

PHÁT TRIỂN THUẬT TOÁN BEES GIẢI BÀI TOÁN JOB SHOP SCHEDULING

Development of a Bees Algorithm for the Job Shop Scheduling Problem

Phan Tấn Quốc^{(1)*}, Đỗ Ngọc Nhung⁽²⁾

⁽¹⁾Trường Đại học Sài Gòn

⁽²⁾Học viên Cao học, Trường Đại học Sài Gòn

Tóm tắt

Bài toán Job Shop Scheduling (Job Shop Scheduling Problem - JSSP) là một trong những bài toán tối ưu tổ hợp thuộc lớp *NP-hard*. JSSP có ý nghĩa quan trọng cả về mặt lý thuyết lẫn ứng dụng, đặc biệt là trong các lĩnh vực quản lý sản xuất, lập kế hoạch công việc, vận tải và hệ thống dịch vụ. Trong nhiều thập kỷ qua, JSSP đã thu hút sự quan tâm của cộng đồng nghiên cứu, với nhiều hướng tiếp cận được đề xuất, bao gồm các thuật toán giải chính xác, heuristic và metaheuristic. Trong bài báo này, chúng tôi phát triển một thuật toán mới dựa trên thuật toán Bees cơ bản để giải JSSP; chúng tôi đặt tên thuật toán này là BeeJSS. Chúng tôi đã tiến hành đánh giá hiệu quả của thuật toán BeeJSS trên hệ thống dữ liệu thực nghiệm chuẩn Lawrence. Kết quả thực nghiệm cho thấy thuật toán BeeJSS có hiệu năng so sánh được với một số thuật toán metaheuristic hiện biết.

Từ khóa: *bees algorithm, heuristic algorithm, job shop scheduling, lawrence benchmark dataset, metaheuristic algorithm*

Abstract

The Job Shop Scheduling Problem (JSSP) is a combinatorial optimization problem classified as *NP-hard*. It has important implications in both theory and applications, particularly in production management, job planning, transportation, and service systems. Over the past decades, JSSP has attracted considerable research attention, with many approaches proposed, including exact algorithms, heuristics, and metaheuristics. In this paper, we propose a new algorithm, called BeeJSS, which is based on the basic Bees Algorithm, to solve the JSSP. We evaluate the effectiveness of BeeJSS on the standard Lawrence benchmark dataset. Experimental results indicate that BeeJSS performs comparably to several metaheuristic algorithms currently known.

Keywords: *bees algorithm, heuristic algorithm, job shop scheduling, lawrence benchmark dataset, metaheuristic algorithm*

1. Giới thiệu

1.1. Định nghĩa bài toán Job Shop Scheduling

Bài toán Job Shop Scheduling (Job Shop Scheduling Problem - JSSP) được phát biểu như sau:

Cho tập n công việc $\{J_1, J_2, \dots, J_n\}$ và m máy $\{M_1, M_2, \dots, M_m\}$. Mỗi công việc J_i bao gồm một dãy các công đoạn (operations) cần được xử lý trên các máy. Mỗi công đoạn $O_{i,k}$

*Tác giả liên hệ: quocpt@sgu.edu.vn

của công việc J_i phải được xử lý liên tục trên một máy cụ thể $m_{i,k}$ trong một khoảng thời gian $p_{i,k}$. Thứ tự thực hiện các công đoạn trong cùng một công việc là cố định.

Các ràng buộc của bài toán JSSP được cho như sau:

Mỗi máy tại một thời điểm chỉ có thể xử lý tối đa một công đoạn. Các công đoạn của cùng một công việc phải được thực hiện tuần tự theo đúng thứ tự đã định.

Mục tiêu của bài toán là xây dựng một lịch biểu (schedule) – tức là xác định thời điểm bắt đầu và kết thúc của mỗi công đoạn trên các máy, cũng như thứ tự thực hiện các công đoạn trên từng máy sao cho thời điểm hoàn thành tất cả công việc là nhỏ nhất; giá trị này gọi là *makespan*.

Như vậy nếu gọi C_i là thời điểm hoàn thành công việc J_i , thì mục tiêu của bài toán là tối thiểu hóa giá trị lớn nhất của tập $\{C_1, C_2, \dots, C_n\}$ [22].

