ $P'(t) = P(t) \cup P_1(t) \cup P_2(t)$

Đánh giá P'(t)

$t \leftarrow t + 1$

Áp dụng toán tử chọn lọc với P'(t-1) được P(t)

End

End

2.1.5. Các kết quả thử nghiệm

Dựa trên phương pháp được đề nghị, luận án đã cài đặt một chương trình giải gần đúng cho bài toán lập lịch flow shop hoán vị với thời gian xử lý của tất cả các thao tác đều dương. Chương trình đã được chạy thử nghiệm trên các bài toán test đã biết trước kết quả tối ưu. Với các bài toán test cỡ không lớn, số các lần chạy đạt được kết quả tối ưu thực sự. Kết quả chạy thử nghiệm được thống kê trong bảng 2.9.

Bảng 2.9 - Kết quả chạy thử nghiệm

Bài toán ($m \times n$)	Cỡ lời giải	Số thế hệ	p_c Xác suất lai ghép	p_m Xác suất đột biến	Kết quả chạy	Tối ưu thực sự
3×5	20	100	0.5	0.5	28	28
4×5	20	100	0.5	0.5	32	32
3×5	20	100	0.5	0.5	49	49

2.2. Bài toán lập lịch flow shop

Bài toán lập lịch flow shop (flow shop scheduling problem - FSP) cũng là bài toán con của JSP nhưng là trường hợp tổng quát hơn bài toán lập lịch flow shop hoán vị. Đối với bài toán này, tuần tự công nghệ của tất cả các công việc là như nhau, nhưng thứ tự xử lý các công việc ở trên mỗi máy có thể khác nhau.

2.2.1. Mô tả bài toán

Bài toán lập lịch flow shop (FSP) là bài toán có n công việc (J_1, J_2, ..., J_n) được xử lý trên m máy (M_1, M_2, ..., M_m) và có các đặc trưng sau đây:

1. Mỗi công việc J_i (i = 1, ..., n) có m thao tác, thao tác thứ j phải được xử lý ở trên máy M_j (j = 1, ..., m). Như vậy, một công việc chỉ có thể bắt đầu được xử lý ở trên máy M_j nếu nó được hoàn thành việc xử lý ở trên máy M_{j-1} và máy M_j đang rỗi.

2. Trình tự xử lý các công việc ở trên các máy có thể khác nhau (đây là đặc trưng khác biệt của FSP với PFSP).

3. Thao tác của công việc J_i được xử lý ở trên máy M_j được ký hiệu là O_{ij} và có thời gian xử lý cho trước là p_{ij}. Thời gian hoàn thành thao tác O_{ij} được tính tương tự như đối với bài toán flow shop hoán vị.

4. Khoảng thời gian kể từ khi bắt đầu xử lý các công việc cho tới khi hoàn thành việc xử lý tất cả các công việc được gọi là makespan của bài toán và được ký hiệu là C_{max}.

Việc giải quyết FSP là xác định một lịch biểu (thứ tự xử lý các công việc ở trên mỗi máy) sao cho makespan là nhỏ nhất có thể.

Conway, Maxwell và Miller [16] đã chứng minh rằng đối với mọi bài toán lập lịch flow shop, luôn tồn tại một lịch biểu tối ưu mà thứ tự xử lý các

công việc ở trên 2 máy đầu và 2 máy cuối là như nhau. Điều này có nghĩa là với các bài toán lập lịch flow shop chỉ có 2 hoặc 3 máy thì luôn tồn tại một lịch biểu tối ưu hoán vị. Nhưng với các FSP 4 máy trở lên thì điều đó không còn được đảm bảo nữa. Để xác nhận điều đó, chúng ta có thể xem xét ví dụ sau đây:

Cho bài toán flow shop 4 máy, 2 công việc và có thời gian xử lý các công việc ở trên mỗi máy như trong bảng 2.10. Lịch biểu flow shop tối ưu có thời gian hoàn thành là 12, trong khi lịch biểu flow shop hoán vị tối ưu có thời gian hoàn thành là 14.

Bảng 2.10 - FSP 4 máy 2 công việc

	M_1	M_2	M_3	M_4
J_1	4	1	1	4
J_2	1	4	4	1

Trong trường hợp tổng quát, ký hiệu Ø(m) là tỷ số giữa makespan của lịch biểu hoán vị tốt nhất và makespan của lịch biểu flow shop tốt nhất, ở đây m là số máy. Röck và Schmidt [59] đã chứng minh được rằng: $Ø(m) \leq \lceil m/2 \rceil$, còn Shmoys và Williamson và những người khác [75] đã chứng minh được rằng: $Ø(m) \geq \lceil \sqrt{m} + 1/2 \rceil / 2$. Tỷ số chính xác cho tới nay vẫn chưa tìm được.

2.2.2. Một thuật toán di truyền mã hóa tự nhiên cho bài toán lập lịch flow shop tổng quát

- **Mã hoá lời giải**

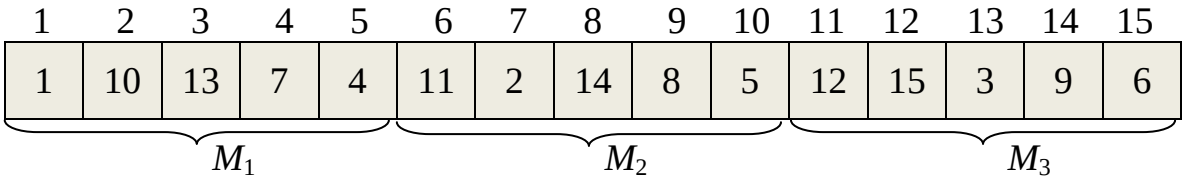
Đối với FSP n công việc, m máy, tổng số các thao tác cần phải thực hiện trong một qui trình là $l = n \cdot m$. Chúng ta đánh số các thao tác của công việc J_1 từ 1 đến m, của J_2 từ $m + 1$ đến $2m$, ..., của J_n từ $(n - 1)m + 1$ đến

n·m. Một lời giải hợp lệ cho bài toán là một dãy bao gồm hoán vị của *n* thao tác đầu tiên, tiếp theo là hoán vị của *n* thao tác thứ hai,..., cuối cùng là hoán vị của *n* thao tác thứ *m*.

Bảng 2.11 - Mã hóa lời giải theo số tự nhiên

Công việc	Mã hoá thao tác		
J_1	1	2	3
J_2	4	5	6
J_3	7	8	9
J_4	10	11	12
J_5	13	14	15

Ví dụ, bài toán 5 công việc, 3 máy. Các thao tác được mã hoá bằng số tự nhiên như bảng 2.11, một lời giải hợp lệ có thể có dạng như hình 2.15:



Hình 2.15 - Một lời giải hợp lệ cho FSP 3 máy × 5 công việc

- **Xây dựng hàm thích nghi**

Hàm đánh giá độ thích nghi của mỗi lời giải được ký hiệu là fitness được xây dựng tương tự như trong trường hợp PFSP:

fitness = M - $g(v)$

- **Khởi tạo tập lời giải ban đầu**

Để khởi tạo một lời giải cho thế hệ đầu $P(0)$, chúng ta tiến hành theo các bước sau:

Bước 1:

+ Xây dựng một mảng *n_job*[1..*n*]. Mảng này gồm có *n* phần tử, tương ứng với *n* công việc. *n_job*[*i*] lưu số thao tác của công việc J_i đã được lập lịch. Ban đầu các giá trị của mảng này đều bằng 0.

+ Xây dựng một danh sách *d* lưu các công việc chưa được lập lịch xong. Ban đầu mảng *d* gồm *n* phần tử (tương ứng với *n* công việc).

Bước 2:

Lặp lại quá trình sau cho đến khi *d* không còn phần tử nào:

1. Chọn ngẫu nhiên một phần tử trong *d* (chọn một công việc để lập lịch). Ví dụ: *job* = ramdom(*n*) + 1 = 2 → J_2 được chọn để lập lịch.

2. Lập lịch cho cho thao tác tiếp theo của công việc vừa được chọn:

- Công việc được lập lịch là *job*.

- Thao tác được lập lịch là: *n_job*[*job*], (ban đầu *n_job[job*] = 0, mỗi lần job được chọn, *n_job*[*job*] tăng thêm 1).

- Xác định mã của thao tác vừa lập lịch: ope = *n_job*[*job*] + (*job*-1)*m*.

- Xác định máy thực hiện thao tác vừa được lập lịch như sau: Lấy mã của thao tác chia cho *m*, nếu không dư thì thương số chính máy thực hiện thao tác đó, nếu có dư thì máy thực hiện thao tác đó bằng thương số cộng 1.

- Tăng số thao tác được lập lịch trên máy đó lên 1.

3. Xoá công việc này trong danh sách *d* nếu *n_job*[*job*] = *m*.

Qui trình trên được lặp lại pop_size (cỡ quần thể) lần.

- **Các toán tử di truyền**

Toán tử chọn lọc

Cơ chế hoạt động của toán tử này tương tự như toán tử chọn lọc trong thuật toán di truyền cổ điển đã trình bày ở chương 1. Tức là chọn lọc một cách ngẫu nhiên theo nguyên tắc bánh xe xổ số.

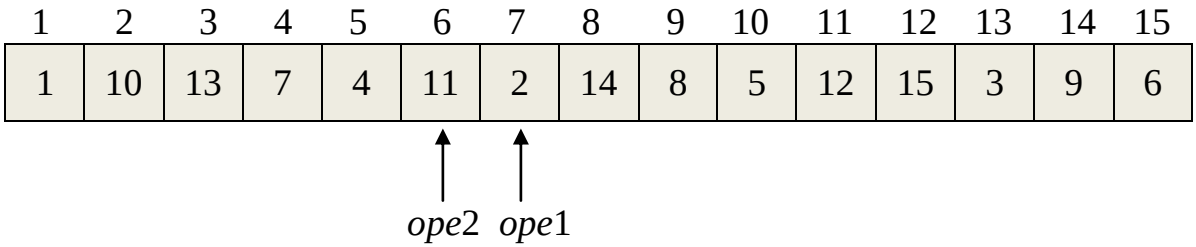
Toán tử đột biến

Toán tử đột biến được tiến hành theo các bước sau:

1. Chọn ngẫu nhiên một thao tác (ký hiệu là *ope*1) trong cá thể cha. Xác định máy thực hiện thao tác đó (ký hiệu là M_{ope1}) và vị trí của thao tác đó trong lơi giải (ký hiệu là *pos*1).

2. Chọn ngẫu nhiên một thao tác (*ope*2) trong cá thể cha. Xác định máy thực hiện thao tác đó (M_{ope2}) và vị trí của thao tác đó trong lời giải (*pos*2).

3. Nếu $M_{ope1} = M_{ope2}$ thì tiến hành đột biến (chèn thao tác *ope*1 vào vị trí *pos*2 hay hoán đổi vị trí của hai thao tác). Kết quả cho chúng ta cá thể con. Trong trường hợp $M_{ope1} \neq M_{ope2}$ thì cá thể cha được giữ nguyên.



Hình 2.16 - Cá thể cha cho phép đột biến

Ví dụ, cá thể cha được chọn để đột biến biểu diễn trong hình 2.16.

+ Chẳng hạn, bước 1 chọn được *ope*1 = 2 → M_{ope1} = 2 và *pos*1 = 7.

+ Bước 2 chọn được *ope*2 = 11→ M_{ope2} = 2 và *pos*2 = 6.

+ M_{ope1} = M_{ope2}, tiến hành đột biến (chèn thao tác 2 vào vị trí 6), chúng ta có cá thể con sau khi đột biến như được biểu diễn trong hình 2.17.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	10	13	7	4	2	11	14	8	5	12	15	3	9	6

Hình 2.17 - Cá thể con sau phép đột biến

Toán tử lai ghép

Toán tử lai ghép được thực hiện trên 2 cá thể cha (ký hiệu là *parent*1 và *parent*2). Các gien trong cá thể con sẽ được tái kết hợp từ các gien trong 2 cá thể cha. Toán tử lai ghép được tiến hành theo các bước sau:

1. Chọn ngẫu nhiên một máy trong số *m* máy: *mach* = *random*(*m*) + 1.

2. Xác định vị trí trao đổi chéo: *pos* = (*mach* - 1)*n* + 1.

Ví dụ, các thể cha được chọn lai ghép được biểu diễn trong hình 2.18.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
*Parent*1	1	10	13	7	4	5	14	11	8	2	6	15	12	9	3

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
*Parent*2	4	13	10	7	1	2	11	14	8	5	12	15	3	9	6

Hình 2.18 - Các cá thể cha tham gia trao đổi chéo

+ Giả sử, kết quả lấy ngẫu nhiên là 2. Khi đó *pos* = 1.5 + 1 = 6.

+ Thực hiện phép lai ghép hai cá thể cha tại vị trí 6, chúng ta được hai cá thể con sau khi lai ghép như trong hình 2.19.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
*Child*1	1	10	13	7	4	2	11	14	8	5	12	15	3	9	6
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
*Child*2	4	13	10	7	1	5	14	11	8	2	6	15	12	9	3

Hình 2.19 - Cá thể con sau phép trao đổi chéo

- **Thủ tục tiến hóa**

Thủ tục tiến hóa cho bài toán lập lịch flow shop tương tự như trong trường hợp PFSP.

2.2.3. Các kết quả thử nghiệm

Bảng 2.12 - Kết quả chạy thử nghiệm

Bài toán ($m \times n$)	Cỡ lời giải	Số thế hệ	p_c Xác suất lai ghép	p_m Xác suất đột biến	Kết quả chạy	Tối ưu thực sự
3×5	20	100	0.5	0.5	28	28
4×5	20	100	0.5	0.5	32	32
5×10	50	200	0.5	0.5	789	789

Dựa trên phương pháp được đề nghị, luận án đã cài đặt một chương trình giải gần đúng cho bài toán lập lịch flow shop với thời gian xử lý của tất cả các thao tác đều dương. Chương trình đã được chạy thử nghiệm trên các bài toán test đã biết trước kết quả tối ưu. Với các bài toán test cỡ không lớn, kết quả chạy đạt tối ưu thực sự.

2.3. Kết luận

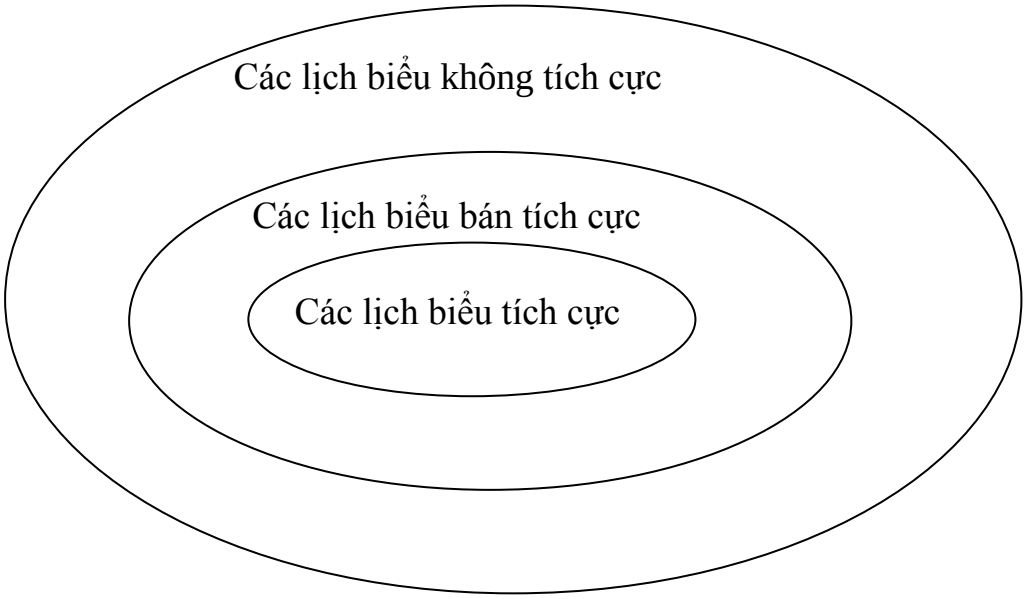
Chương 2 đã trình bày về bài toán flow shop hoán vị và flow shop. Các bài toán lập lịch này chỉ có thể áp dụng các tiếp cận hiệu suất cao cho trường hợp 2 máy và 3 máy có ràng buộc điều kiện dùng thuật toán Johnson [41]. Đối với các trường hợp còn lại của bài toán, chỉ có thể áp dụng các tiếp cận gần đúng để giải quyết chúng. Trong chương này, luận án đã đề xuất một phương pháp dùng thuật toán di truyền mã hóa tự nhiên để giải quyết 2 bài toán này trong trường hợp tổng quát. Thuật toán đề xuất đã được cài đặt và chạy kiểm tra trên các bài toán test đã biết trước kết quả tối ưu cho kết quả tốt. Trên cơ sở nghiên cứu và đề ra giải pháp cho 2 bài toán này, chương tiếp theo của luận án sẽ mở rộng cho bài toán lập lịch tổng quát và phức tạp nhất đó là bài toán lập lịch job shop.

CHƯƠNG 3. MỘT THUẬT TOÁN DI TRUYỀN LAI MỚI CHO BÀI TOÁN LẬP LỊCH JOB SHOP

Bài toán lập lịch job shop (JSP) có nhiều khả năng được đề xuất và nghiên cứu lần đầu tiên bởi Akers và Fiedman [4]. Trong các tài liệu của Liên Xô cũ, JSP thường được gọi là bài toán Akers-Friedman hay bài toán AF. JSP thuộc lớp *NP*-hard và nổi tiếng là một trong những bài toán tối ưu tổ hợp khó tính toán nhất được biết cho tới nay. Trong các bài toán về lập lịch, JSP là bài toán được nghiên cứu nhiều nhất và là một mô hình phát triển tốt nhất về lý thuyết lập lịch. Nó được xem như là một cơ sở để kiểm tra, so sánh các kỹ thuật giải khác nhau, các kỹ thuật cũ và mới,... Ngoài ra, JSP còn được thúc đẩy mạnh mẽ bởi các nhu cầu thực tiễn. Trong chương 1, luận án đã trình bày tổng quan về các kỹ thuật khác nhau được áp dụng cho JSP. Trong chương này, luận án đề xuất một thuật toán di truyền lai mới cho JSP.