1.2. Ví dụ minh họa

Xét ví dụ $n=6$ công việc và $m=3$ máy. Mỗi công việc được mô tả bởi dãy công đoạn (M, p) , trong đó M là máy được sử dụng và p là thời gian xử lý.

Công việc Thứ tự công đoạn (máy, thời gian)

$J_1: (M_1, 3) \rightarrow (M_2, 2) \rightarrow (M_3, 2)$

$J_2: (M_2, 2) \rightarrow (M_3, 1) \rightarrow (M_1, 4)$

$J_3: (M_3, 4) \rightarrow (M_1, 3) \rightarrow (M_2, 2)$

$J_4: (M_1, 2) \rightarrow (M_3, 3) \rightarrow (M_2, 3)$

$J_5: (M_2, 3) \rightarrow (M_1, 2) \rightarrow (M_3, 4)$

$J_6: (M_3, 2) \rightarrow (M_2, 4) \rightarrow (M_1, 3)$

Một lịch biểu tối ưu tìm được có giá trị tối ưu *makespan* bằng 17.

$M_1: J_4 \rightarrow J_1 \rightarrow J_5 \rightarrow J_3 \rightarrow J_6 \rightarrow J_2$

$M_2: J_5 \rightarrow J_6 \rightarrow J_2 \rightarrow J_1 \rightarrow J_3 \rightarrow J_4$

$M_3: J_6 \rightarrow J_3 \rightarrow J_4 \rightarrow J_2 \rightarrow J_5 \rightarrow J_1$

Mặc dù giá trị *makespan* là duy nhất, tuy nhiên có thể tồn tại nhiều lịch biểu khác nữa cùng có giá trị *makespan* này.

1.3. Các biến thể của bài toán JSSP

Trong thực tế, bài toán JSSP được hiệu chỉnh, mở rộng thành nhiều biến thể để phản ánh phù hợp với tình huống sản xuất:

- *Flow shop scheduling problem (FSSP)*: Tất cả công việc đi qua cùng một dãy máy theo thứ tự giống nhau, khác nhau ở thời gian xử lý.

- *Flexible job shop scheduling problem (FJSSP)*: Mỗi công đoạn có thể được gán cho một trong nhiều máy tương đương.
- *Open shop scheduling problem (OSSP)*: Thứ tự công đoạn của mỗi công việc có thể thay đổi, miễn là tuân thủ ràng buộc máy.
- *Parallel machine scheduling problem (PMSP)*: Nhiều máy song song (đồng nhất hoặc khác tốc độ), công việc có thể phân bổ cho bất kỳ máy nào.
- *Single machine scheduling problem (SMSP)*: Tất cả công việc xử lý tuần tự trên một máy duy nhất.
- *General job shop scheduling problem (GJSSP)*: Phiên bản tổng quát, cho phép số công đoạn khác nhau giữa các công việc.

1.4. Ứng dụng của bài toán JSSP

Bài toán JSSP có ứng dụng quan trọng trong nhiều lĩnh vực thực tiễn, đặc biệt trong sản xuất và dịch vụ có quy trình phức tạp, đòi hỏi sự phân bổ hợp lý các công việc trên nhiều máy và tài nguyên khác nhau. Một số ứng dụng tiêu biểu bao gồm:

- *Sản xuất công nghiệp*: JSSP được ứng dụng rộng rãi trong các ngành cơ khí, điện tử, dệt may, chế tạo ô tô,... nơi mỗi sản phẩm phải trải qua nhiều công đoạn gia công trên các máy chuyên biệt. Việc xây dựng lịch biểu tối ưu giúp giảm *makespan*, hạn chế tình trạng máy nhàn rỗi, từ đó nâng cao năng suất và giảm chi phí vận hành.
- *Quản lý xưởng và dây chuyền gia công*: JSSP hỗ trợ việc sắp xếp hợp lý thứ tự công việc trong xưởng, cân bằng tải giữa các máy, giảm tắc nghẽn tại các công đoạn. Đây là nền tảng cho các hệ thống quản lý sản xuất tiên tiến.
- *Dịch vụ*: JSSP được áp dụng trong các lĩnh vực dịch vụ có giới hạn tài nguyên, chẳng hạn như lập lịch cho phòng thí nghiệm (nhiều mẫu thử phải qua nhiều thiết bị đo/kiểm định), lập lịch bảo trì bảo dưỡng máy, hoặc trong bệnh viện khi mà bệnh nhân cần trải qua nhiều loại xét nghiệm, chẩn đoán hình ảnh, và thiết bị y tế khác nhau,...
- *Logistics và chuỗi cung ứng*: JSSP góp phần tối ưu hóa luồng công việc trong xử lý đơn hàng phức tạp, đặc biệt khi nhiều hoạt động phải chia sẻ tài nguyên chung và tuân thủ một trình tự thực hiện bắt buộc.

1.5. Khảo sát một số nghiên cứu liên quan bài toán JSSP

Các thuật toán giải bài toán lập lịch JSSP thường được chia thành ba nhóm: các thuật toán giải chính xác, các thuật toán heuristic, và các thuật toán metaheuristic.

Các thuật toán giải chính xác có vai trò quan trọng trong việc nghiên cứu lý thuyết hoặc dùng để kiểm định chất lượng các thuật toán giải gần đúng.

Các thuật toán heuristic và metaheuristic đang là các hướng nghiên cứu chủ đạo hiện nay nhờ tính hiệu quả và khả năng mở rộng đối với các bài toán JSSP có kích thước lớn.

1.5.1. Các thuật toán giải chính xác

Tập trung vào việc tìm nghiệm tối ưu bằng các kỹ thuật như nhánh cận, quy hoạch động. Carlier và Pinson đã áp dụng thuật toán nhánh cận giải JSSP [4]. Brucker đã có nhiều công trình mở rộng với thuật toán quy hoạch động [2],[3],[17]; trong khi Yamamoto đã kết hợp thuật toán nhánh cận và thuật toán quy hoạch động để giảm không gian tìm kiếm [25]. Tuy nhiên, do độ phức tạp tăng theo hàm mũ, các thuật toán giải chính xác chỉ phù hợp với các bài toán có kích thước nhỏ và trung bình [9].

1.5.2. Các thuật toán heuristic

Các thuật toán heuristic hướng tới việc tìm lời giải gần đúng khả thi trong thời gian có thể chấp nhận được, tiêu biểu là các luật ưu tiên (Shortest Processing Time - *SPT*, Longest Processing Time - *LPT*, Most Work Remaining - *MWKR*, First In First Out - *FIFO*) [3],[4]. Một hướng quan trọng khác là thuật toán nút cổ chai (Shifting Bottleneck - *SB*) do Adams et al. đề xuất [1], sau đó được cải tiến thành Shifting Bottleneck Modification (*SBM*) [7] và lai ghép với các thuật toán khác như thuật toán Simulated Annealing hay thuật toán di truyền. Mặc dù không đảm bảo tìm được giá trị tối ưu, nhưng các thuật toán này hiệu quả trên các bài toán có kích thước lớn.

1.5.3. Các thuật toán metaheuristic

Đây là hướng nghiên cứu chủ đạo hiện nay nhờ khả năng tìm kiếm lời giải có chất lượng tốt hơn trong thời gian chấp nhận được.

Các thuật toán tiêu biểu gồm thuật toán Simulated Annealing (Laarhoven et al. [10]), thuật toán Tabu Search (Laguna et al. [11], Nowicki và Smutnicki [15]), thuật toán di truyền (Nakano & Yamada [14], Nguyễn Hữu Mùi [16],...), các thuật toán bầy ong (Phạm et al. [18], Artificial Bee Colony (Yin et al. [26], Meenambigai R. [12])). Nhiều nghiên cứu lai của các thuật toán này nhằm nâng cao hiệu quả, như thuật toán Tabu Search lai với thuật toán Simulated Annealing [24],[28], thuật toán Tabu Search lai với thuật toán di truyền [13], Youssefi et al. [27].

2. Phát triển thuật toán bees giải bài toán JSSP

2.1. Thuật toán Bees cơ bản

Các thuật toán mô phỏng hành vi tìm kiếm thức ăn của loài ong mật đã được đề xuất và phát triển trong nhiều công trình tiêu biểu, chẳng hạn Karaboga & Basturk (2007) [6], Pham et al. (2005) [19], Xin-She Yang (2010) [23], Nguyen Duc Nghia et al. [20],[21],...

Thuật toán Bees cơ bản kết hợp ba chiến lược tìm kiếm thường gặp trong các thuật toán metaheuristic: tìm kiếm lân cận (neighborhood search), tìm kiếm ngẫu nhiên (random search), tìm kiếm sâu hơn ở những vùng tiềm năng (intensification).