3.1. Các lịch biểu tích cực và bán tích cực

B. Giffler và Thompson [28], đã đưa ra kết luận: ″Không cần thiết phải tìm kiếm một lịch biểu tối ưu trong toàn bộ không gian các lịch biểu mà chỉ cần tìm kiếm trong một tập con các lịch biểu khả thi″. Các lịch biểu khả thi này còn được gọi là tập các lịch biểu tích cực. Theo B. Giffler và Thompson, không gian các lịch biểu bao gồm 3 lớp: Các lịch biểu không tích cực, các lịch biểu bán tích cực và các lịch biểu tích cực. Trong đó, tập các lịch biểu tích cực là nhỏ nhất và một lịch biểu tối ưu luôn là một lịch biểu tích cực. Dựa trên kết luận hết sức quan trọng này, chúng ta có thể giới hạn không gian tìm kiếm lời giải tối ưu cho JSP là tập tất cả các lịch biểu tích cực. Hạn chế này sẽ thu gọn không gian tìm kiếm và giảm thời gian trong quá trình tìm kiếm mà vẫn đảm bảo hướng tới được lời giải tối ưu.



Hình 3.1 - Các lớp lịch biểu

Định nghĩa 1: Một lịch biểu được gọi là bán tích cực khi không thao tác nào trong lịch biểu đó có thể được bắt đầu sớm hơn mà không thay đổi thứ tự xử lý các thao tác ở trên một máy nào đó.

Định nghĩa 2: Một lịch biểu được gọi là tích cực khi không thao tác nào trong lịch biểu đó có thể được bắt đầu sớm hơn mà không phá vỡ các ràng buộc đi trước của một số thao tác nào đó.

- **Một ví dụ minh họa**

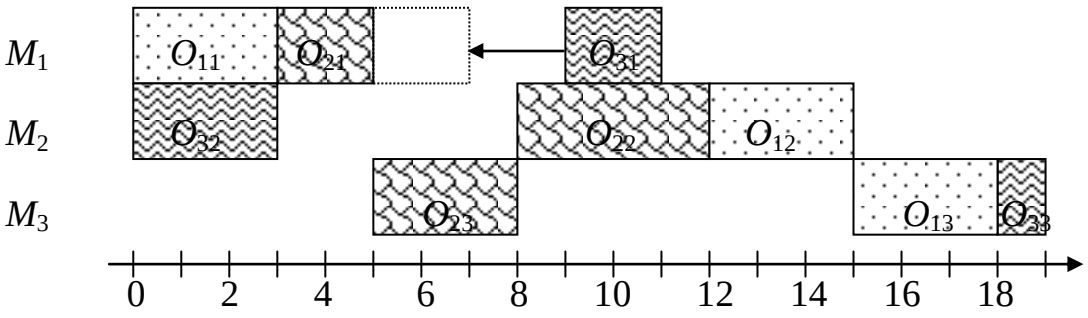
Bảng 3.1 - JSP 3 công việc, 3 máy

Công việc	Máy (thời gian xử lý)		
1	1 (4)	2 (4)	3 (4)
2	1 (3)	3 (4)	2 (5)
3	2 (4)	1 (3)	3 (2)

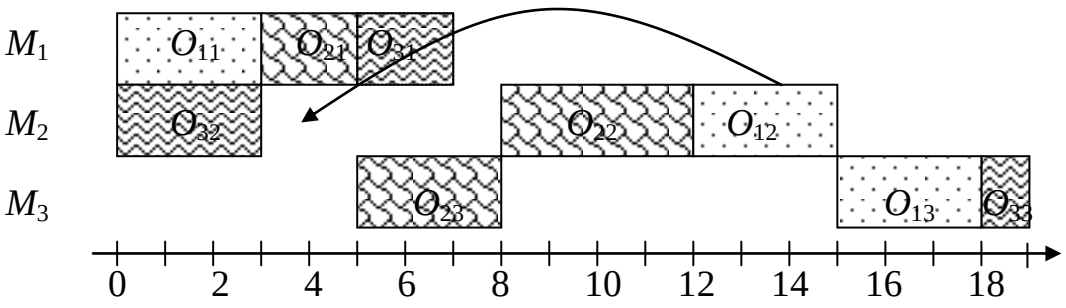
Để minh họa về các lớp lịch biểu, trở lại ví dụ về JSP 3 × 3 đã được trình bày trong mục 1.1 của chương 1 (ví dụ này luận án tham khảo trong [77]). Bảng 3.1 có thể được thay bởi một ma trận tuần tự công nghệ $\{T_{ik}\}$ và một ma trận thời gian xử lý $\{p_{ik}\}$ như sau:

$$\{T_{ik}\} = \begin{vmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \\ 2 & 1 & 3 \end{vmatrix} \qquad \{p_{ik}\} = \begin{vmatrix} 4 & 4 & 4 \\ 3 & 4 & 5 \\ 4 & 3 & 2 \end{vmatrix}$$

Hình 3.2 là biểu đồ Grant biểu diễn một lời giải cho JSP 3 × 3 được cho trong bảng 3.1. Lịch biểu này không tích cực vì thao tác O_{31} có thể được bắt đầu xử lý sớm hơn mà không cần thay đổi thứ tự xử lý của bất kỳ thao tác nào trong lịch biểu.

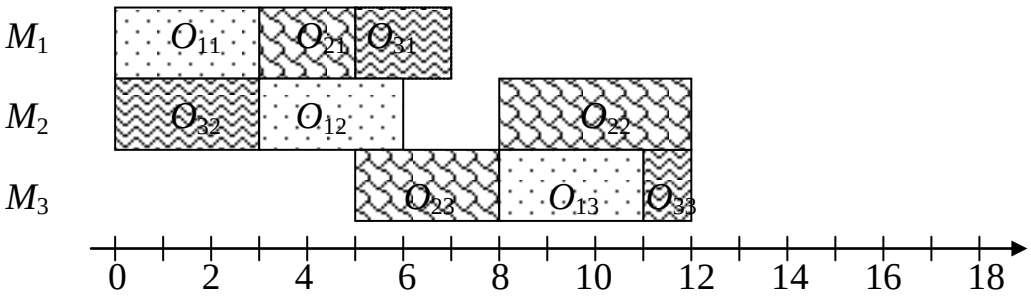


Hình 3.2 - Lịch biểu không tích cực



Hình 3.3 - Một lịch biểu bán tích cực

Lịch biểu trong hình 3.2 sẽ trở thành bán tích cực nếu thao tác O_{31} được bắt đầu tại thời điểm đơn vị thời gian 5 và các thao tác còn lại cũng tịnh tiến theo như trong hình 3.3.



Hình 3.4 - Một lịch biểu tích cực

Lịch biểu trong hình 3.3 sẽ trở thành lịch biểu tích cực nếu thực hiện một phép dịch trái đối với thao tác O_{12} theo chiều mũi tên (O_{12} được lập lịch ngay sau thao tác O_{32}) và các thao tác liên quan cũng được lập lịch lại như trong hình 3.4.

3.2. Thuật toán GT

Tập luật ưu tiên do Giffler và Thompson [28] đề nghị còn được gọi ngắn gọn là thuật toán GT. Thuật toán GT là một trong các công trình sớm nhất về các luật ưu tiên, thuật toán này rất quan trọng và được sử dụng rộng rãi trong lập lịch. Mặc dù đã được đề xuất nhiều năm nay, thuật toán GT vẫn được xem như là nền tảng cho các luật ưu tiên khác. Tầm quan trọng của nó xuất phát từ thực tế đó là nó sinh ra các lịch biểu tích cực. Trước khi trình bày thuật toán GT, luận án trình bày định nghĩa một số khái niệm được sử dụng trong thuật toán này.

1. Một thao tác *O* nếu không phải là thao tác được lập lịch đầu tiên trên một máy đều có 2 thao tác đi trước trực tiếp:

+ Thao tác đi trước cùng công việc được ký hiệu bởi $PJ(O)$ là thao tác được lập lịch ngay trước thao tác O trong tuần tự công nghệ.

+ Thao tác đi trước cùng máy được ký hiệu bởi $PM(O)$ là thao tác được lập lịch gần đây nhất trên máy đó, hay thao tác với thời gian hoàn thành lớn nhất trong số tất cả các thao tác đã được lập lịch ở trên cùng một máy với O.

2. Một thao tác O chưa được lập lịch được gọi là có thể được lập lịch khi cả thao tác đi trước cùng công việc và đi trước cùng máy (nếu chúng tồn tại) đều đã được lập lịch. Tập tất cả các thao tác có thể được lập lịch được ký hiệu là G.

3. Thời gian có thể bắt đầu sớm nhất của thao tác O được ký hiệu là $ES(O)$ và thời gian có thể hoàn thành sớm nhất của thao tác O được ký hiệu là $EC(O)$ được xác định như sau:

$$ES(O) = \max\{c(PJ(O)), c(PM(O))\} \text{ và } EC(O) = ES(O) + p(O).$$

Trong đó, $c(PJ(O))$ và $c(PM(O))$ là thời gian hoàn thành của hai thao tác đi trước trên cùng công việc và trên cùng máy của thao tác O.

4. Thao tác có thể hoàn thành sớm nhất O_{*j} trong G_j với máy M_j, là một thao tác mà thời gian có thể hoàn thành sớm nhất của nó $EC(O_{*j})$ là nhỏ nhất trong số các thao tác trong G_j:

$$EC(O_{*j}) = \min \{EC(O) / O \in G_j\}. \quad (1)$$

5. Cho một thao tác có thể hoàn thành sớm nhất O_{*j} và nếu có k - 1 thao tác đã được lập lịch ở trên máy M_j, một tập cạnh tranh $C[M_j, k]$ là một tập các thao tác ứng cử viên cho việc lập lịch tiếp theo ở trên máy M_j được định nghĩa như sau:

$$C[M_j, k] = \{O_{ij} \in G_j / ES(O_{ij}) < EC(O_{*j})\}. \quad (2)$$

Lưu ý rằng $O_{*j} \in C[M_j, k]$.

Mục tiêu của thuật toán GT là lập lịch cho các thao tác sao cho tránh được thời gian nghỉ của máy đủ dài cho phép một phép dịch trái có thể. Do đó mà tập $C[M_j, k] \subset G_j$ được duy trì. Chỉ cần thao tác tiếp theo được chọn từ tập cạnh tranh này, một thời gian nghỉ sẽ được giữ đủ ngắn và lịch biểu kết quả được đảm bảo là tích cực. Một lịch biểu tích cực có được bởi việc lặp lại thuật toán GT cho tới khi tất cả các thao tác trong quy trình được lập lịch.

- **Thuật toán GT áp dụng cho JSP để sinh ra các lịch biểu tích cực**

Một bài toán lập lịch job shop được cho bởi ma trận tuần tự công nghệ $\{T_{ik}\}$ và ma trận thời gian xử lý $\{p_{ik}\}$, đây là 2 ma trận dữ liệu vào của bài toán lập lịch job shop cần giải quyết. Một lịch biểu tích cực có thể được sinh ra bằng cách sử dụng thuật toán GT theo các bước sau:

1. Khởi tạo G là tập các thao tác đầu tiên trong tuần tự công nghệ của tất cả các công việc (cột đầu tiên của ma trận tuần tự công nghệ $\{T_{ik}\}$), tức là $G = \{O_{1T_{11}}, O_{2T_{21}},...,O_{nT_{n1}}\}$. Đối với mỗi thao tác $O \in G$, $ES(O) := 0$ và $EC(O) := p(O)$.

2. Tìm thao tác có thể hoàn thành sớm nhất $O_{*j} \in G$. Một tập con của G chứa các thao tác được xử lý ở trên máy M_j được ký hiệu là G_j.

3. Tính tập cạnh tranh $C[M_j, k] \subset G_j$, ở đây k - 1 là số các thao tác đã được lập lịch trên máy M_j.

4. Chọn ngẫu nhiên một thao tác trong $C[M_j, k]$, gọi thao tác được chọn là O_{i*j}.

5. Lập lịch cho O_{i*j} là thao tác thứ k trên máy M_j, tức là $S_{jk} := i*$, với thời gian bắt đầu và thời gian hoàn thành của nó là: $s(O_{i*j}) = ES(O_{i*j})$ và $c(O_{i*j}) = EC(O_{i*j})$.

6. Đối với tất cả các thao tác $O_{ij} \in G_j \setminus \{O_{i*j}\}$:

- Cập nhật $ES(O_{ij})$ như sau: $ES\ (O_{ij}) := \max\{ES(O_{ij}), EC(O_{i*j})\}$.

- Cập nhật $EC(O_{ij})$ như sau: $EC(O_{ij}) := ES\ (O_{ij}) + p(O_{ij})$.

7. Xoá O_{i*j} khỏi G, và bổ sung thêm thao tác O_{is} kế tiếp O_{i*j} trong tuần tự công nghệ của công việc J_i vào G nếu nó tồn tại. Tức là, nếu $j = T_{ik}$ và $k < m$, thì $s := T_{ik+1}$ và $G := (G\backslash\{O_{i*j}\} \cup \{O_{is}\}$. Tính $ES(O_{is})$ và $EC(O_{is})$ như sau:

- $ES\ (O_{is}) := \max\{EC(O_{i*j}), EC(PM(O_{is}))\}$.

- $EC(O_{is}) := ES\ (O_{is}) + p(O_{is})$.

8. Lặp lại bước 2 đến bước 7 tới khi tất cả các thao tác được lập lịch.

9. Ma trận lời giải ra $\{S_{jk}\}$ là lịch biểu tích cực thu được với tập thời gian bắt đầu là $\{s(O_{ij})\}$ và tập thời gian hoàn thành là $\{c(O_{ij})\}$. Ở đây $i = S_{jk}$.

3.3. Một thuật toán di truyền lai mới cho bài toán lập lịch job shop

Trong mục này luận án đề xuất một thuật toán di truyền lai mới cho JSP, thuật toán này có một số cải tiến mới sau đây:

1. Mã hoá các thao tác của một lịch biểu bởi các số tự nhiên: Cách mã hóa này tạo điều kiện thuận lợi cho việc thực thi các toán tử di truyền và đơn giản hóa trong cài đặt chương trình.

2. Sử dụng chiến lược ‘‘đột biến lại’’: Vì một trong những điểm yếu của GA là không phù hợp cho việc điều chỉnh các lời giải khi rất gần với lời giải tối ưu vì toán tử trao đổi chéo thường phá vỡ sự điều chỉnh này. Cải tiến này được đưa vào có tác dụng tinh chỉnh các lời giải hướng tới lời giải tối ưu, nó đặc biệt hữu ích khi một một cá thể cha tham gia đột biến mà cá thể này đã gần chạm tới lời giải tối ưu của bài toán.

3. Toán tử trao đổi chéo được thực hiện trên 3 cá thể cha: Cải tiến này có tác dụng tạo ra một cá thể con mang nhiều thuộc tính của các cá thể cha khác nhau, nhằm tăng cường sự khám phá không gian tìm kiếm.

3.3.1. Mã hoá lời giải

Giả sử JSP đã cho có n công việc được xử lý trên m máy. Số các thao tác của công việc thứ i được ký hiệu là $job[i]$ (không quá m với mọi i). Tổng số các thao tác cần được xử lý của tất cả các công việc là $L = \sum_{i=1}^{n} job[i]$. Chúng ta mã hoá các thao tác của J_1 từ 1 đến $job[1]$, của J_2 từ $job[1] + 1$ đến $job[1] + job[2]$,..., của J_n từ $job[1] + job[2] + ... + job[n\text{-}1] + 1$ đến L. Như vậy một lời giải là một hoán vị nào đó của dãy số tự nhiên $\{1, 2, 3, ..., L\}$ thoả mãn các ràng buộc của bài toán.

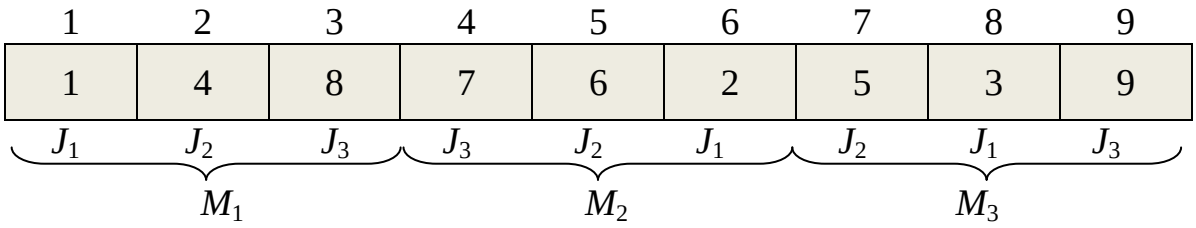
Ví dụ, với bài toán 3 công việc, 3 máy đã cho trong bảng 3.1. Các thao tác được mã hoá bằng các số tự nhiên như trong bảng 3.2.

Bảng 3.2 - Mã hoá các thao tác bằng số tự nhiên của JSP 3 × 3

Công việc	Mã hoá thao tác		
J_1	1	2	3
J_2	4	5	6
J_3	7	8	9

Giải thích: Theo $\{T_{jk}\}$, các thao tác đầu tiên của J_1, J_2 và thao tác thứ 2 của J_3 được xử lý trên máy 1. Vì vậy, mã của các thao tác trên M_1 là một hoán vị nào đó của tập thao tác $\{1, 4, 8\}$. Tương tự các thao tác trên M_2 là một hoán vị nào đó của tập thao tác $\{2, 6, 7\}$, trên M_3 là một hoán vị nào đó của tập thao tác $\{3, 5, 9\}$.

Một lời giải hợp lệ có dạng như được biểu diễn trog hình 3.5.



Hình 3.5 - Một lời giải hợp lệ cho JSP 3 × 3

Lời giải trong hình 3.5 cũng có thể được biểu diễn bởi một ma trận lời giải S_{jk}. Trong đó, $S_{jk} = i$, tức là thao tác thứ k trên máy M_j là của công việc J_i.

$$\{S_{jk}\} = \begin{vmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 2 & 1 & 3 \end{vmatrix}$$

3.3.2. Khởi tạo tập lời giải cho thế hệ ban đầu

Để sinh ra một tập lời giải cho thế hệ ban đầu $P(0)$, bao gồm các lịch biểu tích cực với JSP được cho bởi ma trận tuần tự công nghệ $\{T_{ik}\}$, và ma trận thời gian xử lý $\{p_{ik}\}$, chúng ta sử dụng thuật toán GT đã được trình bày trong mục 3.2. Sau khi khởi tạo tập lời giải ban đầu, chọn ra một cá thể có độ thích nghi cao nhất gọi là ″cá thể tinh hoa″. Cá thể này không tham gia vào các toán tử di truyền và sẽ được cập nhật sau mỗi thế hệ.

3.3.3. Xây dựng hàm thích nghi

Giả sử $P(0) = \{p_1, p_2, ..., p_n\}$. Chúng ta ký hiệu makespan của p_i là eval(p_i), khi đó makespan trung bình được tính theo công thức:

$$EvalP(0) = \frac{1}{n}\sum_{1}^{n} eval(p_i)$$

Hàm thích nghi của mỗi cá thể được xây dựng như sau:

fitness(p_i) = M - eval(p_i), trong đó M = 2Eval(P(0)), M là tham số được đưa vào để chuyển bài toán tìm min thành bài toán tìm max (sự chuyển đổi này nhằm tạo điều kiện thuận lợi cho các phần nghiên cứu tiếp theo của luận án).

3.3.4. Các toán tử di truyền

- **Toán tử đột biến**

Toán tử đột biến được tiến hành trên một cá thể cha theo các bước sau:

1. Chọn ngẫu nhiên một thao tác (ký hiệu là *ope*1) trong cá thể cha. Xác định máy thực hiện thao tác đó (ký hiệu là M_{ope1}) và vị trí của thao tác đó trong lời giải (ký hiệu là *pos*1).

2. Chọn ngẫu nhiên một thao tác (*ope*2) trong cá thể cha. Xác định máy thực hiện thao tác đó (M_{ope2}) và vị trí của thao tác đó (*pos*2).

3. Nếu $M_{ope1} = M_{ope2}$ thì tiến hành đột biến (hoán đổi vị trí của hai thao tác). Kết quả cho chúng ta cá thể con. Trong trường hợp $M_{ope1} \neq M_{ope2}$ thì cá thể cha được giữ nguyên.

4. Tính độ thích nghi của cá thể con, cá thể con chỉ được chấp nhận khi có độ thích nghi tốt hơn cá thể cha hoặc số lần đột biến lại vượt quá ngưỡng cho phép (theo qui định). Mỗi cá thể con thu được sau phép đột biến có thể xem như là một lân cận của cá thể cha.

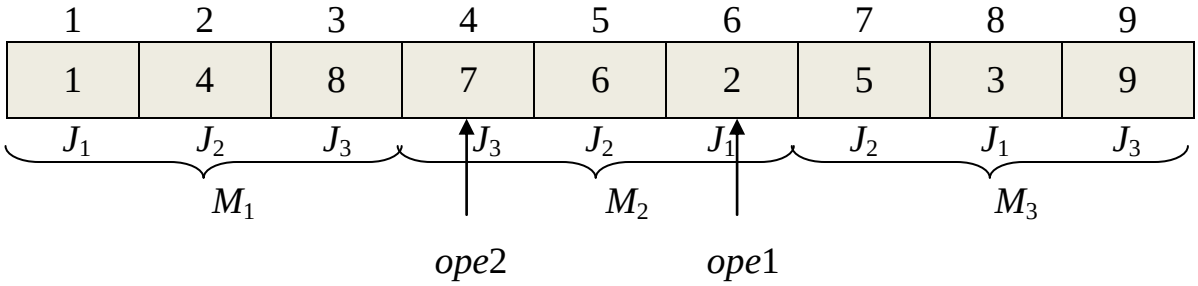
5. Nếu cá thể con sau đột biến có độ thích nghi tốt hơn cá thể cha thì nó sẽ được thay thế cho cá thể cha, ngược lại giữ nguyên cá thể cha.

Ví dụ, cá thể cha được chọn để đột biến như trong hình 3.6.

+ Chẳng hạn: *ope*1 = 2 → M_{ope1} = 2 và *pos*1 = 6.

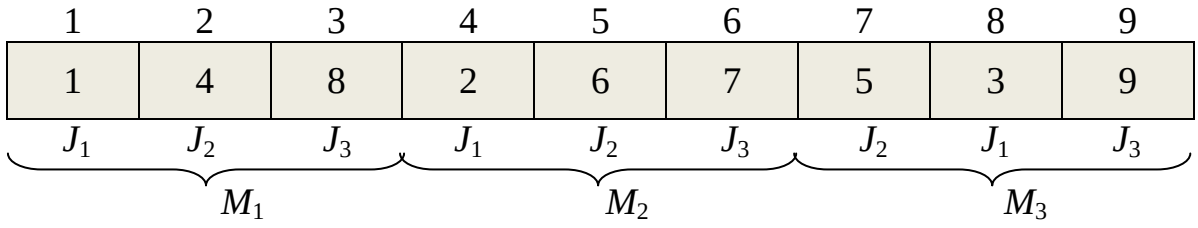
+ *ope*2 = 7 → M_{ope2} = 2 và *pos*2 = 4.

+ $M_{ope1} = M_{ope2}$ → hoán đổi các thao tác ở vị trí 4 và vị trí 6 cho nhau.



Hình 3.6 - Cá thể cha cho phép đột biến

Cá thể con sau khi đột biến được biểu diễn trong hình 3.7. Cá thể con này sẽ được kiểm tra độ thích nghi, nếu tốt hơn cá thể cha thì nó sẽ được chấp nhận, còn không phép đột biến được tiến hành lại cho tới khi gặp điều kiện kết thúc hoặc có cá thể con có độ thích nghi tốt hơn.



Hình 3.7 - Cá thể con thu được sau phép đột biến

- **Toán tử trao đổi chéo**

Toán tử trao đổi chéo được thực hiện trên 3 cá thể cha p_1, p_2 và p_3 được biểu diễn bởi các ma trận lời giải tương ứng $S^1 = \{S^1_{jk}\}$, $S^2 = \{S^2_{jk}\}$ và $S^3 = \{S^3_{jk}\}$. Các gien trong cá thể con $p = \{S_{jk}\}$ sẽ được tái kết hợp từ các gien trong 3 cá thể cha. Toán tử đổi chéo kết hợp đồng thời phép đổi chéo đồng nhất, thuật toán GT và được thực hiện trên 3 cá thể cha để tăng tính đa dạng của cá thể con. Do sử dụng thuật toán GT nên sau khi đổi chéo, cá thể con vẫn là một lịch biểu tích cực. Các bước của phép đổi chéo được mô tả như sau:

1. Khởi tạo G là tập các thao tác đầu tiên trong tuần tự công nghệ của tất cả các công việc (cột đầu tiên của ma trận $\{T_{ik}\}$), $G = \{ O_{1T_{11}}, O_{2T_{21}}, ..., O_{nT_{n1}} \}$. Đối với mỗi thao tác $O \in G$, $ES(O) := 0$ và $EC(O) := p(O)$.

2. Tìm thao tác hoàn thành sớm nhất $O_{*j} \in G$. Một tập con của G chứa các thao tác được xử lý ở trên máy M_j ký hiệu là G_j.

3. Xác định tập cạnh tranh $C[M_j, k] \subset G_j$, ở đây k - 1 là số các thao tác đã được lập lịch trên máy M_j.

4. Chọn một trong các cha $\{p_1, p_2, p_3\}$ tùy theo giá trị của ma trận H_{ji}, tức là $p := p_{H_{ij}}$ và $S^p = S^{H_{ij}}$. Đối với mỗi $O_{ij} \in C[M_j, k]$ tồn tại chỉ số l sao cho $S_{jl} = i$. Gọi l_m là chỉ số nhỏ nhất, tức là $l_m = \min \{l / S_{jl} = i \text{ và } O_{ij} \in C[M_j, k]\}$. Gọi $r := S_{jlm}$. $O_{rj} \in C[M_j, k]$ sẽ được chọn để lập lịch ở trong cá thể con p.

5. Lập lịch cho O_{rj} là thao tác thứ k trên máy M_j; tức là $S_{jk} := r$, với thời gian bắt đầu và thời gian hoàn thành của nó là $ES(O_{rj})$ và $EC(O_{rj})$ được tính như sau: $s(O_{rj}) = ES(O_{rj})$; $c(O_{rj}) = EC(O_{rj})$.

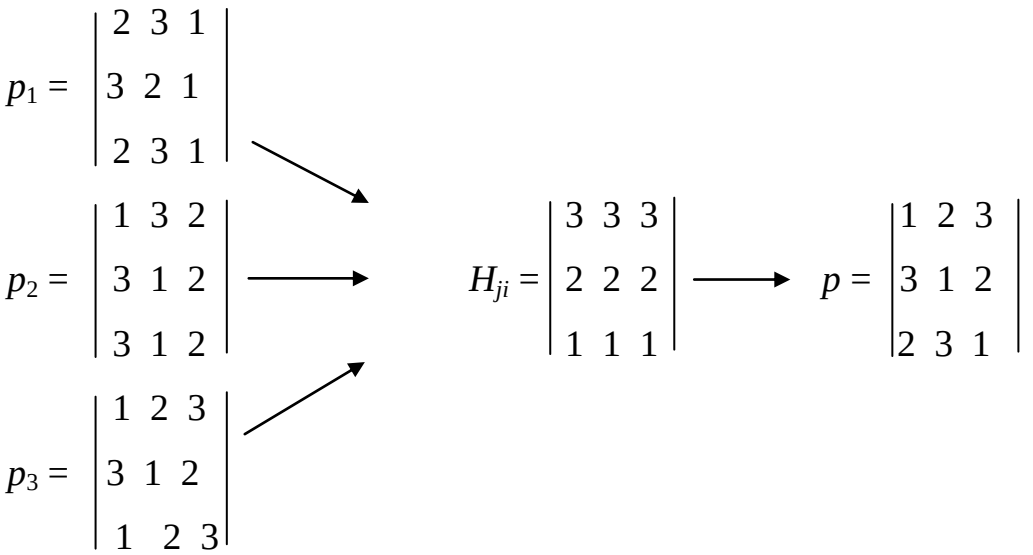
6. Đối với tất cả các thao tác $O_{ij} \in G_j \setminus \{O_{rj}\}$:

- Cập nhật $ES(O_{ij})$ như sau: $ES(O_{ij}) := \max\{ES(O_{ij}), EC(O_{rj})\}$.
- Cập nhật $EC(O_{ij})$ như sau: $EC(O_{ij}) := ES(O_{ij}) + p(O_{ij})$.

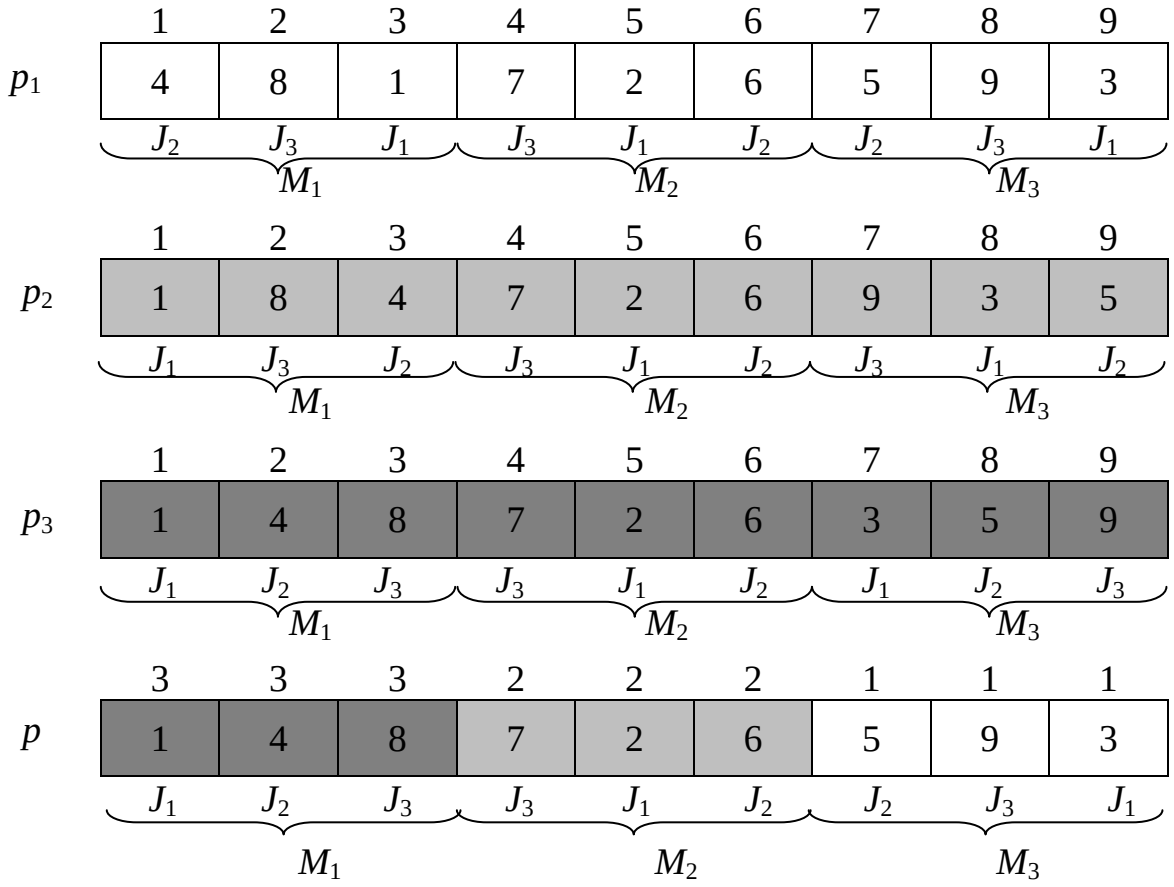
7. Xoá O_{rj} khỏi G (và do đó khỏi G_j), bổ sung thêm thao tác O_{rs} kế tiếp O_{rj} trong tuần tự công nghệ vào G nếu nó tồn tại. Tức là, nếu $j = T_{ik}$ và $k < m$, thì $s := T_{i,k+1}$ và $G := (G\setminus\{O_{rj}\} \cup \{O_{rs}\}$. Tính $ES(O_{rs})$ và $EC(O_{rs})$ như sau:

- $ES(O_{rs}) := \max\{EC(O_{rj}), EC(PM(O_{rs}))\}$.
- $EC(O_{rs}) := ES(O_{rs}) + p(O_{rs})$.

8. Lặp lại từ bước 2 đến bước 7 cho tới khi tất cả các thao tác được lập lịch trong cá thể con p.



Hình 3.8 - Trao đổi chéo dùng GT và thực hiện trên 3 cá thể cha



Hình 3.9 - Các cha tham gia đổi chéo và cá thể con sau đổi chéo

9. Cá thể con thu được chính ma trận lời giải ra $\{S_{jk}\}$ là một lịch biểu tích cực với tập thời gian bắt đầu và tập thời gian hoàn thành là $\{s(O_{ij})\}$ và $\{c(O_{ij})\}$. Ở đây $i = S_{jk}$.

Hình 3.8 minh họa về toán tử trao đổi chéo đồng nhất sử dụng thuật toán GT, được áp dụng cho ba cha p_1, p_2 và p_3 với một ma trận ngẫu nhiên H_{ji}. Con p là kết quả của phép trao đổi chéo.

Các cha tham gia trao đổi chéo và cá thể con sau phép trao đổi chéo được biểu diễn bằng hình vẽ như hình 3.9.

- **Toán tử chọn lọc**

Toán tử chọn lọc chọn ngẫu nhiên pop_size cá thể cho thế hệ $t + 1$ theo nguyên lý bánh xe sổ xố.

- **Toán tử sao chép**

Toán tử này thay thế cá thể có độ thích nghi cao nhất của thế hệ hiện tại làm cá thể tinh hoa (trong trường hợp độ thích nghi của cá thể này cao hơn độ thích nghi của cá thể tinh hoa).

3.3.5. Thuật toán tiến hóa

Thuật toán di truyền lai mới cho JSP được đặc tả như sau:

```
Procedure NHGA_JSP
  Begin
    t = 0
    Khởi tạo P(t)                    {hàm InitPopulation, phần phụ lục}
    Đánh giá P(t)
    Chọn cá thể tinh hoa
   While ( not điều kiện dừng ) do
    Begin
```

t = t + 1;

Thực hiện phép trao đổi chéo {hàm InitCrossOver3} {phần phụ lục}

Thực hiện phép đột biến {hàm Mutation, phần phụ lục}

Đánh giá độ thích nghi của mỗi cá thể

Thực hiện chọn lọc {hàm Select, phần phụ lục}

Xác định cá thể có độ thích nghi cao nhất

Thực hiện sao chép {hàm SelectMax, phần phụ lục}

End

End

3.3.6. Tính đúng đắn của thuật toán được đề nghị

Tính đúng đắn của thuật toán được giải thích ngắn gọn như sau:

1. Do sử dụng thuật toán GT để sinh ra các lịch biểu, cho nên mỗi cá thể con được sinh ra đều là một lịch tích cực.

2. Trong phép đột biến, cá thể tham gia đột biến được sửa đổi bằng cách thay đổi thứ tự sắp xếp trong một máy nào đó, sau đó thời gian bắt đầu và thời gian kết thúc của toàn lịch biểu được cập nhật lại nên cá thể con sau đột biến vẫn đảm bảo là một lịch biểu hợp lệ.

3. Phép trao đổi chéo sử dụng thuật toán GT để sinh ra các lịch biểu con, cho nên mỗi cá thể con được sinh ra đều là một lịch biểu hợp lệ và hơn nữa chúng vẫn còn là lịch biểu tích cực.

4. Vì thuật toán luôn duy trì lời giải tốt nhất (cá thể tinh hoa) trong quần thể và được cập nhật sau mỗi thế hệ nên thuật toán được chứng tỏ hội tụ tới tối ưu toàn cục. Kết quả này sẽ được thảo luận trong chương 4 của luận án.

3.4. Song song hóa thuật toán di truyền lai mới cho bài toán lập lịch job shop

Bài toán lập lịch job shop có độ phức tạp tính toán rất lớn, nhất là trong trường hợp nhiều máy và nhiều công việc. Với thuật toán tuần tự được đề xuất trong mục 3.3, thời gian chạy máy sẽ khá lâu. Trong mục này, luận án trình bày thuật toán song song cho bài toán lập lịch job shop nhằm mục đích giảm rút ngắn thời gian chạy máy.

3.4.1. Mô tả thuật toán

Trong giải thuật song song hóa này, luận án áp dụng hình thức song song dữ liệu, bằng cách chia dữ liệu thành nhiều phần, mỗi phần sẽ do một bộ xử lý thực thi. Như vậy, khi thao tác với dữ liệu thì mỗi bộ xử lý chỉ thao tác với phần dữ liệu mà nó quản lý. Cách quản lý như thế chính là phân nhỏ bài toán ban đầu thành các bài toán trên các đoạn dữ liệu khác nhau.

Giả sử chúng ta có S là số cá thể trong quần thể, N là số bộ xử lý chạy song song với nhau. Mô hình được sử dụng trong phương pháp này là mô hình Master - Slave, trong đó có một bộ xử lý làm master còn N-1 bộ xử lý còn lại làm slave. Các công việc mà master và các slave sẽ thực hiện được liệt kê trong bảng 3.3.

+ Trong hình thức song song này, tất cả các hoạt động như: Mã hóa lời giải, khởi tạo quần thể ban đầu, các toán tử di truyền đều được thực hiện giống như đối với thuật toán tuần tự đã trình bày trong mục 3.3.

+ Tất cả các tác vụ của giải thuật di truyền lai tuần tự: Khởi tạo quần thể, chọn lọc, trao đổi chéo, đột biến đều được đồng thời thực hiện trên N-1 bộ xử lý đóng vai trò slave.