Để việc tìm kiếm có hiệu quả, thuật toán cần đạt được sự cân bằng giữa khai thác cục bộ (exploitation) và khám phá toàn cục (exploration). Thuật toán Bees có khả năng thoát khỏi bẫy cực trị cục bộ và nhờ đó được kỳ vọng tìm ra nghiệm tối ưu toàn cục. Do tính linh

hoạt và khả năng mở rộng, thuật toán Bees được đánh giá là một trong những metaheuristic phù hợp để giải quyết các bài toán tối ưu tổ hợp khó [5],[19],[20].

Sơ đồ thuật toán Bees cơ bản.

Bước 1: Khởi tạo quần thể ban đầu với N cá thể ong (được tạo ngẫu nhiên hoặc bằng một hoặc một số heuristic)

Bước 2: Đánh giá độ thích nghi (fitness) của mỗi cá thể.

Bước 3: while (điều kiện dừng chưa thỏa {

- Trong N cá thể ong, chọn ngẫu nhiên ra p cá thể để thực hiện tìm kiếm lân cận ($p < N$). Trong số p cá thể được chọn này chọn ra h cá thể có độ thích nghi cao nhất.
- Cử thêm nep ong để thực hiện việc tìm kiếm lân cận cho h cá thể tốt nhất trên và nsp ong để thực hiện việc tìm kiếm lân cận cho $p - h$ cá thể được chọn còn lại. Đánh giá lại độ thích nghi cho các cá thể vừa được bổ sung.
- Sau khi thực hiện tìm kiếm lân cận, mỗi cá thể (gốc) có thể sinh thêm một hoặc một số lân cận. Trong số cá thể gốc và các cá thể lân cận của nó (có thể gọi là một vùng các cá thể) chọn ra một cá thể có độ thích nghi cao nhất đại diện cho vùng cá thể đó ở lần tiến hóa tiếp theo.
- Mỗi cá thể trong số $N - p$ cá thể không được chọn sẽ được thay thế bằng một cá thể ngẫu nhiên.
- Đánh giá độ thích nghi cho các cá thể được bổ sung. }

Bước 4: return cá thể tốt nhất trong quần thể ở thế hệ cuối cùng.

Danh sách các tham số chính của thuật toán bao gồm: N là số lượng cá thể ong được khởi tạo, p là số lượng cá thể được chọn trong số N cá thể, h là số lượng cá thể có chất lượng tốt nhất trong số p cá thể được chọn, $p - h$ là số lượng cá thể được chọn còn lại, $N - p$ là số lượng cá thể không được chọn, nep là số lượng ong cử đến h cá thể tốt nhất, nsp là số lượng ong cử đến $p - h$ cá thể được chọn còn lại.

2.2. Thuật toán Bees giải bài toán JSSP

2.2.1. Mã hóa lời giải bài toán JSSP

Một lời giải của bài toán JSSP được biểu diễn bằng một mảng một chiều có độ dài $n \times m$ phần tử, trong đó n là số công việc, m là số công đoạn của mỗi công việc, và $S[k]$ biểu thị chỉ số công việc tại vị trí thứ k trong mảng. Mỗi công việc xuất hiện đúng m lần trong mảng, tương ứng với số công đoạn cần được thực hiện theo đúng trình tự công nghệ. Thứ tự xuất hiện của các phần tử trong mảng đóng vai trò quyết định độ ưu tiên xử lý: khi một máy trở nên khả dụng, công đoạn có vị trí xuất hiện sớm hơn trong mảng sẽ được lựa chọn để thực hiện trước.

Áp dụng theo sơ đồ thuật toán Bees cơ bản; trong đó mỗi cá thể ong tương ứng với một lời giải, tức là một lịch trình sản xuất hoàn chỉnh. Độ thích nghi của một lời giải được đánh giá thông qua tổng thời gian hoàn thành các công việc, gọi là *makespan*.

2.2.2. Tính *makespan*

Độ thích nghi của lời giải được xác định bằng giá trị của hàm mục tiêu, cụ thể là *makespan*. Quá trình tính toán được thực hiện theo các bước sau.

Trước hết, toàn bộ các công đoạn được duyệt theo thứ tự mã hóa trong mảng lời giải.

Với mỗi công đoạn, ta xác định thời điểm sớm nhất có thể bắt đầu dựa trên hai yếu tố: thời điểm công đoạn trước đó của cùng công việc đã hoàn thành, và thời điểm máy yêu cầu là khả dụng.

Thời điểm kết thúc công đoạn được tính bằng cách cộng thời gian bắt đầu với thời gian xử lý, đồng thời cập nhật trạng thái máy cũng như công việc tương ứng.

Sau khi tất cả công đoạn đã được xử lý, *makespan* được xác định là thời điểm hoàn thành muộn nhất trong số tất cả các thời điểm hoàn thành các công việc. Mục tiêu tối ưu đặt ra là tìm một lịch biểu có giá trị *makespan* nhỏ nhất, đồng thời đảm bảo tuân thủ các ràng buộc về trình tự công nghệ và giới hạn rằng tại mỗi thời điểm, một máy chỉ có thể xử lý duy nhất một công đoạn.

2.2.3. Tạo lời giải ban đầu

Trong quá trình khởi tạo cá thể lời giải hoặc quần thể lời giải ban đầu, một số lượng lớn lời giải hợp lệ ban đầu được sinh ra bằng các phương pháp khác nhau nhằm đảm bảo tính đa dạng của quần thể.

Cách đơn giản nhất là khởi tạo ngẫu nhiên, vốn dễ thực hiện và có ưu điểm là tạo ra được nhiều lời giải khác biệt, giúp mở rộng khả năng khám phá không gian tìm kiếm.

Bên cạnh đó, các chiến lược heuristic cũng được sử dụng. Chẳng hạn, chiến lược SPT (Shortest Processing Time) sắp xếp các công việc theo thứ tự tăng dần của tổng thời gian xử lý, nhờ đó các công việc có thời gian xử lý ngắn hơn sẽ được ưu tiên trước, tạo ra những lịch trình có *makespan* ban đầu nhỏ. Để tăng thêm tính đa dạng, chiến lược SPT đảo ngược (Reversed SPT) được áp dụng bằng cách lấy thứ tự các công việc trong lời giải SPT và đảo ngược hoàn toàn.

Việc kết hợp giữa ngẫu nhiên và heuristic như vậy cho phép tạo ra nhiều lời giải khởi đầu vừa chất lượng vừa đa dạng.

2.2.4. Các chiến lược tìm kiếm lân cận

Phép hoán đổi

Một trong những chiến lược đơn giản trong quá trình tìm kiếm lân cận là phép hoán đổi vị trí giữa hai công việc trong chuỗi lời giải. Chiến lược này thực hiện chọn ngẫu nhiên hai vị trí khác nhau trong lời giải và hoán đổi trực tiếp hai công việc tại đó. Phép hoán đổi này

giữ nguyên số lần xuất hiện của mỗi công việc, do đó không làm mất tính hợp lệ của lời giải. Chiến lược này là hiệu quả khi cần đa dạng hóa không gian tìm kiếm.

Phép chèn

Chiến lược chèn sẽ chọn ngẫu nhiên một công việc trong chuỗi lời giải, loại bỏ nó khỏi vị trí ban đầu và chèn lại vào một vị trí khác trong chuỗi. Chiến lược này giúp có sự thay đổi nhỏ về mặt cấu trúc lời giải mà vẫn giữ nguyên số lượng công việc và đảm bảo tính hợp lệ nhờ quá trình giải mã thứ tự công đoạn.

Phép dời khối

Phép dời khối được triển khai để thực hiện thao tác chọn hai phần tử liên tiếp trong chuỗi lời giải (gọi là một khối), sau đó dời cả khối này đến một vị trí khác trong chuỗi. Phép biến đổi này giúp thay đổi cục bộ lớn hơn so với phép hoán đổi hoặc phép chèn, giúp thuật toán thoát khỏi các vùng tối ưu cục bộ hiệu quả hơn.