+ Cấu trúc thuật toán tuần tự không bị phá vỡ cho nên chất lượng lời giải của giải thuật sau khi song song hóa được đảm bảo như giải thuật tuần tự.

+ Ngoài thời gian thực thi, trong trường hợp song song hóa còn cộng thêm chi phí truyền thông giữa Master và các Slave. Vì vậy chất lượng máy móc sẽ có ảnh hưởng đáng kể tới gian thực hiện thuật toán.

+ Các hàm và thủ tục trong chương trình cho giải thuật di truyền được song song hóa đa phần được thừa kế từ chương trình cho giải thuật di truyền tuần tự. Các thủ tục chọn lọc, lai ghép, đột biến, sao chép được thừa kế hoàn toàn từ giải thuật di truyền tuần tự.

Bảng 3.3 - Nhiệm vụ của Master và Slave

Master	Slave
- Khởi tạo môi trường để các tiến trình giao tiếp với nhau. - Truyền các tham số: cỡ quần thể, xác suất trao đổi chéo, xác suất đột biến, số thế hệ cho các Slave.	- Thực hiện thuật toán tuần tự NHGA_JSP.
- Nhận các kết quả từ Slave.	Xác định cá thể có độ thích nghi cao nhất, gửi kết quả về cho Master.
- Lựa chọn kết quả tốt nhất từ các kết quả nhận về từ các Slave. - Gửi trở lại cho các Slave làm cá thể tinh hoa.	

3.4.2. Thủ tục di truyền song song cho JSP

Procedure PGA_JSP
Begin
Master:
Mở kênh truyền thông và khởi tạo các tuyến đoạn
Gửi các tham số: cỡ quần thể, xác suất trao đổi chéo, xác suất đột biến, số thế hệ cho các Slave
Các Slave:
t = 0
Khởi tạo P(t) *{hàm InitPopulation}*
Đánh giá P(t)
Chọn cá thể tốt nhất và gửi về Master
Master:
Chọn cá thể tốt nhất trong các cá thể vừa nhận và gửi trở lại cho các Slave làm cá thể tinh hoa
While (not điều kiện dừng) do
Begin
t = t + 1;
Các Slave:
Thực hiện trao đổi chéo *{hàm InitCrossOver3}*
Thực hiện đột biến *{hàm Mutation}*
Đánh giá độ thích nghi của mỗi cá thể
Thực hiện chọn lọc *{hàm Select}*
Xác định độ thích nghi cao nhất
Thực hiện sao chép
Chọn cá thể tốt nhất gửi về Master
Master:
Chọn cá thể tốt nhất trong các cá thể vừa nhận và gửi trở lại cho các Slave làm cá thể tinh hoa
End
End

3.4.3. Cài đặt thuật toán

Chương trình được cài đặt và chạy trên hệ thống máy chủ đặt tại Trung tâm Khoa học Tính toán (The Center for Computational Science - CCS), Trường Đại học Sư phạm Hà Nội. Hệ thống có 7 máy (nodes) được cài đặt hệ điều hành linux debian và cấu hình thành một PC cluster với thư viện xử lý song song MPICH2 (MPI- Message Passing Interface, CH - là một thư viện có tên là Chameleon mà William Gropp sử dụng để phát triển MPICH, 2 là version) và bộ quản lý chương trình PBS TORQUE (PBS- Portable Batch System; TORQUE (Terascale Open-source Resource and QUEue manager) là một PBS của công ty Adaptive Computing Enterprises, inc. Hệ thống CCS có thể truy cập ở bất cứ đâu và bất cứ thời điểm nào với tên internet toàn cầu là: ccs1.hnue.edu.vn. Hệ thống đã cài đặt các trình biên dịch cho các ngôn ngữ lập trình thông dụng như *C*, *C*++, java, Fortran, matlab, …

Sau khi truy cập hệ thống, người sử dụng có thể viết (hoặc gửi chương trình từ máy tính cá nhân của mình lên), biên dịch và chạy chương trình trên hệ thống của Trung tâm. Có 2 loại chương trình: single-node program (chương trình đơn, chạy trên 1 bộ xử lý) và multi-node program (chương trình song song); và có 2 cách thực hiện chương trình trên hệ thống PC cluster của CCS: chạy trực tiếp trên nền hệ điều hành và chạy thông qua bộ quản lý chương trình PBS TORQUE.

Dựa vào phương pháp đề xuất ở trên, luận án đã cài đặt một chương trình sử dụng thư viện lập trình song song MPI (Message Passing Interface) với mã nguồn *C*++ và chạy thử nghiệm trên hệ thống máy chủ CCS đặt tại trung tâm khoa học tính toán với 7 máy mỗi máy có tốc xử lý 2.8 GHz. Dữ liệu vào cho chương trình thử nghiệm là các bài toán test do Muth & Thompson đề nghị.

3.5. Kết quả thử nghiệm

3.5.1. Kết quả thử nghiệm thuật toán tuần tự

Dựa vào thuật toán NHGA_JSP đề xuất trong mục 3.3, luận án đã cài đặt một chương trình chạy thử nghiệm trên máy PC với bộ vi xử lý có tốc độ 2.8 GHz, hệ điều hành Windows. Kết quả chạy thử nghiệm trên các bài toán test được đề xuất bởi S. Lawrence (1984), Trường Đại Học Quản trị công nghiệp, Đại học Carnegie-Mellon, Pittsburgh, Pennsylvania. Các bài toán test này được đề xuất để thử nghiệm các kỹ thuật lập lịch heuristic. Kết quả chạy thử nghiệm được thống kê trong bảng 3.4.

Bảng 3.4 - Kết quả chạy thử nghiệm trên các bài toán test của Lawrence

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
Bài toán	Số công việc	Số máy	Cỡ quần thể	p_c	p_m	Thời gian TB (s)	Kết quả chạy	Tối ưu của BT
LA01	10	5	100	0.8	0.1	100	666	666
LA02	10	5	200	0.8	0.1	120	655	655
LA03	10	5	100	0.8	0.1	150	597	597
LA04	10	5	100	0.8	0.1	200	590	590
LA05	10	5	100	0.8	0.1	250	593	593
LA06	15	5	200	0.8	0.1	250	926	926
LA07	15	5	200	0.8	0.1	80	890	890
LA08	15	5	200	0.8	0.1	20	863	863
LA09	15	5	200	0.8	0.1	250	951	951
LA10	15	5	200	0.8	0.1	250	958	958
LA11	20	5	200	0.8	0.1	150	1222	1222

LA12	20	5	200	0.8	0.1	100	1039	1039
LA13	20	5	200	0.8	0.1	150	1150	1150
LA14	20	5	200	0.8	0.1	150	1292	1292
LA15	20	5	200	0.8	0.1	150	1207	1207
LA16	10	10	300	0.8	0.1	950	945	945
LA17	10	10	300	0.8	0.1	950	794	794
LA18	10	10	300	0.8	0.1	950	848	848
LA19	10	10	300	0.8	0.1	950	842	842
LA20	10	10	300	0.8	0.1	950	907	907
LA21	15	10	400	0.8	0.1	2500	1055	?
LA22	15	10	300	0.8	0.1	2150	927	927
LA23	15	10	300	0.8	0.1	2150	1032	1032
LA24	15	10	400	0.8	0.1	2500	940	?
LA25	15	10	400	0.8	0.1	2500	978	?
LA26	20	10	300	0.8	0.1	2450	1218	1218
LA27	20	10	400	0.8	0.1	3250	1270	?
LA28	20	10	300	0.8	0.1	2450	1216	1216
LA29	20	10	400	0.8	0.1	2450	1190	?
LA30	20	10	300	0.8	0.1	2450	1355	1355
LA31	30	10	300	0.8	0.1	2600	1784	1784
LA32	30	10	300	0.8	0.1	2600	1850	1850
LA33	30	10	300	0.8	0.1	3000	1719	1719
LA34	30	10	300	0.8	0.1	3900	1721	1721
LA35	30	10	300	0.8	0.1	4100	1888	1888

LA36	15	15	300	0.8	0.1	3950	1275	1268
LA37	15	15	300	0.8	0.1	3950	1415	1397
LA38	15	15	400	0.8	0.1	4250	1210	?
LA39	15	15	300	0.8	0.1	3950	1240	1233
LA40	15	15	400	0.8	0.1	4250	1235	?

Bảng thống kê kết quả chạy thử nghiệm cho thấy đa số các bài toán test đều tìm được lời giải tối ưu thực sự trong khoảng thời gian chạy máy trung bình không dài (cột 7). Ở đây, các bài toán test đều được chạy với số lần lặp 100 lần và chạy thử 10 lần. Cột 8 là các kết quả chạy thuật toán do luận án đề xuất, cột 9 là kết quả tối ưu thực sự của bài toán. Các vị trí có dấu ? là ký hiệu cho biết các bài toán này cho tới nay chưa biết lời giải tối ưu thực sự của chúng.

Để chứng tỏ tính vượt trội của thuật toán mà luận án đề xuất, kết quả chạy thử nghiệm của thuật toán được so sánh với các kết quả chạy thử nghiệm các thuật toán GA-ACO, GA, ACO (Ant Colony Optimization) của các tác giả Andrea Rossi và Elena Boschi người Italy [5]. Các thuật toán này được đề xuất năm 2010, chương trình được chạy trên máy PC tốc độ 2800 MHz, hệ điều hành Windows. Thuật toán do luận án đề xuất cũng được cài đặt chạy trên máy PC tốc độ 2.8 GHz và hệ điều hành Windows. Bảng 3.5 so sánh kết quả thử nghiệm thuật toán do luận án đề xuất với các kết quả thử nghiệm các thuật toán của hai tác giả người Italy đề xuất năm 2010 trên một số bài toán test của Lawrence.

Trong bảng thống kê so sánh có hai phần: Phần kết quả tính toán và phần thời gian chạy máy trung bình cho kết quả tính toán. Trong các thuật toán được đề nghị của các tác giả người Italy, thuật toán GA-ACO là tốt nhất

cả về kết quả tính toán lẫn thời gian chạy máy. Thuật toán NHGA do luận án đề xuất được so sánh với thuật toán GA-ACO. Bảng so sánh cho thấy kết quả tính toán của NHGA tốt hơn của GA-ACO, đồng thời thời gian chạy máy cũng nhanh hơn.

Bảng 3.5 - So sánh kết quả chạy thử nghiệm

Bài toán test	Số công việc	Số máy	Kết quả tối ưu	Kết quả chạy các thuật toán				Thời gian chạy trung bình			
				GA-ACO	GA	ACO	NHGA	GA-ACO	GA	ACO	NHGA
LA01	10	5	666	**666**	675	669	**666**	**183**	143	171	**100**
LA02	10	5	655	**688**	712	693	**655**	**221**	125	322	**120**
LA03	10	5	597	**626**	644	642	**597**	**290**	125	497	**150**
LA04	10	5	590	**611**	628	625	**590**	**312**	139	313	**200**
LA07	15	5	890	**894**	939	908	**890**	**110**	92	71	**80**
LA08	15	5	863	**863**	872	865	**863**	**13**	42	63	**20**
LA15	20	5	1207	**1246**	1284	1249	**1207**	**360**	189	184	**150**

3.5.2. Kết quả thử nghiệm thuật toán song song

Bảng 3.6 là kết quả chạy thử nghiệm của hai thuật toán tuần tự (NHGA) và song song (PHGA). Dữ liệu thử nghiệm là các bài toán test chuẩn do Muth & Thompson đề nghị (*mt*06: 6 công việc × 6 máy; *mt*10: 10 công việc × 10 máy; *mt*20: 20 công việc × 5 máy). Đây là các bài toán test nổi tiếng về khó giải quyết và đã biết kết quả tối ưu.

Bảng 3.6 - Kết quả chạy thử nghiệm NHGA và PHGA trên các bài toán test do Muth & Thompson đề nghị

Bài toán test	Cỡ quần thể	Số thế hệ	p_c	p_m	Kế quả chạy	Kết quả tối ưu
*mt*06	100	200	0.9	0.1	55	55
*mt*10	5000	200	0.9	0.1	930	930
*mt*20	1000	200	0.9	0.1	1170	1165

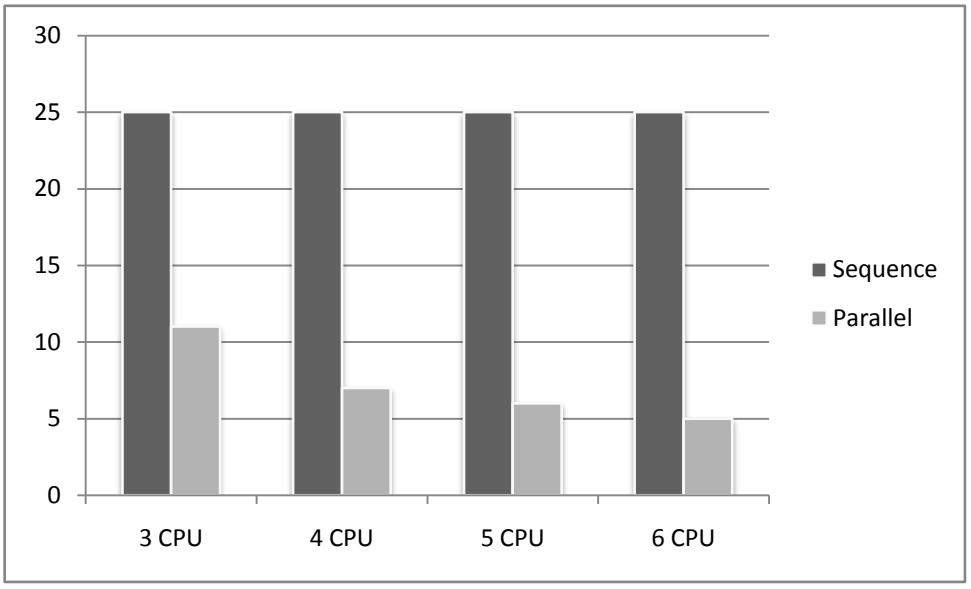
Bảng 3.7 - So sánh thời gian chạy thử nghiệm NHGA và PHGA

Bài toán test	NHGA			PHGA			
	Cỡ quần thể	Số lần chạy	Thời gian chạy TB (s)	Số lần chạy	Cỡ quần thể trên mỗi CPU	Số CPU (Slave)	Thời gian chạy TB (s)
*mt*06	120	100	25	100	40	3	11
					30	4	7
					24	5	6
					20	6	5
*mt*10	600	100	5500	100	200	3	2109
					150	4	1600
					120	5	1268
					100	6	980
*mt*20	900	100	8500	100	300	3	3100
					225	4	2350
					180	5	2188
					150	6	1700

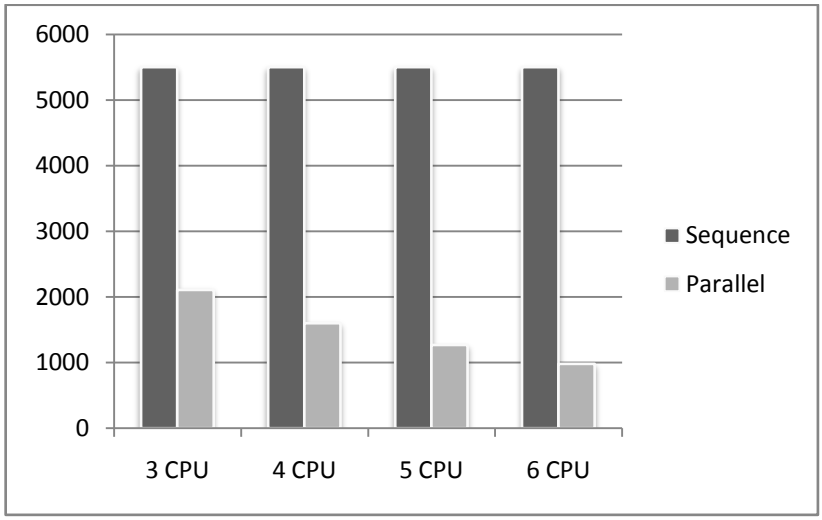
Bảng 3.7 so sánh thời gian chạy của hai thuật toán trên cùng một cấu hình máy (tốc độ CPU 2.8 GHz), cùng một hệ điều hành (Linux) và cùng một ngôn ngữ cài đặt (*C*++).

Qua bảng thống kê cho thấy thuật toán di truyền lai song song cho bài toán lập lịch job shop của luận án đề nghị có kết quả tìm ra lời giải tối ưu tương đương với thuật toán tuần tự. Tuy nhiên, tính ưu việt của nó là rút ngắn được thời gian chạy máy.

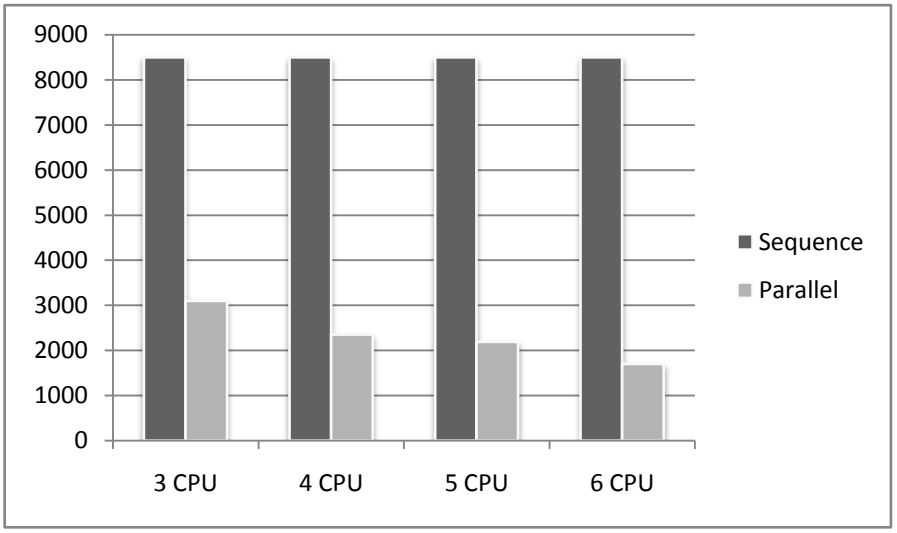
Hình 3.10, 3.11 và 3.12 là biểu đồ so sánh thời gian chạy máy của thuật toán NHGA và PHGA trên ba bài toán test *mt*06, *mt*10 và *mt*20 do Muth & Thompson đề nghị.