Tìm kiếm lời giải lân cận

Với thuật toán BeeJSS, mỗi cá thể trong quần thể đại diện cho một vùng tìm kiếm. Sau khi đánh giá độ thích nghi, thuật toán lựa chọn p cá thể tốt nhất và trong đó e cá thể có độ thích nghi cao nhất được xác định làm vùng ưu tú. Đối với mỗi vùng ưu tú, thuật toán phân bổ nep ong khai thác để thực hiện tìm kiếm cục bộ. Mỗi ong áp dụng ngẫu nhiên một trong ba phép biến đổi lân cận (phép hoán đổi, phép chèn, hoặc phép dời khối) để tạo ra lời giải mới. Trong số các lời giải sinh ra, lời giải có *makespan* nhỏ nhất sẽ được chọn làm đại diện của vùng đó cho thế hệ tiếp theo. Đối với $p-e$ vùng còn lại, thuật toán vẫn thực hiện khai thác cục bộ nhẹ hơn bằng cách phân bổ nsp ong cho mỗi vùng. Các ong này tạo lời giải lân cận đơn giản bằng một phép dời khối, sau đó chọn ra lời giải có độ thích nghi tốt nhất trong tập để đưa vào quần thể kế tiếp.

Tìm kiếm lời giải ngẫu nhiên

Trong giai đoạn khai phá ngẫu nhiên, $N-p$ ong được phân bổ vào các vùng tìm kiếm mới, với mục tiêu tăng cường tính đa dạng và khả năng khám phá của thuật toán. Đối với mỗi ong trong vùng này, thuật toán khởi tạo một số lời giải ngẫu nhiên độc lập. Trong số các lời giải được sinh ra, chỉ duy nhất cá thể có độ thích nghi tốt nhất được lựa chọn làm đại diện cho vùng và được đưa vào quần thể thế hệ kế tiếp.

2.2.5. Điều kiện dừng

Thuật toán BeeJSS áp dụng điều kiện dừng sau: trong một số lần lặp liên tiếp mà chất lượng lời giải tốt nhất không được cải thiện. Điều kiện dừng này vừa đảm bảo hiệu quả tính toán, vừa tránh việc thuật toán tiếp tục chạy khi không còn khả năng cải thiện chất lượng lời giải.

3. Thực nghiệm và đánh giá

3.1. Dữ liệu thực nghiệm cho bài toán JSSP

Trong các nghiên cứu về bài toán JSSP, nhiều hệ thống dữ liệu thực nghiệm đã được đề xuất như Fisher and Thompson, Lawrence, ORB, ABZ, YN, SWV, TA [29]. Các bộ dữ liệu

cho bài toán JSSP được xây dựng dựa trên những đặc trưng như số lượng công việc, số lượng máy, trình tự xử lý và thời gian xử lý.

Trong bài báo này, chúng tôi chọn hệ thống dữ liệu Lawrence để tiến hành thực nghiệm thuật toán BeeJSS.

Hệ thống dữ liệu Lawrence bao gồm 40 bộ dữ liệu từ LA01 đến LA40, với quy mô từ 10 đến 30 công việc. Hệ thống dữ liệu Lawrence được giới thiệu lần đầu tiên vào năm 1984 bởi S. Lawrence trong một báo cáo nội bộ tại Đại học Carnegie Mellon [29],[30].

Chúng tôi so sánh chất lượng lời giải của thuật toán BeeJSS với thuật toán nhánh cận, thuật toán ACA (năm 2015), thuật toán DDDQNPR (năm 2020); chất lượng của các thuật toán này chúng tôi lấy từ kho lưu trữ GitHub của tác giả Thomas Weise [30]. Trong đó, thuật toán DDDQNPR là đại diện cho hướng tiếp cận học sâu tăng cường, sử dụng mô hình Double Dueling Deep Q-Network kết hợp với mạng nơ-ron tích chập (CNN), thuật toán ACA là sự kết hợp giữa thuật toán kiến (ACO) và tìm kiếm cục bộ. Chúng tôi cũng so sánh thuật toán BeeJSS với thuật toán NHGA (năm 2013) [16]; đây là hướng tiếp cận theo thuật toán di truyền giải bài toán JSSP.

3.2. Môi trường thực nghiệm

Chúng tôi đã cài đặt thuật toán BeeJSS bằng ngôn ngữ C++, sử dụng trình biên dịch Microsoft Visual Studio 2025, và chạy trên máy tính với cấu hình: Hệ điều hành Microsoft Windows 11 Pro (64 bit), RAM 16GB, bộ xử lý 13th Gen Intel(R) Core(TM) i5-1340P 1.90 GHz [8].