Hình 3.10 - Thời gian chạy máy của NHGA và PHGA đối với bài toán *mt*06



Hình 3.11 - Thời gian chạy máy của NHGA và PHGA đối với bài toán *mt*10



Hình 3.12 - Thời gian chạy máy của NHGA và PHGA đối với bài toán *mt*20

3.6. Kết luận

Trong chương này, một thuật toán di truyền lai mới cho JSP đã được đề nghị, thuật toán này sử dụng hợp lý các kết quả nghiên cứu của những người đi trước kết hợp với những đề xuất mới của luận án. Thuật toán đã được cài đặt và chạy thử nghiệm với dữ liệu vào là các bài toán test của Lawrence đề

nghị, đa số các bài toán test đều cho kết quả tối ưu thực sự. Việc cài đặt thuật toán này không mấy khó khăn do tính đơn giản của thuật toán.

Thuật toán do luận án đề nghị được so sánh với các thuật toán mới công bố gần đây của các tác giả người Italy. Kết quả thử nghiệm đã cho thấy những điểm mạnh của thuật toán do luận án đề nghị với các phương pháp khác đó là tính đơn giản, thời gian tính toán nhanh và tỷ lệ tìm được các lời giải tối ưu thực sự cao hơn.

Đê rút ngắn thời gian chạy máy, một thuật toán song song cho bài toán lập lịch job shop đã được đề xuất. Thuật toán đã được cài đặt và chạy thử nghiệm trên các bài toán test chuẩn của Muth & Thompson . Kết quả chạy thu được tương tự như thuật toán tuần tự nhưng thời gian tính toán được cải thiện nhiều lần.

CHƯƠNG 4. PHÂN TÍCH TÍNH HỘI TỤ CỦA THUẬT TOÁN DI TRUYỀN LAI MỚI CHO BÀI TOÁN LẬP LỊCH JOB SHOP

Các đề xuất mới cho JSP trong thời gian qua (kể cả trong và ngoài nước) thường chỉ được đánh giá thông qua thử nghiệm trên các bài toán test chuẩn. Vấn đề hội tụ của các thuật toán mới đề xuất cho JSP không được chứng minh trên cơ sở lý thuyết. Trong chương này, luận án phân tích các thuộc tính hội tụ của thuật toán đề nghị trong chương 3 bằng cách áp dụng các tính chất của xích Markov. Trên cơ sở phân tích xích Markov của thuật toán di truyền, luận án chứng tỏ rằng thuật toán được đề nghị trong chương 3 hội tụ tới tối ưu toàn cục.

4.1. Lý thuyết Xích Markov

Lý thuyết về xích Markov được đề xuất lần đầu tiên vào năm 1906 bởi nhà khoa học người Nga Andrei Andreyevich Markov. Kể từ đó cho tới nay, lý thuyết này được ứng dụng rất rộng rãi trong khoa học kỹ thuật hiện đại. Trước khi tìm hiểu về xích Markov, luận án nhắc lại một số định nghĩa về ma trận [65]:

Định nghĩa 1: *Một ma trận vuông A: n×n được gọi là không âm (A ≥ 0), nếu* $a_{ij} \geq 0$ *với* $\forall i,j \in \{1,\dots,n\}$*và dương (A > 0) nếu* $a_{ij} > 0$ *với* $\forall i,j \in \{1,\dots,n\}$.

Định nghĩa 2: *Một ma trận vuông A: n×n được gọi là:*

+ Chính quy, nếu $\exists k \in N$ *sao cho ma trận* A^k *là ma trận dương, như vậy một ma trận dương chắc chắn cũng là một ma trận chính quy.*

+ Ngẫu nhiên nếu $a_{ij} \geq 0$ *và* $\sum_{j=1}^{n} a_{ij} = 1, \forall i,j \in \{1,\dots,n\}$.

+ Giản ước được nếu A có thể được biến đổi về dạng $\begin{bmatrix} C & 0 \\ R & T \end{bmatrix}$.

(với C và T là các ma trận vuông) bằng cách hoán vị cùng một cách giữa các hàng và cột của ma trận.

Định nghĩa 3: *Một ma trận ngẫu nhiên A: n×n được gọi là thỏa mãn cột nếu nó có ít nhất một phần tử dương ở mỗi cột và được gọi là ổn định nếu tất cả các hàng của nó đều giống nhau.*

Từ đây ta có:

Bổ đề 1: [60]

Gọi C, M, S là các ma trận ngẫu nhiên, trong đó M là một ma trận dương và S là ma trận thỏa mãn cột. Khi đó tích C·M·S là một ma trận ngẫu nhiên dương.

4.1.1. Khái niệm xích Markov

Xét một hệ tiến trình tiến triển theo thời gian. Tại thời điểm $t = 0$, tiến trình có thể rơi vào một trong số các trạng thái của không gian trạng thái một cách ngẫu nhiên, gọi $X(t)$ là trạng thái của hệ tại thời điểm t. Như vậy, ứng với mỗi thời điểm t, $X(t)$ chính là một biến ngẫu nhiên mô tả trạng thái của tiến trình. Tiến trình $\{X(t)\}_{t \geq 0}$ được gọi là một quá trình ngẫu nhiên.

Tập hợp các trạng thái có thể có của tiến trình gọi là không gian trạng thái ký hiệu là $S = \{S_1, S_2, ...\}$.

Giả sử trước thời điểm s, tiến trình đã ở một trạng thái bất kỳ, còn tại thời điểm s, hệ đang ở trạng thái i, chúng ta muốn đánh giá xác suất để tại thời điểm t ($t > s$), hệ sẽ ở trạng thái j. Nếu xác suất đó chỉ phụ thuộc vào bộ bốn (s, i, t, j), tức là $p[X(t) = j/X(s) = i] = p(s, i, t, j), \forall s, i, t, j$ thì điều này có nghĩa là sự tiến triển của tiến trình trong tương lai chỉ phụ thuộc vào hiện tại và hoàn toàn độc lập với quá khứ (tính không nhớ). Đó chính là tính chất

Markov. Một quá trình ngẫu nhiên X(t) có tính chất Markov như trên được gọi là quá trình Markov.

Nếu không gian trạng thái S gồm một số hữu hạn hoặc vô hạn đếm được các trạng thái thì quá trình Markov X(t) được gọi là một xích Markov.

Xét một xích Markov, nếu xác suất chuyển trạng thái $p(s,i,t,j) = p(s+h,i,t+h,j), \forall i,j,s,t,h > 0$, thì ta nói rằng xích Markov trên là xích Markov thuần nhất theo thời gian. Như vậy một xích Markov thuần nhất thì xác suất chuyển từ trạng thái i sang trạng thái j không phụ thuộc vào thời điểm của tiến trình.

Giả sử tại thời điểm $t = n$, $X(n)$ cũng có thể nhận một trong N giá trị 1, 2, ..., N với các xác suất tương ứng là: $\pi_1^{(n)}, \pi_2^{(n)}, \dots, \pi_N^{(n)}$ (với $\pi_1^{(n)} + \pi_2^{(n)} \dots + \pi_N^{(n)} = 1$), thì véc tơ $\Pi^{(n)} = \left[\pi_1^{(n)}, \pi_2^{(n)}, \dots, \pi_N^{(n)}\right]$ được gọi là véc tơ phân phối xác suất tại thời điểm $t = n$.

Với $t = 0$, ta có véc tơ phân phối xác suất khởi tạo là:

$$\Pi^{(0)} = \left[\pi_1^{(0)}, \pi_2^{(0)}, \dots, \pi_N^{(0)}\right].$$

Ma trận: $P = \left[p_{ij}\right]_{N \times N}$, (với $p_{ij} = p(t,i,t+1,j) = p[X(t+1) = j / X(t) = i] \forall t$ là xác suất chuyển trạng thái từ vị trí i sang vị trí j sau một bước, $\forall i = 1,2,\dots,N$ và $\forall j = 1,2,\dots,N$) được gọi là ma trận xác suất chuyển trạng thái hay ma trận chuyển sau một bước. Vì chắc chắn sau một bước tiến trình sẽ chuyển từ trạng thái i sang một trạng thái j bất kỳ với xác suất $p_{ij} \geq 0$, nên ta có:

$$\sum_{j=1}^{N} p_{ij} = 1.$$

Như vậy, ma trận chuyển trạng thái P là một ma trận ngẫu nhiên.

Tương tự ta có ma trận $P^{(n)} = \left[p_{ij}^{(n)}\right]$là ma trận chuyển sau *n* bước.

4.1.2. Các tính chất của Xích Markov

Gọi $p_{ij}^{(2)}$ là xác suất chuyển từ trạng thái *i* sang trạng thái *j* của tiến trình sau 2 bước, giả sử không gian trạng thái *S* có *r* phần tử ta có $p_{ij}^{(2)}$ được tính như sau:

$$p_{ij}^{(2)} = \sum_{k=1}^{r} p_{ik}^{(1)} \cdot p_{kj}^{(1)}.$$

Từ đó ta có ma trận chuyển trạng thái sau 2 bước $P^{(2)}$ có giá trị:

$$P^{(2)} = \left[p_{ij}^{(2)}\right] = P^2.$$

Kéo theo đó, một cách tổng quát, ta có:

Định lý 1: ([17], trang 409)

Gọi P là ma trận chuyển trạng thái của một xích Markov, phần tử p_{ij}^{n} của ma trận P^n là xác suất chuyển từ trạng thái i sang trạng thái j sau n bước, hay ma trận chuyển trạng thái sau n bước của một xích Markov là lũy thừa bậc n của ma trận chuyển trạng thái một bước của nó:

$$P^{(n)} = \left[p_{ij}^{(n)}\right] = P^n.$$

Với $\Pi^{(n)}$là véc tơ phân phối xác suất tại thời điểm *n* ta có:

$$\Pi^{(n)} = \Pi^{(0)} \cdot P^{(n)}.$$

Từ đó ta suy ra được:

$$\Pi^{(n)} = \Pi^{(0)} \cdot P^{n}.$$

Như vậy xác suất phân bố trạng thái của tiến trình mang tính chất Markov chỉ phụ thuộc vào cặp (Π^0, P).

Có 2 loại xích Markov tiêu biểu là:

- **Xích Markov hấp thụ** (Absorbing Markov Chain): Một trạng thái i được gọi là hấp thụ nếu như khi đạt đến trạng thái đó, ta không thể chuyển sang trạng thái khác ($p_{ii} = 1$). Xích Markov hấp thụ là xích Markov chứa ít nhất một trạng thái hấp thụ và từ bất cứ trạng thái không hấp thụ nào khác ta đều có thể đến được trạng thái hấp thụ (không nhất thiết phải trong một bước).

- **Xích Markov Ergodic** (Ergodic Markov Chain): Phần sau đây sẽ tìm hiểu chi tiết hơn về xích Markov Ergodic, tính chất của xích Markov Ergodic có vai trò rất quan trọng trong việc khảo sát tính hội tụ của thuật toán di truyền.

4.2. Xích Markov Ergodic

- **Khái niệm**

Một xích Markov được gọi là xích Markov Ergodic nếu từ một trạng thái gốc bất kỳ, ta có thể di chuyển đến mọi trạng thái khác trong không gian trạng thái (không nhất thiết phải sau 1 bước). Như vậy, một xích Markov có ma trận chuyển là ma trận chính quy (primitive) là xích Markov Ergodic.

- **Tính chất của Xích Markov Ergodic**

Định lý 2: ([36], trang 123]

Gọi P là ma trận ngẫu nhiên chính quy. Khi đó P^k hội tụ khi $k \to \infty$ tới một ma trận ổn định:

$$P^\infty = 1'\Pi^\infty.$$

Trong đó 1' là ma trận cột chứa toàn phần tử 1 và $\Pi^{\infty} = \Pi^{(0)} \cdot lim_{k\to\infty} P^k = \Pi^{(0)} \cdot P^{\infty}$ *là vector phân phối xác suất ở thời gian vô cùng gồm các phần tử có giá trị dương và không phụ thuộc vào giá trị của phân phối xác suất khởi tạo* $\Pi^{(0)}$.

Như vậy, từ định lý 2 ta suy ra rằng phân phối xác suất trạng thái của một xích Markov Ergodic sẽ ổn định khi thời gian tiến tới vô cùng và không phụ thuộc vào phân phối xác suất khởi tạo.

Đây là tính chất quan trọng nhất của xích Markov Ergodic được sử dụng để chứng minh tính chất hội tụ của thuật toán di truyền.

Ở đây ta có thêm một định lý quan trọng nữa.

Định lý 3: ([36], trang 126]

Gọi $P: n \times n$ *là một ma trận ngẫu nhiên giản ước được, trong đó* $C: m \times m$ *là một ma trận ngẫu nhiên chính quy và* $R, T \neq 0$ *ta có:*

$$P^{\infty} = limP^k = \lim_{k\to\infty} \begin{pmatrix} C^k & 0 \\ \sum_{i=0}^{k-1} T^i R C^{k-i} & T^k \end{pmatrix} = \begin{pmatrix} C^{\infty} & 0 \\ R^{\infty} & 0 \end{pmatrix}$$

là một ma trận ổn định với $P^{\infty} = 1'\Pi^{\infty}$ *và* $\Pi^{\infty} = \Pi^{(0)} \cdot P^{\infty}$ *là giá trị véc tơ hàng ổn định của* P^{∞} *không phụ thuộc vào giá trị* $\Pi^{(0)}$ *và thỏa mãn:*

$\pi_i^{\infty} > 0$ *với* $1 \le i \le m$ *và* $\pi_i^{\infty} = 0$ *với* $m < i \le n$.

4.3. Phân tích tính hội tụ của thuật toán di truyền lai tuần tự cho bài toán lập lịch job shop

4.3.1. Phân tích tính hội tụ của thuật toán di truyền truyền thống

Trong mục này luận án sẽ phân tích tính hội tụ của thuật toán di truyền truyền thống (không có cá thể tinh hoa) để làm cơ sở cho việc phân tích tính

hội tụ của thuật toán đề nghị trong chương 3. Bài toán lập lịch job shop được giả sử với *n* công việc được xử lý trên *m* máy với một tuần tự công nghệ xác định. Mỗi lời giải coi là một cá thể gồm *m* máy, mỗi máy đều xử lý *n* công việc, mỗi phần công việc được xử lý trên mỗi máy là một gen (một thao tác). Gọi *N* là số cá thể của quần thể. Khi đó, mỗi một trạng thái *i* của quần thể có thể coi là một dãy mã hóa theo số tự nhiên có dài *n·m·N*, trong đó phép chiếu $\pi_k(i)$ cho ta cá thể thứ *k* trong quần thể. Không gian tìm kiếm có số các trạng thái là $(n!)^{m \cdot N}$.

Định nghĩa 4: *Gọi* $Z_t = max\left\{f\left(\pi_k^{(t)}(i)\right) / k = 1, \dots, N\right\}$ *là giá trị của biến ngẫu nhiên đại diện cho phần tử có độ thích nghi cao nhất của quần thể ở trạng thái i và bước t. Một thuật toán di truyền được gọi là hội tụ tới tối ưu toàn cục khi và chỉ khi:*

$$\lim_{t \to \infty} P\{Z_t = f^*\} = 1$$

trong đó f^* *là giá trị tối ưu toàn cục của bài toán.*

Như vậy, chúng ta cần phân tích là khi thời gian tiến tới vô cùng, thuật toán di truyền có chắc chắn tìm được nghiệm tối ưu của bài toán hay không.

- **Thuật toán tiến hoá truyền thống**

```
Begin
    t = 0
    Khởi tạo P(t)
    Đánh giá P(t)
        Repeat
            t = t + 1
            Thực hiện lai ghép
            Thực hiện đột biến
            Xác định độ thích nghi của mỗi cá thể
            Thực hiện chọn lọc
```

until một số tiêu chuẩn dừng được thỏa mãn

End

Quá trình tiến hóa xảy ra trong vòng lặp với ba toán tử: Lai ghép, đột biến và chọn lọc. Sau đây chúng ta sẽ xây dựng và đánh giá các ma trận xác suất chuyển trạng thái $P(t)$ sau mỗi toán tử.

- **Ma trận xác suất chuyển trạng thái của quần thể gây ra bởi toán tử lai ghép**

Xét toán tử lai ghép, gọi C là ma trận chuyển trạng thái của quần thể gây ra bởi toán tử lai ghép, ta có:

$$C = \begin{pmatrix} c_{11} & \cdots & c_{1,(n!)^{m\cdot N}} \\ \vdots & \ddots & \vdots \\ c_{(n!)^{m\cdot N},1} & \cdots & c_{(n!)^{m\cdot N},(n!)^{m\cdot N}} \end{pmatrix}$$

Với c_{ij} là xác suất của quần thể chuyển từ trạng thái i sang trạng thái j với xác suất lai p_c. Ta thấy với đầu vào của thuật toán tiến hóa truyền thống, xác suất lai ghép là $p_c \in (0,1)$, quá trình lai ghép diễn ra bằng việc chọn bất kỳ một cá thể cha mẹ ở trạng thái hiện thời i với xác suất lai p_c, tiến hành cho lai ghép 3 cá thể theo luật GT hoàn toàn ngẫu nhiên. Sau quá trình lai ghép, quần thể có thể ở trạng thái j bất kỳ với xác suất c_{ij} và

$$\sum_{j=1}^{(n!)^{m\cdot N}} c_{ij} = 1,$$

xác suất c_{ij} này không phụ thuộc vào việc quần thể đang ở thế hệ thứ bao nhiêu, đang ở thời điểm nào, mà chỉ phụ thuộc vào xác suất lai ghép $P_c \in (0,1)$ cố định theo đầu vào của thuật giải.

Như vậy, ma trận chuyển trạng thái sinh ra bởi phép lai ghép C của quần thể là một ma trận ngẫu nhiên và cố định không phụ thuộc vào trạng thái hiện thời cũng như thời điểm của quần thể.

- **Ma trận xác suất chuyển trạng thái của quần thể gây ra bởi toán tử đột biến**

Xét toán tử đột biến, gọi *M* là ma trận chuyển trạng thái của quần thể gây ra bởi toán tử đột biến, ta có:

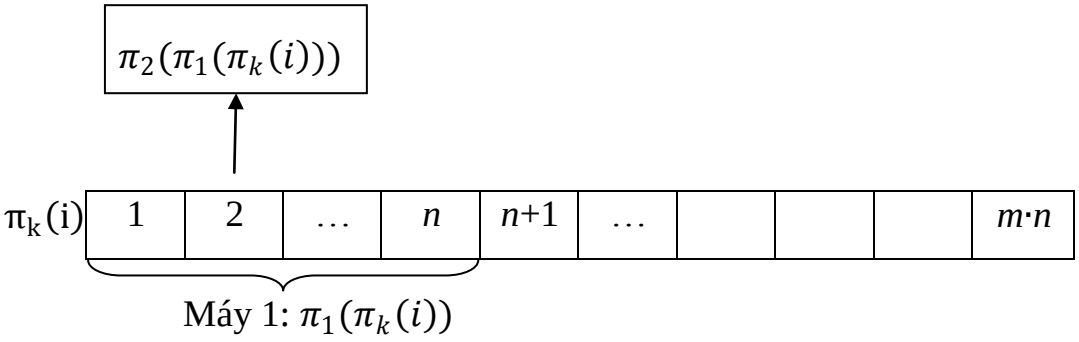
$$M = \begin{pmatrix} m_{11} & \dots & m_{1,(n!)^{m\cdot N}} \\ \vdots & \ddots & \vdots \\ m_{(n!)^{m\cdot N},1} & \dots & m_{(n!)^{m\cdot N},(n!)^{m\cdot N}} \end{pmatrix}$$

Với m_{ij} là xác suất quần thể chuyển từ trạng thái *i* sang trạng thái *j* sau toán tử đột biến với xác suất lai p_m. Tất nhiên sau toán tử đột biến, quần thể phải ở một trong số các trạng thái bất kỳ thuộc vào không gian trạng thái, nên ta có:

$$\sum_{j=1}^{(n!)^{m\cdot N}} m_{ij} = 1.$$

Như vậy, ma trận *M* cũng là một ma trận ngẫu nhiên.