Đối với mỗi bộ dữ liệu, chúng tôi chạy thực nghiệm 10 lần, giá trị *makespan* tốt nhất trong các lần chạy được ghi nhận làm chất lượng lời giải. Thời gian chạy được ghi nhận bằng trung bình cộng thời gian của 10 lần chạy. Các giá trị tham số thực nghiệm tiếp theo mà chúng tôi sử dụng trong thuật toán này bao gồm: Số vòng lặp thuật toán: 3500, số lần lặp liên tiếp mà giá trị tốt nhất không được cải thiện: 1250, $n=120$, $p=80$, $h=20$, $nep=5$, $nsp=3$.

3.3. Kết quả thực nghiệm

Kết quả thực nghiệm thuật toán BeeJSS được trình bày trong Bảng 1 có cấu trúc như sau: Cột **Test** là tên của các bộ dữ liệu [29],[30]; các cột ***n***, ***m*** lần lượt tương ứng với số công việc và số máy trong mỗi bộ dữ liệu. Các cột tiếp theo của Bảng 1 lần lượt là: **LB** (Lower bound of the optimal value - cận dưới của giá trị tối ưu); đây là kết quả của thuật toán nhánh cận (Branch and Bound), cột **BeeJSS** ghi nhận chất lượng lời giải của thuật toán BeeJSS, cột Time of BeeJSS (s) ghi nhận thời gian chạy của thuật toán BeeJSS (đơn vị là giây), cột **NHGA** ghi nhận chất lượng lời giải tiếp cận theo thuật toán di truyền được công bố năm 2013 [16], các cột **ACA** và **DDQNPR** ghi nhận chất lượng của giải của các metaheuristic được công bố bởi tác giả Thomas Weise [30].

Bảng 1: Kết quả thực nghiệm của thuật toán BeeJSS trên các bộ dữ liệu Lawrence

Test	n	m	LB	BeeJSS	Time of BeeJSS (s)	NHGA	ACA	DDDQNPR
LA01	10	5	666	666	135	666	666	666
LA02	10	5	655	655	136	655	655	655
LA03	10	5	597	597	136	597	597	597
LA04	10	5	590	590	136	590	590	609
LA05	10	5	593	593	133	593	593	593
LA06	15	5	926	926	166	926	926	926
LA07	15	5	890	890	168	890	890	890
LA08	15	5	863	863	179	863	863	863
LA09	15	5	951	951	177	951	951	951
LA10	15	5	958	958	170	958	958	958
LA11	20	5	1222	1222	224	1222	1222	1222
LA12	20	5	1039	1039	209	1039	1039	1047
LA13	20	5	1150	1150	209	1150	1150	1151
LA14	20	5	1292	1292	207	1292	1292	1292
LA15	20	5	1207	1207	212	1207	1207	1221
LA16	10	10	945	945	573	945	946	980
LA17	10	10	784	784	528	794	789	799
LA18	10	10	848	848	530	848	848	859
LA19	10	10	842	842	896	842	842	872
LA20	10	10	902	902	930	907	902	924
LA21	15	10	1046	1055	1528	1055	1050	1162
LA22	15	10	927	927	1435	927	938	1021
LA23	15	10	1032	1032	1602	1032	1032	1053
LA24	15	10	935	941	1633	940	959	1029
LA25	15	10	977	981	1488	978	977	1067
LA26	20	10	1218	1218	1118	1218	1218	1327
LA27	20	10	1235	1256	2087	1270	1242	1397
LA28	20	10	1216	1222	1520	1216	1227	1386
LA29	20	10	1152	1204	1235	1190	1177	1323

Test	n	m	LB	BeeJSS	Time of BeeJSS (s)	NHGA	ACA	DDDQNPR
LA30	20	10	1355	1355	1566	1355	1355	1417
LA31	30	10	1784	1784	1326	1784	1784	1854
LA32	30	10	1850	1850	1320	1850	1850	1900
LA33	30	10	1719	1719	1302	1719	1719	1782
LA34	30	10	1721	1721	1306	1721	1725	1880
LA35	30	10	1888	1888	1323	1888	1888	1941
LA36	15	15	1268	1291	1634	1275	1275	1355
LA37	15	15	1397	1429	2458	1415	1412	1540
LA38	15	15	1196	1230	1078	1210	1196	1348
LA39	15	15	1233	1251	1119	1240	1240	1357
LA40	15	15	1222	1247	1747	1235	1222	1336