Gọi *i* là trạng thái tại thời điểm *t*, gọi $\pi_k(i)$ là cá thể thứ *k* trong quần thể gồm *N* cá thể, $\pi_h(\pi_k(i))$ là máy thứ *h* trong cá thể thứ *k*, $\pi_l(\pi_h(\pi_k(i)))$ là gien ở vị trí *l*, được mô tả trong hình 4.1.



Hình 4.1 - Gien ở vị trí thứ 2 trạng thái *i* của quần thể

Thuật toán đột biến cho $\pi_k(i)$ được mô tả vắn tắt như sau:

1. Gieo xác suất $p_m > 0$ (rất nhỏ) là xác suất để $\pi_k(i)$ xảy ra đột biến. Vậy $1 - p_m \approx 1$ là xác suất để $\pi_k(i)$ không xảy ra đột biến.

2. Khi $\pi_k(i)$ đã xảy ra đột biến, gây đột biến ở $\pi_k(i)$ như sau:

Duyệt các máy h = 1,.., m, tại mỗi máy:

+ Gieo xác suất $p \approx 1$ thì giữ nguyên máy đó.

+ Ngược lại, thay máy h bởi một hoán vị bất kỳ khác đồng nhất của nó với khả năng như nhau. Vậy xác suất của mỗi sự thay đổi này là:

$$\frac{1-p}{n!-1}.$$

Với thuật toán trên, một cá thể có thể đột biến thành cá thể bất kỳ trong không gian tìm kiếm với xác suất dương, xác suất này chỉ phụ thuộc vào p. Gọi m_{ij} là xác suất chuyển của quần thể từ trạng thái i sang trạng thái j thông qua đột biến, thì $m_{ij} > 0, \forall i, j$. Xác suất m_{ij} này chỉ phụ thuộc vào p (không tính các hằng số n đã cho), và có thể tính cụ thể được như sau:

Xét hai trạng thái i, j bất kỳ, nếu i, j có K cá thể giống nhau và N - K cá thể khác nhau thì:

+ Xác suất để K cá thể giống nhau (không đột biến) là $(1 - p_m)^K$.

+ Với $N - K$ đôi cá thể khác nhau ở cùng một vị trí, $\pi_{k_t}(i)$ và $\pi_{k_t}(j), t = 1, \dots, N - K$:

Gọi L_{k_t} là số máy có các gen giống nhau giữa $\pi_{k_t}(i)$ và $\pi_{k_t}(j)$, còn $m - L_{k_t}$ là số máy có các gen khác nhau trong hai cá thể này.

- Xác suất để L_{k_t} máy giống nhau là $p^{L_{k_t}}$.

- Xác suất để $m - L_{k_t}$ máy còn lại khác nhau là: $\left(\frac{1-p}{n!-1}\right)^{m-L_{k_t}}$.

Vậy: $m_{ij} = (1-p_m)^K p_m^{N-K} \cdot \prod_{t=1}^{N-K}\left(p^{L_{k_t}}\left(\frac{1-p}{n!-1}\right)^{m-L_{k_t}}\right) > 0$

Vì xác suất $p_m \in (0,1)$ cố định, do đó m_{ij} là cố định dương và không phụ thuộc vào trạng thái hiện thời cũng như thời điểm của quần thể.

Như vậy, ma trận chuyển trạng thái *M* của quần thể gây ra bởi toán tử đột biến là một ma trận ngẫu nhiên dương không phụ thuộc vào thời điểm cũng như trạng thái của quần thể.

- **Ma trận xác suất chuyển trạng thái của quần thể gây ra bởi toán tử chọn lọc**

Toán tử thứ 3 sẽ được thực hiện để sinh ra một quần thể mới từ quần thể cũ là toán tử chọn lọc. Gọi *S* là ma trận chuyển trạng thái của quần thể gây ra bởi toán tử chọn lọc, ta có:

$$S = \begin{pmatrix} s_{11} & \cdots & s_{1,(n!)^{m\cdot N}} \\ \vdots & \ddots & \vdots \\ s_{(n!)^{m\cdot N},1} & \cdots & s_{(n!)^{m\cdot N},(n!)^{m\cdot N}} \end{pmatrix}$$

Với s_{ij} là xác suất quần thể chuyển từ trạng thái *i* sang trạng thái *j* gây ra bởi toán tử chọn lọc, ta cũng có:

$$\sum_{j=1}^{(n!)^{m\cdot N}} s_{ij} = 1.$$

Như vậy, ma trận *S* cũng là một ma trận ngẫu nhiên.

Xác suất để một cá thể $\pi_k(i)$ trong quần thể hiện thời ở trạng thái *i* được giữ lại trong quần thể mới là:

$$p_k = f(\pi_k(i)) / \sum_{h=1}^{N} f\big(\pi_h(i)\big).$$

Như vậy, xác suất biến đổi từ trạng thái *i* sang trạng thái *j* của quần thể gây ra bởi toán tử chọn lọc chỉ phụ thuộc vào hàm *f* mà không phụ thuộc vào trạng thái hiện thời cũng như thời điểm hiện tại của quần thể.

Từ đó, xác suất để quần thể hiện thời giữ nguyên trạng thái *i* sau phép chọn lọc sẽ là:

$$s_{ij} = \frac{\prod_{k=1}^{N} f(\pi_k(i))}{(\sum_{h=1}^{N} f(\pi_h(i))^N}.$$

Vì hàm *f* là một hàm luôn dương ta suy ra: $s_{ij} > 0$.

Ma trận chuyển trạng thái *S* luôn có ít nhất một phần tử trên một cột là dương nên ma trận *S* là ma trận thỏa mãn cột.

Như vậy, ma trận chuyển trạng thái *S* của quần thể gây ra bởi toán tử chọn lọc là một ma trận ngẫu nhiên, thỏa mãn cột và không phụ thuộc vào trạng thái hiện thời cũng như thời điểm của quần thể.

Tóm lại, quá trình sinh ra một thế hệ mới từ thế hệ cũ theo giải thuật tiến hóa truyền thống sẽ được thực hiện thông qua 3 toán tử lai ghép, đột biến và chọn lọc với các ma trận chuyển trạng thái *C, M* và *S* tương ứng. Gọi *P* là ma trận chuyển trạng thái tổng hợp từ thế hệ *t* sang thế hệ *t* + 1, ta có:

$$P = \begin{pmatrix} p_{11} & \cdots & p_{1,(n!)^{m\cdot N}} \\ \vdots & \ddots & \vdots \\ p_{(n!)^{m\cdot N},1} & \cdots & p_{(n!)^{m\cdot N},(n!)^{m\cdot N}} \end{pmatrix} = C \cdot M \cdot S$$

Trong đó p_{ij} là xác suất chuyển trạng thái của quần thể sau một thế hệ, và *C, M, S* tương ứng là các ma trận chuyển trạng thái của quần thể gây ra bởi các toán tử lai ghép, đột biến, chọn lọc.

Vì các ma trận *C*, *M* và *S* là các ma trận ngẫu nhiên không phụ thuộc vào thời điểm cũng như trạng thái hiện thời của quần thể nên ma trận *P* cùng với các phần tử p_{ij} cũng không phụ thuộc vào trạng thái hiện thời cũng như thời điểm của quần thể.

Lại có ma trận *C*, *M* và *S* là các ma trận ngẫu nhiên, *M* là ma trận dương và *S* là ma trận thỏa mãn cột, từ bổ đề 1 ta suy ra ma trận xác suất chuyển trạng thái tổng hợp *P* là một ma trận ngẫu nhiên dương.

Như vậy, thuật toán tiến hóa truyền thống có thể được mô tả thông qua một xích Markov với trạng thái của quần thể nằm trong không gian trạng thái và ma trận xác suất chuyển trạng thái *P* dương. Ta suy ra thuật toán này chính là một xích Markov Ergodic.

▪ **Chúng ta sẽ chứng tỏ rằng thuật toán di truyền truyền thống không hội tụ tới tối ưu toàn cục**

Thật vậy, theo định lý 2 ta có phân bố xác suất của các trạng thái của quần thể khi thời gian tiến tới vô cùng:

$\Pi^{\infty} = \left[\pi_1^{\infty}, \pi_2^{\infty}, \dots, \pi_{(n!)^{m \cdot N}}^{\infty}\right]$ là vector hàng với các phần tử dương.

Xét một trạng thái *j* bất kỳ không chứa nghiệm tối ưu, xác suất quần thể đạt tới trạng thái *j* khi $t \to \infty$ là $\pi_j^{\infty} > 0$.

gọi p_d là xác suất các trạng thái của quần thể không chứa cá thể mang nghiệm tối ưu khi thời gian $t \to \infty$, ta suy ra $p_d \geq \pi_j^{\infty} > 0$. Theo định nghĩa ta có:

$$\lim_{t \to \infty} P\{Z_t = f^*\} = 1 - p_{\mathrm{d}} < 1.$$

Như vậy, thuật toán di truyền truyền thống không hội tụ tới tối ưu toàn cục.

4.3.2. Phân tích tính hội tụ của thuật toán di truyền với cá thể tinh hoa và toán tử sao chép

- **Thuật toán di truyền với cá thể tinh hoa và toán tử sao chép**

Để tiện theo dõi, ở đây trình bày lại thuật toán lai cho JSP được đề xuất trong chương 3:

Begin

t = 0

Khởi tạo P(t)

Đánh giá P(t)

Chọn cá thể tinh hoa

// cá thể có độ thích nghi tốt nhất //

// cá thể này không tham gia vào các toán tử di truyền //

While (not điều kiện dừng) do

Begin

Thực hiện phép trao đổi chéo

Thực hiện phép đột biến

Đánh giá độ thích nghi của mỗi cá thể

Thực hiện chọn lọc

Xác định cá thể có độ thích nghi cao nhất

Thực hiện sao chép

End

End

Với sự xuất hiện của cá thể tinh hoa, cỡ quần thể sẽ là *N+1*, như vậy không gian trạng thái của quần thể sẽ không còn là $(n!)^{m \cdot N}$ nữa, mà sẽ là $(n!)^{m(N+1)}$. Ta đặt vị trí của siêu cá thể ở đầu của chuỗi mã hóa chiều dài $(N+1)m \cdot n$ và ta có thể truy xuất đến siêu cá thể đó thông qua phép chiếu

$\pi_0(i)$ khi quần thể ở trạng thái i. Với mỗi trạng thái của quần thể, chúng ta có 1 cá thể tinh hoa tương ứng đứng ở đầu, ta sắp xếp không gian các trạng thái theo thứ tự giảm dần độ thích nghi của cá thể tinh hoa (tức là $i < j \rightarrow f(\pi_0\,(i)) > f(\pi_0\,(j))$.

Khi đó ma trận chuyển trạng thái gây ra bởi các toán tử lai ghép, đột biến và chọn lọc sẽ đều có kích thước là $(n!)^{m(N+1)} \times (n!)^{m(N+1)}$. Gọi C^+, M^+, S^+ lần lượt là ma trận chuyển trạng thái gây ra bởi các toán tử lai ghép, đột biến và chọn lọc lên quần thể có chứa cá thể tinh hoa, vì các siêu cá thể không tham gia vào quá trình tiến hóa nên các ma trận C^+, M^+, S^+sẽ có dạng:

$$C^+ = \begin{pmatrix} C & & \\ & \ddots & \\ & & C \end{pmatrix}, M^+ = \begin{pmatrix} M & & \\ & \ddots & \\ & & M \end{pmatrix}, S^+ = \begin{pmatrix} S & & \\ & \ddots & \\ & & S \end{pmatrix}$$

Với mỗi đường chéo bao gồm $(n!)^m$ ma trận *C*, *M*, *S* đã được xét trong giải thuật tiến hóa chưa có cá thể tinh hoa, từ đó ta có:

$$C^+ \cdot M^+ \cdot S^+ = \begin{pmatrix} C \cdot M \cdot S & & \\ & \ddots & \\ & & C \cdot M \cdot S \end{pmatrix} = \begin{pmatrix} P & & \\ & \ddots & \\ & & P \end{pmatrix}$$

Toán tử sao chép được biểu diễn dưới dạng một ma trận nâng cấp *U* kích thước $(n!)^{m(N+1)} \times (n!)^{m(N+1)}$, ma trận này có các phần tử được tính như sau:

Xét quần thể bao gồm siêu cá thể ở trạng thái i, gọi $\pi_l(i)$ là cá thể có độ thích nghi cao nhất trong số các cá thể thường tức là:

$f(\pi_l(i)) = max\{f(\pi_k(i))/k = 1, \dots, N\}$, khi đó:

Nếu $f(\pi_0(i)) < f(\pi_l(i))$ thì $u_{ij} = 1$ với $j = (\pi_l(i), \pi_1(i), \pi_2(i), \dots, \pi_N(i)) \in S$, còn lại $u_{ik} = 0$ (với $k \neq j$).

Nếu $f(\pi_0(i)) \geq f(\pi_l(i))$ thì $u_{ii} = 1$, các phần tử còn lại của ma trận *U* bằng 0.

Toán tử sao chép sẽ thay thế cá thể thường có độ thích nghi cao hơn so với cá thể tinh hoa ở cùng thế hệ vào vị trí của cá thể tinh hoa, nếu không có cá thể nào thỏa mãn, quần thể sẽ giữ nguyên.

Ta thấy ma trận *U* cũng chỉ phụ thuộc vào giá trị của hàm *f*, như vậy ma trận *U* đối với một bài toán có kích thước cố định là cố định, vì các cá thể tinh hoa được sắp xếp theo thứ tự giảm dần của độ thích nghi nên ma trận *U* sẽ có dạng:

$$U = \begin{pmatrix} U_{11} & & \\ & \ddots & \\ U_{(n!)^m,1} & & U_{(n!)^m,(n!)^m} \end{pmatrix}$$

Với các ma trận con U_{ij} có kích thước $(n!)^{m \cdot N} \times (n!)^{m \cdot N}$.

Giả sử bài toán chỉ có một nghiệm tối ưu, như vậy các trạng thái từ 1 đến $(n!)^{m \cdot N}$ là các trạng thái mà tất cả đều có cá thể tinh hoa (chung một cá thể tinh hoa) mang nghiệm tối ưu (do cách đánh số không gian trạng thái), như vậy trong ma trận nâng cấp *U* trên, chỉ có ma trận con U_{11} là ma trận đơn vị tức là:

$$U_{11} = \begin{pmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{pmatrix}$$

Gọi P^+ là ma trận chuyển trạng thái tổng hợp của thuật giải tiến hóa, ta có:

$$P^+ = C^+ \cdot M^+ \cdot S^+ \cdot U = \begin{pmatrix} P & & \\ & \ddots & \\ & & P \end{pmatrix} \cdot \begin{pmatrix} U_{11} & & \\ & \ddots & \\ U_{(n!)^m,1} & & U_{(n!)^m,(n!)^m} \end{pmatrix}$$

$$= \begin{pmatrix} PU_{11} & & \\ & \ddots & \\ PU_{(n!)^m,1} & & PU_{(n!)^m,(n!)^m} \end{pmatrix}$$

- **Từ đây, ta có thể chứng minh được thuật toán di truyền với cá thể tinh hoa và toán tử sao chép hội tụ tới tối ưu toàn cục**

Thật vậy, vì U_{11} là ma trận đơn vị nên $PU_{11} = P$ là một ma trận ngẫu nhiên và dương. Do chỉ có một nghiệm tối ưu nên các ma trận con $PU_{k,1} \neq 0$ với $k \geq 2$, có thể tập hợp thành một ma trận chữ nhật $R \neq 0$. Ma trận vuông phù hợp với ma trận PU_{11} và ma trận R ở trên là ma trận:

$$T = \begin{pmatrix} PU_{22} & & \\ & \ddots & \\ PU_{(n!)^m,2} & & PU_{(n!)^m,(n!)^m} \end{pmatrix} \neq 0$$

Vậy, $P^+ = \begin{pmatrix} P & 0 \\ R & T \end{pmatrix}$, theo định nghĩa 2, P^+ là ma trận giản ước được.

Từ đó, áp dụng định lý 3 ta có:

$$(P^+)^\infty = \begin{pmatrix} P^\infty & 0 \\ R^\infty & 0 \end{pmatrix}$$

là một ma trận ổn định với $(P^+)^\infty = 1'\Pi^\infty$ và $\Pi^\infty = \Pi^{(0)} \cdot (P^+)^\infty$ là giá trị véc tơ hàng ổn định của $(P^+)^\infty$ không phụ thuộc vào giá trị $\Pi^{(0)}$ và thỏa mãn:

$\pi_i^\infty > 0$ với $1 \leq i \leq (n!)^{m \cdot N}$ và $\pi_i^\infty = 0$ với $((n!)^{m \cdot N} < i \leq (n!)^{m(N+1)}$

Như vậy, khi số thế hệ tiến tới vô cùng, phân phối xác suất trạng thái của quần thể tập trung hoàn toàn vào $(n!)^{m \cdot N}$ trạng thái đầu, tức là tập trung vào tất cả các trạng thái có chứa nghiệm tối ưu. Tức là:

$$\lim_{t \to \infty} P\{Z_t = f^*\} = 1.$$

Điều đó có nghĩa thuật toán di truyền với cá thể tinh hoa và toán tử sao chép có tính chất hội tụ tới tối ưu toàn cục. Sự hội tụ tới tối ưu toàn có thể được giải thích một cách trực quan như sau:

Giả sử tại thế hệ thứ t của quần thể, quần thể $P(t)$ đang ở trạng thái $i \in S$ nào đó và $\pi_0(i)$ là cá thể tinh hoa tương ứng, tức là cá thể tốt nhất của quần thế tại thế hệ này và nó không tham gia vào các toán tử di truyền của quần thể. Bây giờ ta xét thế hệ tiếp theo $t+1$. Nếu thế hệ này không có cá thể nào tốt hơn cá thể tinh hoa $\pi_0(i)$, thì cá thể tinh hoa này được duy trì cho thế hệ $t+1$. Để khẳng định rằng khi số thế hệ tiến ra vô hạn ($t \to \infty$) quần thể gặp cá thể tối ưu toàn cục, ta giả sử j là trạng thái của quần thể mà có cá thể tinh hoa $\pi_0(j)$ tốt hơn (tức là $f(\pi_0(j)) > f(\pi_0(i))$). Do $p_{ij}^{(n)} > 0, \forall n \in N$ (do P là ma trận chính quy), nên tồn tại một thế hệ thứ $t+k$ nào đó đạt được trạng thái j (vì nếu không, tức mọi thế hệ kể từ thế hệ thứ t không đạt được trạng thái j, thì $p_{ij}^{(n)} > 0, \forall n \geq t+1$, điều này mâu thuẫn). Mặt khác, không gian trạng thái S là hữu hạn, do đó bài toán tồn tại tối ưu toàn cục f^*. Chứng minh ở trên khẳng định quần thể sẽ gặp tối ưu toàn cục khi $t \to \infty$ với xác suất 1, nghĩa là chắc chắn.

4.4. Kết luận

Chương 4 đã trình bày một số vấn đề cơ bản nhất về xích Markov. Tiếp theo đó đã phân tích các thuộc tính hội tụ của thuật toán tiến hóa truyền thống và thuật toán tiến hóa mà tác giả đề nghị trong chương 3 bằng cách áp dụng các tính chất của xích Markov. Trên cơ sở phân tích xích Markov của thuật toán di truyền, luận án đã chứng tỏ thuật toán đề nghị trong chương 3 hội tụ tới tối ưu toàn cục.

KẾT LUẬN

Trong thời gian qua, mặc dù với những khó khăn vốn có của nghiên cứu sinh trong nước về các vấn đề như: Tiếp cận các thành tựu nghiên cứu của thế giới, trao đổi thông tin khoa học với các nhà khoa học nước ngoài, các nguồn tài liệu tham khảo,...Nhưng với sự nỗ lực của bản thân và được sự hướng dẫn tận tình của hai cán bộ hướng dẫn, tác giả luận án đã hoàn thành các mục tiêu luận án đặt ra ban đầu. Các kết quả cụ thể đạt được như sau:

1. Nghiên cứu về tổng quan của bài toán: Luận án đã phân tích, đánh giá, so sánh các giải pháp đã áp dụng cho các bài toán lập lịch job shop. Trên cơ sở đó đề xuất một số hướng nghiên cứu để giải quyết bài toán này.

2. Luận án đã đề xuất một thuật toán di truyền lai mới: Thuật toán này kết hợp thuật toán di truyền với các kỹ thuật tìm kiếm khác cho bài toán lập lịch job shop. Trong phương pháp đề xuất này, có sự đổi mới các công đoạn của thuật toán di truyền như: Mã hóa lời giải, đột biến và trao đổi chéo. Phương pháp đề xuất này đã được cài đặt và chạy thử nghiệm trên các bài toán test chuẩn cho kết quả tốt. Kết quả đã được so sánh với kết quả các giải pháp trước đó để chứng tỏ tính vượt trội của nó.

3. Luận án đã song song hóa thuật toán đề xuất: Thuật toán mới đề xuất được song song hóa, cài đặt và chạy thử nghiệm cho kết quả tốt và rút ngắn được nhiều lần thời gian thực thi với cùng bộ tham số và dữ liệu vào trong thuật toán tuần tự. Kết quả này đã được chuyển thành bài báo tham gia hội nghị quốc tế về ‘‘xử lý tín hiệu số và công nghệ thông tin - ISSPIT’’ 2012.

4. Luận án đã chứng minh tính hội tụ tới tối ưu toàn cục của thuật toán di truyền lai mới với mã hóa tự nhiên cho bài toán lập lịch job shop. Kết quả này đã chuyển thành bài báo tham gia hội nghị quốc tế về ‘‘xử lý tín hiệu số và công nghệ thông tin - ISSPIT’’ 2012.

HƯỚNG NGHIÊN CỨU TIẾP THEO

Đã có nhiều giải pháp cho bài toán lập lịch job shop trong những năm qua. Tuy nhiên, một khó khăn cố hữu mà các phương pháp cho JSP đang phải đối mặt đó là không một heuristic nào có thể đảm bảo rằng sự thực thi của nó đã được khai thác đúng mức và giải quyết triệt để bài toán hóc búa này. Để vượt qua các rào cản hiện tại của các giải pháp cho JSP, các thiết kế thử nghiệm nghiêm ngặt và sự phân tích kỹ lưỡng các phương pháp lai là điều hết sức cần thiết để tạo ra được các phương pháp mạnh cho JSP.

Một vấn đề nữa là các công trình nghiên cứu về lập lịch nói chung còn đơn giản hơn nhiều so với các bài toán thế giới thực. Trong xu thế phát triển của khoa học kỹ thuật ở kỷ nguyên 21, các bài toán trong thực tiễn có thể có nhiều ràng buộc phức tạp hơn, các hàm mục tiêu cũng mềm dẻo hơn và các đặc trưng động hơn. Điều đó cho thấy cần phải mở rộng các cách tiếp cận đã đề xuất cho JSP sao cho nó kết hợp chặt chẽ các ràng buộc của các bài toán trong thực tế.

Đây là vấn đề được đặt ra cho những nghiên cứu tiếp theo về giải pháp cho các bài toán lập lịch job shop.

DANH MỤC CÔNG TRÌNH KHOA HỌC CỦA TÁC GIẢ LIÊN QUAN ĐẾN LUẬN ÁN

1. Nguyễn Hữu Mùi, Vũ Đình Hoà (2009), "Solving the permutation flow shop scheduling problem by genetic algorithms", *Journal of Science of HNUE* Vol. 54 (1), pp. 40-45.
2. Nguyễn Hữu Mùi, Vũ Đình Hoà (2009), "Solving the flow shop scheduling problem by genetic algorithms", *Journal of Science of HNUE* Vol. 54 (6), pp. 35-41.
3. Nguyễn Hữu Mùi, Vũ Đình Hoà (2010), "Active schedules and a new hybrid genetic algorithm for the job shop scheduling problem", *VNU Journal of Science, Mathematics - Physics* Vol. 26 (4), pp. 213-221.
4. Nguyễn Hữu Mùi, Vũ Đình Hoà (2010), "Solving the job shop scheduling problem by genetic algorithm", *Addendum Proceedings IEEE RIVF 2010*, pp. 29-32.
5. Nguyễn Hữu Mùi, Vũ Đình Hoà (2011), "Giải bài toán lập lịch job shop bằng thuật toán di truyền", *Kỷ yếu hội thảo quốc gia lần thứ XIII, một số vấn đề chọn lọc của công nghệ thông tin và truyền* thông, tr. 71-82.
6. Nguyễn Hữu Mùi, Vũ Đình Hoà (2011), "Một thuật toán di truyền lai mới cho bài toán lập lịch công việc", *Kỷ yếu hội nghị khoa học công nghệ quốc gia lần thứ V*, tr. 239-249.
7. Nguyễn Hữu Mùi, Vũ Đình Hoà (2012), "Một thuật toán di truyền hiệu quả cho bài toán lập lịch job shop", *Tạp chí khoa học và công nghệ, Viện Khoa học và Công nghệ Việt Nam* Tập 50 (5), tr. 565-577.
8. Nguyễn Hữu Mùi, Vũ Đình Hoà, Lục Trí Tuyên (2012), "A Parallel Genetic Algorithm for the Job Shop Scheduling Problem", *Proceedings IEEE ISSPIT 2012*, Published online.

9. Nguyễn Hữu Mùi, Vũ Đình Hoà, Lục Trí Tuyên (2012), "Convergence Analysis of the New Hybrid Genetic Algorithm for the Job Shop Scheduling Problem", *Proceedings IEEE ISSPIT 2012*, Published online.

.

TÀI LIỆU THAM KHẢO

Tiếng Việt

1. Nguyễn Duy Tiến (2001), *Các mô hình xác suất và ứng dụng, Phần 1-Xích Markov và ứng dụng*, NXB Đại học Quốc gia Hà Nội.

Tiếng Anh

2. Aarts E. H. L. & Van Laarhoven P. J. M. & Lenstra J. K. & Ulder N. L. J. (1994), "A Computational Study of Local Search Algorithms for Job-Shop Scheduling", *ORSA Journal on Computing* Vol. 6 (2), pp.118-125.
3. Adams J. & Balas E. & Zawack D. (1998), "The shifting bottleneck procedure for job shop scheduling", *Management Science* Vol. 34 (3), pp. 391-401.
4. Akers S. B. (1956), "A Graphical Approach to Production Scheduling Problems", *Operations Research* Vol. 4, pp. 244-245.
5. Andrea Rossi và Elena Boschi (2010), "A hybrid heuristic to solve the parallel machines job shop sheduling problem", *Advances in Engineering Software* Vol. 40, pp. 118-127.
6. Applegate D. & Cook W. (1991), "A Computational Study of the Job-Shop Scheduling Problem", *ORSA Journal on Computing*, Spring Vol. 3 (2), pp. 149-156.
7. Ashour S. (1967), "A Decomposition Approach for the Machine Scheduling Problem", *International Journal of Production Research* Vol. 6 (2), pp. 109-122.

8. Atif Shahzad & Nasser Mebarki (2010), "Discovering dispatching rules for job shop scheduling problem through data mining", www.enim.fr/mosim2010/articles/206.pdf

9. Blazewicz J. & Dror M. & Weglarz J. (1991), "Mathematical Programming Formulations for Machine Scheduling: A Survey", *European Journal of Operational Research, Invited Review* Vol. 51 (3), pp. 283-300.

10. Brooks G. H. & White C. R. (1969), "An algorithm for finding optimal or near optimal solutions to the production scheduling proble", *The Journal of Industrial Engineering* Vol. 16 (1), pp. 34-40.

11. Brucker P. (1994), *A polynomial time algorithm for the two machines Job Shop scheduling problem with a fixed number of jobs*, OR Spektrum 16, pp. 5-7.

12. Carlier J. & Pinson E. (1989), "An algorithm for solving the job-shop problem", *Management Science* Vol. 35 (2), pp. 164-176.

13. Cheng R. et al. (1996), "A Tutorial Survey of Job-Shop Scheduling Problems using Genetic Algorithms-I", *Representation, Computers & Industrial Engineering* Vol. 30 (4), pp. 983-997.

14. Christian Artigues & Dominique Feillet (2008), "A branch and bound method for the job-shop problem with sequence-dependent setup times", *Annals of Operations Research* Vol. 159, pp. 135-159.

15. Colorni A. & Dorigo M. & Maniezzo V. & Trubian M. (1994), "Ant-System for Job-Shop Scheduling", *Belgian Journal of Operations Research, Statistics and Computer Science* Vol. 34 (1), pp. 39-54.

16. Conway R. W. & Maxwell W. L. & Miller L. W. (1967), *Theory of Scheduling*, Addison Wesley, Reading, Mass., USA.

17. Davis T.E. and Principe J.C. (1991), “A simulated annealing like convergent theory for the simple genetic algorithm”, *Proceedings of the fourth Conference on Genetic Algorithm*s, R.K. Belew and L.B. Booker (Eds.), San Mateo: Morgan Kaufmann, pp. 174-181

18. Della Croce F. & Menga G. & Tadei R. & Cavalotto M. & Petri L. (1993), "Cellular Control of Manufacturing Systems", *European Journal of Operational Research* Vol. 69, pp. 498-509.

19. Deming Lei, "Fuzzy job shop scheduling problem with availability constraints", *Computers & Industrial Engineering, journal homepage*: www.elsevier.com/ locate/caie.

20. Denebourg J. L. & Pasteels J. M. & Verhaeghe J. C. (1983), "Probabilistic behaviour in Ants: A Strategy of Errors ?", *Journal of Theoretical Biology* Vol. 105, pp. 259-271.

21. Erschler J. & Roubellat F. & Vernhes J. P. (1976), "Finding Some Essential Characteristics of the Feasible Solutions for a Scheduling Problem", *Operations Research* Vol. 24 (4), pp. 774-783.

22. Fisher M. L. & Lageweg B. J. & Lenstra J. K. & Rinnooy Kan A. H. G. (1983), "Surrogate Duality Relaxation for Job-Shop Scheduling", *Discrete Applied Mathematics* Vol. 5 (1), pp. 65-75.

23. Fisher M. L. (1973), "Optimal Solution of Scheduling Problems using Lagrange Multipliers", *Operations Research* Vol. 21, pp. 1114-1127.

24. Fox M. S. & Sadeh N. (1990), "Why Is Scheduling Difficult? A CSP Perspective", *in Aiello L. (ed) ECAI-90 Proceedings of the 9th European Conference on Artificial Intelligence*, August 6-10, Stockholm, Sweden, pp. 754-767.

25. French S. (1982), *Sequencing and Scheduling - An Introduction to the Mathematics of the Job-Shop*, Ellis Horwood, John-Wiley & Sons, New York.

26. Garey M. R. & Johnson D. S. & Sethi R. (1976), "The complexity of flow shop and job shop scheduling", *Mathematics of Operations Research* Vol. 1 (2), pp. 117-129.

27. Garrido A. & Salido M. A. & Barber F. & López M. A. (2009), "Heuristic Methods for Solving Job-Shop Scheduling Problems", www.en.scientificcommons.org/50066401 - Hoa Kỳ.

28. Giffler B. & Thompson G. (1960), "Algorithms for Solving Production Scheduling Problems", *Operations Research* Vol. 8 (4), pp. 487-503.

29. Glover F. & Greenberg H. J. (1989), "New Approaches for Heuristic Search: A Bilateral Linkage with Artificial Intelligence", *European Journal of Operations Research* Vol. 39 (2), pp. 119-130.

30. Glover, F. (1989) Tabu Search - Part I, *ORSA Journal on Computing* Vol. 1 (3), pp. 190-206.

31. Goldberg D. E. (1986), *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley, Reading, Mass.

32. Guerriero F. (2008)**,** "Hybrid Rollout Approaches for the Job Shop Scheduling Problem", *Jounal Optimization Theory and Applications* Vol. 139, pp. 419-438.

33. Hefetz N. & Adiri I. (1982), "An Efficient Optimal Algorithm for the Two-Machines Unit-Time Job-Shop Schedule-Length Problem", *Mathematics of Operations Research* Vol. 7, pp. 354-360.

34. Holland J. H. (1975), *Adaptation in Natural and Artificial Systems*. Unuv. of Michigan Press.

35. Hung-Pin Chiu & Kun-Lin Hsieh & Yi-Tsung Tang & Ching-Yu Wang (2007), "A Tabu Genetic algorithm with Search Area Adaptation for the Job-Shop Scheduling Problem", *Proceedings of the 6th WSEAS Int. Conf. on Artificial Intelligence, Knowledge Engineering and Data Bases*, Corfu Island, Greece, February 16-19, pp. 76-80.

36. Iosifescu M. (1980), *Finite Markov Prosseses and Their Applications*, Chichester: Wiley.

37. Jackson J. R. (1955), *Scheduling a Production Line to Minimise Maximum Tardiness*, Research Report 43, Management Science Research Projects, University of California, Los Angeles, USA.

38. Jackson J. R. (1956), "An Extension of Johnson's Result on Job Lot Scheduling", *Naval Research Logistics Quarterly* Vol. 3 (3), pp. 201-203.

39. Jin-hui Yang & Liang Sun & Heow Pueh Lee & Yun Qian (2008), "Clonal Selection Based Memetic Algorithm for Job Shop Scheduling Problems", *Journal of Bionic Engineering* Vol. 5, pp. 111-119.

40. Johnson D. S. & Aragon C. R. & McGeoch L. A. & Schevon C. (1989), "Optimization by Simulated Annealing: An Experimental Evaluation", *Graph Partitioning, Operations Research* Vol. 37 (6), pp. 865-892.

41. Johnson S. M. (1954), "Optimal Two- and Three-Stage Production Schedules with Set-Up Times Included", *Naval Research Logistics Quarterly* Vol. 1, pp. 61-68.

42. Kamrul Hasan S. M. (2008), "GA with Priority Rules for Solving Job-Shop Scheduling Problems", *Evolutionary Computation*, CEC 2008 (IEEE World Congress on Computational Intelligence), pp. 1913-1920.

43. Karimi Gavareshki M. H. & Fazel Zarandi M. H. (2008), "A Heuristic Approach for Large Scale Job Shop Scheduling Problems", *Journal of Applied Sciences* Vol. 8 (6), pp. 992-999.

44. Kravchenko S. A. & Sotskov Y. N. (1996), "Optimal makespan schedule for three jobs on two machines", *ZOR - Mathematical Methods of Operations Research* Vol. 43, pp. 233-238.

45. Laguna M. & Glover F. (1993), "Integrating Target Analysis and Tabu Search for Improved Scheduling System", *Expert Systems with Applications* Vol. 6, pp. 287-297.

46. Lining Xing & Yingwu Chen & Kewei Yang (2008), "Knowledge-based ant colony optimization for the flexible job shop scheduling problems", *Dynamics of Continuous, Discrete and Impulsive Systems, Series B: Applications & Algorithms* 15, pp. 431-446.

47. Lo & Hsu (1993), "A Parallel Distributed Processing Technique for Job-Shop Scheduling Problems", *IJCNN International Joint Conference on Neural Networks*, Nagoya, Japan, 25-29 Oct, Vol. 2, pp. 1602-1605.

48. Manne A. S. (1960), "On the Job-Shop Scheduling Problem", *Operations Research* Vol. 8, pp. 219-223.

49. Miloš Šeda (2007), *Mathematical Models of Flow Shop and Job Shop Scheduling Problems*, World Academy of Science, Engineering and Technology 31.

50. Moghaddas R. & Houshmand M. (2008), "Job-Shop Scheduling Problem With Sequence Dependent Setup Times", *Proceedings of the International MultiConference of Engineers and Computer Scientists* Vol. II IMECS 2008, pp. 19-21.

51. Moraglio A. & Ten Eikelder H. M. M. & Tadei R. (2005), "Genetic Local Search for Job Shop Scheduling Problem", www.essex.ac.uk/technical-reports/2005/ csm435.pdf

52. Muth J. F. & Thompson G. L. (1963), *Industrial Scheduling*, Prentice-Hall, Englewood Cliffs, N. J.

53. Nakano R. & Yamada T. (1991), "Conventional genetic algorithm for job shop problems", *In Proceedings of International Conference on Genetic Algorithms (ICGA '91)*, pp. 474-479.

54. Omar Castrillon & William Sarache & Jaime Giraldo (2009), "Job shop methodology based on an ant colony", *Dyna, Año 76* (159), pp. 177-184. Medellin, Septiembre de 2009. ISSN 0012-7353

55. Osman I. H. & Kelly J. P. (1996), "Meta-Heuristics: An Overview", in Osman, I. H. and Kelly, J. P. *Meta-Heuristics: Theory and Applications*, Kluwer Academic Publishers, Norwell, MA, USA, Chapter 1, pp. 1-21.

56. Panwalkar S. S. & Iskander W. (1977), "A Survey of Scheduling Rules", *Operations Research*, Jan-Feb Vol. 25 (1), pp. 45-61.

57. Pesch E. & Tetzlaff U. A. W. (1996), "Constraint Propagation Based Scheduling of Job Shops", *INFORMS Journal on Computing*, Spring Vol. 8 (2), pp. 144-157.

58. Ramezanali Mahdavinejad (2010), "A New Approach to Job Shop Scheduling Problem", *Journal of Achievements in Materials and Manufacturing Engineering* Vol. 41 (1-2), pp. 200-206.

59. Röck H. & Schmidt G. (1983), "Machine aggregation heuristics in shop scheduling", *Methods of Operations Research* Vol. 45, pp. 303-314.

60. Rudolph G. (1994) "Convergence Analysis of Canonical Genetic Algorithms", *IEEE Transactions on Neural Networks, special issue on evolutonary computation* Vol. 5 (1), pp. 96-101.

61. Rui Zhang & Cheng Wu (2008), "Bottleneck machine identification based on optimization for the job shop scheduling problem", *ICIC Express Letters* Vol. 2 (2), pp. 175-180.

62. Rui Zhang & ChengWu (2009), "Bottleneck identification procedures for the job shop scheduling problem with applications to genetic algorithms", *International Journal Advanced Manufacturing Technology* Vol. 42, pp. 1153-1164.

63. Rui Zhang & Cheng Wu (2010), "A hybrid approach to large-scale job shop scheduling", *Application Intelligence* Vol. 32, pp. 47-59.

64. Sadeh N. & Fox M. S. (1996), "Variable and Value Ordering Heuristics for the Job Shop Scheduling Constraint Satisfaction Problem", *Artificial Intelligence* Vol. 86 (1), pp. 1-41.

65. E. Seneta E. (1981), *None-nagative Matrices and Markov Chains*, 2nd edition, New York:Springer.

66. Sotskov Y.N. & Shaklevich N. V. (1995), "NP-hardness of shop-scheduling problems with three jobs", *Descrete Applied Mathemayics* Vol. 59, pp. 237-266.

67. Surekha P. & Sumathi S. (2010), "Solving Fuzzy based Job Shop Scheduling Problems using Ga and Aco", *Journal of Emerging Trends in Computing and Information Sciences* Vol. 1 (2), pp. 95-102.

68. Szu H. & Hartley R. (1987), "Fast Simulated Annealing", *Phys. Lett. A.* Vol 122, pp. 157-162.

69. Ulder N. L. J. & Pesch E. & Van Laarhoven P. J. M. & Bandelt H. J. & Aarts E. H. L. (1994), "Genetic local search algorithm for the traveling salesman problem", *In Parallel Problem Solving from Nature* Vol. 1, pp. 109-116.

70. Van De Velde S. (1991), *Machine Scheduling and Lagrangian Relaxation,* Ph. D. Thesis, CWI Amsterdam, The Netherlands.

71. Van Laarhoven P. J. M. & Aarts E. H. L. & Lenstra J. K. (1992), "Job shop scheduling by simulated annealing", *Operations Research* Vol. 40 (1), pp. 113-125.

72. Van Laarhoven P. J. M. & Aarts E. H. L. (1987), S*imulated Annealing: Theory and Applications*, D. Reidel Publishing Company, Dordrecht, Netherlands.

73. Vinod V. & Sridharan R. (2008), "Dynamic job-shop scheduling with sequence-dependent setup times: simulation modeling and analysis", *International Journal Advanced Manufacturing Technology* Vol. 36, pp. 355-372.

74. Wang W. & Brunn P. (1995), "Production Scheduling and Neural Networks", *Operations Research Proceedings 1994*, Springer-Verlag, Berlin, pp. 173-178.

75. Williamson D. P. & Hall L. A. & Hoogeveen J. A. & Hurkens C. A. J. & Lenstra J. K. & Sevast'janov S. V. & Shmoys D. B. (1997), "Short Shop Schedules", *Operations Research* Vol. 45 (2), pp. 288-294.

76. Yamada T. & Nakano R. (1995), "Job-Shop Scheduling by Simulated Annealing Combined with Deterministic Local Search", *MIC'95 Meta-heuristics International Conference*, Hilton, Breckenridge, Colorado, USA, July 22-26, pp. 344-349.

77. Yamada T. (2003), *Studies on Metaheuristics for Jobshop and Flowshop scheduling problems,* Kyoto University, Kyoto - Japan.

78. Ye Li & Yan Chen (2010), "A Genetic Algorithm for Job-Shop Scheduling", *Journal of Software* Vol. 5 (3), pp. 269-274.

79. Ye Li & Yan Chen (2010), "Hybrid Algorithm Approach To Job Shop Scheduling Problem", *Global Journal of Computer Science and Technology* Vol. 10 Issue 8 Ver. 1.0 September 2010 Page 55.

80. Yokoi H. & Kakazu Y. & Minagawa M. (1994), "An Approach to the Autonomous Job-Shop Scheduling Problem by Vibrating potential Method", *IEEE ICNN'94 International Conference on Neural Network*, Orlando, Florida, 26-29 June Vol. 6, pp. 3877-3882.

81. Zhang C. et al. (2008), "A very fast TS/SA algorithm for the job shop scheduling problem", *Computers and Operations Research* Vol. 35 (1), pp. 282-294.

82. Zhi Huang, "A Modified Shifting Bottleneck Procedure for Job Shop Scheduling", www.paper.edu.cn/index.php/default/releasepaper/...

PHỤ LỤC

Các hàm chính của thuật toán di truyền lai mới mà luận án đề xuất được cài đặt bằng ngôn ngữ *C++* như dưới đây:

- **Hàm GT() dùng cho khởi tạo quần thể**

```
void Gene :: InitGT(Operation ***arrope, int numMachine, int numJob, int
                                                                numOpe)
{
    Random rd;
    Operation ***S;
    S = new Operation **[numMachine + 1];
    for(int i = 0; i < numMachine + 1; i++) S[i] = new
                                                Operation*[numJob + 1];
    List<Operation, minPtr> G;
    List<Operation, minPtr> C;
    List<Operation, minPtr> **PM= new List<Operation, minPtr>
                                                *[numMachine  + 1];
    for (int i = 0; i < numMachine + 1; i++)
        PM[i] = new List<Operation, minPtr>;
    for (int i = 1; i <= numJob; i++)
     {
        Operation *ope = arrope[i][1]->Clone();
        ope->ES = 0;
        ope->EC = ope->getPt();
        G.Add(ope);
     }
    for (int k = 0; k < numOpe ; k++)
```

```
{
   Operation *opeMinEc = findOpeMinEC(G);
   int machineIndex = opeMinEc->getMachine();
   List<Operation, minPtr> GM;
   for (int i = 0; i < G.Count; i++)
      if (G[i]->getMachine() == machineIndex)
        GM.Add(G[i]);
   List<Operation, minPtr> CM;
   for (int i = 0; i < GM.Count; i++)
       if (GM[i]->ES < opeMinEc->EC) CM.Add(GM[i]);
   int idOpeChooser = rd.Next(CM.Count);
   Operation *opeChooser = CM[idOpeChooser];
   int J = machineIndex;
   int I = PM[machineIndex]->Count + 1;
   opeChooser->s = opeChooser->ES;
   opeChooser->c = opeChooser->EC;
   S[J][I] = opeChooser->Clone();
   PM[J]->Add(S[J][I]);
   for (int I = 0; i < GM.Count; i++)
    if (GM[i] != opeChooser)
     {
       GM[i]->ES = Max(GM[i]->ES, opeChooser->EC);
       GM[i]->EC = GM[i]->ES + GM[i]->getPt();
     }
   Operation *opeNext = opeChooser->findOpeNext (arrope,
                                                 numMachine);
   if (opeNext != NULL)
```

```
        {
            G.Add(opeNext);
            Operation *opeResult = findOpeBothMachine(PM, opeNext);
            opeNext->ES = Max(opeChooser->EC, opeResult->EC);
            opeNext->EC = opeNext->ES + opeNext->getPt();
        }
       G.Remove(opeChooser);
    }
    for (int i = 1; i <= numMachine; i++)
     for (int j = 1; j <= numJob; j++)
      if (S[i][j] != NULL)
       {
         Add(S[i][j]->Clone());
       }
}
```

- **Hàm khởi tạo quần thể ban đầu**

```
void Population::InitPopulation()
  {
    M = 0;
    for (int k = 0; k < nPopulation; k++)
     {
        Gene *gene = new Gene;
        gene->InitGT(arrope, numMachine, numJob, numOpe);
        gene->makespan();
        Add(gene);
     }
    tinhM();
    tinhFitness();
  }
```

- **Hàm tính tham số M**

```
void Population :: tinhM()
  {
    M = 0;
    float _M = 0.0;
    for (int i = 0; i < Count; i++)
    {
       ptu[i]->makespan();
       _M+ = 2.0*((float)ptu[i]->MakeSpan/nPopulation);
    }
    M = (int)_M;
  }
```

- **Hàm tính độ thích nghi của cá thể**

```
void Population :: tinhFitness()
  {
    for (int i = 0; i < Count; i++)
       ptu[i]->fitness(M);
  }
```

- **Hàm chọn cá thể tham gia đột biến**

```
void Population :: selectMutation()
   {
     List<Gene, maxPtr> temp = Select();
     List<Gene, maxPtr> listMutation;
     for (int i = 0; i < nPopulation; i++)
       if (rd.Next() < pMutation)
        {
          listMutation.Add(temp[i]);
```

```
    }
    if (listMutation.Count > 0)
    for (int i = 0; i < listMutation.Count ; i ++)
      {
            for(int ii = 0;ii < 5; ii++)
            {
                  Gene *ge = listMutation[i]->Clone();
                  ge->Mutation();
                  if (ge->updateGene() == true &&
                                    kiemtratrunggene(ge)==false)
                {
                  ge->Eval(M);
                  if(ge->MakeSpan <= listMutation[i]->MakeSpan)
                        {
                              Add(ge->Clone());
                              break;
                        }
                  }
             }
        }
  }
```

- **Hàm đột biến**

```
void Gene :: Mutation()
   {
     Random rd;
     int p1, p2;
     do
```

```
    {
        p1 = rd.Next(Count);
        p2 = rd.Next(Count);
        if ((ptu[p1]->getMachine() == ptu[p2]->getMachine()) && (p1
                                                        != p2))
        {
            Operation *temp;
            temp = ptu[p2];
            ptu[p2] = ptu[p1];
            ptu[p1] = temp;
        }
    }
    while (ptu[p1]->getMachine() != ptu[p2]->getMachine() || (p1 ==
                                                        p2));
  }
```

- **Hàm chọn cá thể tham gia trao đổi chéo**

```
void Population :: selectCrossOver3()
  {
    List<Gene, maxPtr> temp = Select();
    List<Gene, maxPtr> listcross;
    for (int i = 0; i < nPopulation; i++)
     if (rd.Next() < pCrossover)
     {
       listcross.Add(temp[i]);
     }
    while (listcross.Count % 3 != 0) listcross.RemoveAt(listcross.Count –
                                                        1);
```

```
if (listcross.Count > 0)
 for (int i = 0; i < listcross.Count - 2; i += 3)
   {
      Gene *g= new Gene;
      do
      {
         g->InitCrossOver3 (arrope, numMachine, numJob, numOpe,
                        *listcross[i], *listcross[i + 1], *listcross[i + 2]);
      } while (g->updateGene() == false);
      if (kiemtratrunggene(g) == false)
      {
         g->Eval(M);
         Add(g->Clone());
      }
   }
}
```

- **Hàm trao đổi chéo**

```
void Gene :: InitCrossOver3(Operation ***arrope,int numMachine,
                  int numJob, int numOpe, Gene g1, Gene g2, Gene g3)
{
    Operation ***S1;  S1= new Operation **[numMachine + 1];
    for(int i =0; i < numMachine + 1; i++) S1[i] = new
                                          Operation*[numJob + 1];
    Operation ***S2;
    S2 = new Operation **[numMachine + 1];
    for(int i = 0; i < numMachine + 1; i++) S2[i] = new
                                          Operation*[numJob + 1];
```

```
Operation ***S3;
S3= new Operation **[numMachine + 1];
for(int i = 0; i < numMachine + 1; i++) S3[i] = new
                                        Operation*[numJob + 1];
{
    int k = 0;
    for (int i = 1; i <= numMachine; i++)
       for (int j = 1; j <= numJob; j++)
       {
          S1[i][j] = g1[k]->Clone();
          k++;
       }
}
{
    int k = 0;
    for (int i = 1; i <= numMachine; i++)
       for (int j = 1; j <= numJob; j++)
       {
          S2[i][j] = g2[k]->Clone();
          k++;
       }
}
{
    int k = 0;
    for (int i = 1; i <= numMachine; i++)
       for (int j = 1; j <= numJob; j++)
       {
```

```
            S3[i][j] = g3[k]->Clone();
            k++;
        }
}

Random rd;
Operation ***S;
S= new Operation **[numMachine + 1];
for(int i =0; i < numMachine + 1;i++) S[i] = new
                                        Operation*[numJob + 1];
List<Operation, minPtr> G;
List<Operation, minPtr> C;
List<Operation, minPtr> **PM = new List<Operation, minPtr>
                                        *[numMachine + 1];
for (int i = 0; i < numMachine + 1; i++)
    PM[i] = new List<Operation, minPtr>;
for (int i = 1; i <= numJob; i++)
 {
    Operation *ope = arrope[i][1]->Clone();
    ope->ES = 0;
    ope->EC = ope->getPt();
    G.Add(ope);
 }
for (int k = 0; k < numOpe ; k++)
{
   Operation *opeMinEc = findOpeMinEC(G);
   int machineIndex = opeMinEc->getMachine();
```

```
List<Operation,minPtr> GM;
for (int i = 0; i < G.Count; i++)
   if (G[i]->getMachine() == machineIndex)
     GM.Add(G[i]);
List<Operation, minPtr> CM;
for (int i = 0; i < GM.Count; i++)
    if (GM[i]->ES < opeMinEc->EC) CM.Add(GM[i]);
Operation *opeChooser = NULL;
int num = rd.Next(3);
int Lmin = 30000;
if (num == 0)
{
    for (int kk = 0; kk < CM.Count; kk++)
    {
       int i = CM[kk]->getJob();
       int l = 30000;
       for (int j = 1; j <= numJob; j++)
          if (S1[machineIndex][j]->getJob() == i)
          {
             l = j;
             break;
          }
       if (l < Lmin) Lmin = l;
    }
    int r = S1[machineIndex][Lmin]->getJob();
    for (int i = 0; i < CM.Count; i++)
```

```
    {
        Operation *x = CM[i];
        if (x->getJob() == r) opeChooser = x;
    }
}
else if (num == 1)
{
    for (int kk = 0; kk < CM.Count; kk++)
    {
        int i = CM[kk]->getJob();
        int l = 30000;
        for (int j = 1; j <= numJob; j++)
            if (S2[machineIndex][j]->getJob() == i)
            {
                l = j;
                break;
            }
        if (l < Lmin) Lmin = l;
    }
    int r = S2[machineIndex][Lmin]->getJob();
    for (int i = 0; i < CM.Count; i++)
    {
        Operation *x = CM[i];
        if (x->getJob() == r) opeChooser = x;
    }
}
else
```

```
{
    for (int kk = 0; kk < CM.Count; kk++)
    {
        int i = CM[kk]->getJob();
        int l = 30000;
        for (int j = 1; j <= numJob; j++)
            if (S3[machineIndex][j]->getJob() == i)
            {
                l = j;
                break;
            }
        if (l < Lmin) Lmin = l;
    }
    int r = S3[machineIndex][Lmin]->getJob();
    for (int i = 0; i < CM.Count; i++)
    {
        Operation *x = CM[i];
        if (x->getJob() == r) opeChooser = x;
    }
}
int J = machineIndex;
int I = PM[machineIndex]->Count + 1;
opeChooser->s = opeChooser->ES;
opeChooser->c = opeChooser->EC;
S[J][I] = opeChooser->Clone();
PM[J]->Add(S[J][I]);
for (int i = 0; i < GM.Count; i++)
```

```
            if (GM[i] != opeChooser)
            {
               GM[i]->ES = Max(GM[i]->ES, opeChooser->EC);
               GM[i]->EC = GM[i]->ES + GM[i]->getPt();
            }
           Operation *opeNext = opeChooser->findOpeNext(arrope,
                                                        numMachine);
           if (opeNext != NULL)
            {
               G.Add(opeNext);
               Operation *opeResult = findOpeBothMachine(PM,
                                                         opeNext);
               opeNext->ES = Max(opeChooser->EC, opeResult->EC);
               opeNext->EC = opeNext->ES + opeNext->getPt();
            }
           G.Remove(opeChooser);
        }
        for (int i = 1; i <= numMachine; i++)
         for (int j = 1; j <= numJob; j++)
          if (S[i][j] != NULL)
           {
             Add(S[i][j]->Clone());
           }
    }
```

- **Hàm chọn lọc**

```
List<Gene,maxPtr> Population :: Select()
  {
```

```
List<Gene, maxPtr> newPopulation;
Gene *max;
max = SelectMax()->Clone();
if(max->MakeSpan <= TheBestGene->MakeSpan)
  {
    newPopulation.Add(max);
    Copy();
  }
else  newPopulation.Add(TheBestGene);
tinhFtotal();
tinhP();
tinhQ();
for(int i=0;i<nPopulation - 1;i++)
  {
   float rd1 = (float)rd.Next();
   if (ptu[0]->q >= rd1) newPopulation.Add(ptu[0]->Clone());
   else
    for(int j=1;j<Count;j++)
        if (ptu[j - 1]->q < rd1 && rd1 <= ptu[j]->q)
        {
           newPopulation.Add(ptu[j]->Clone());
           break;
        }
  }
return newPopulation;
}
```

- **Hàm sao chép**

```
void  Population:: Copy()
```

```
{
  Gene *max = NULL;
  float GiaTri = 0;
  for (int i=0; i<Count; i++)
  if (GiaTri < ptu[i]->Fitness)
   {
      max = ptu[i];
      GiaTri = max->Fitness;
   }
  TheBestGene = max->Clone();
}
```