3.4. Đánh giá kết quả thực nghiệm

3.4.1. So sánh thuật toán BeeJSS với các thuật toán NHGA, ACA, DDDQNPR

Từ Bảng 1 trên, ta thấy rằng thuật toán BeeJSS tốt hơn thuật toán NHGA trên 3 bộ dữ liệu, chiếm 7.5%; thuật toán BeeJSS cho kết quả tương đương thuật toán NHGA trên 28 bộ dữ liệu, chiếm 70.0%; thuật toán BeeJSS cho kết quả kém hơn thuật toán NHGA trên 9 bộ dữ liệu, chiếm 22.5%.

Thuật toán BeeJSS tốt hơn thuật toán ACA trên 6 bộ dữ liệu, chiếm 15.0%; thuật toán BeeJSS cho kết quả tương đương thuật toán ACA trên 25 bộ dữ liệu, chiếm 62.5%; thuật toán BeeJSS cho kết quả kém hơn thuật toán ACA trên 9 bộ dữ liệu, chiếm 22.5%.

Thuật toán BeeJSS tốt hơn thuật toán DDDQNPR trên 29 bộ dữ liệu, chiếm 72.5%; thuật toán BeeJSS cho kết quả tương đương thuật toán DDDQNPR trên 11 bộ dữ liệu, chiếm 27.5%.

Chúng tôi cũng tóm lược nội dung mô tả này trong Bảng 2 sau đây.

Bảng 2: So sánh chất lượng lời giải của thuật toán BeeJSS với các thuật toán NHGA, ACA, DDDQNPR (đơn vị là bộ dữ liệu)

BeeJSS	NHGA		ACA		DDDQNPR	
	Số lượng	%	Số lượng	%	Số lượng	%
Tốt hơn	3	7.5	6	15.0	29	72.5
Tương đương	28	70.0	25	62.5	11	27.5
Kém hơn	9	22.5	9	22.5	0	0.0

3.4.2. So sánh thuật toán giải chính xác với các thuật toán NHGA, ACA, DDDQNPR

Khi so sánh chất lượng các thuật toán BeeJSS, NHGA, ACA, DDDQNPR với thuật toán nhánh cận (cột LB ở Bảng 1) thì thuật toán BeeJSS kém hơn LB không quá 5%, thuật toán NHGA kém hơn LB không quá 3%, thuật toán ACA kém hơn LB không quá 3% và thuật toán DDDQNPR kém hơn LB không quá 15%.

3.4.3. Về thời gian thực nghiệm thuật toán BeeJSS

Tổng thời gian thực nghiệm một lần chạy cho cả 40 bộ test là 35879 giây. Do mỗi bộ test chúng tôi thực nghiệm 10 lần, nên tổng thời gian thực nghiệm của thuật toán BeeJSS tương đương với khoảng 100 giờ chạy máy tính.

4. Kết luận

Trong bài báo này chúng tôi đã đề xuất một thuật toán metaheuristic cụ thể để giải quyết bài toán lập lịch Job Shop, chúng tôi đặt tên là BeeJSS. Đóng góp mới của chúng tôi trong bài báo này là xây dựng thuật toán BeeJSS trên nền thuật toán Bees cơ bản để giải quyết bài toán JSSP. Chúng tôi đã thực nghiệm thuật toán BeeJSS trên 40 bộ dữ liệu Lawrence, kết quả thực nghiệm cho thấy rằng thuật toán BeeJSS cho chất lượng lời giải cạnh tranh được với một số thuật toán metaheuristic hiện biết. Bên cạnh đó, chúng tôi đã khảo sát về một số hướng tiếp cận giải bài toán JSSP cũng như giới thiệu một số hệ thống dữ liệu thực nghiệm chuẩn cho bài toán JSSP.

Các kết quả đạt được trong bài báo này góp phần mở rộng ứng dụng của thuật toán metaheuristic nói chung và thuật toán Bees nói riêng trong việc giải các bài toán tối ưu tổ hợp.

Chúng tôi hy vọng rằng bài báo này cung cấp thông tin hữu ích cho những nghiên cứu tiếp theo về bài toán JSSP.

Lời cảm ơn: Bài viết thuộc đề tài nghiên cứu khoa học “Khảo sát một số thuật toán metaheuristic giải bài toán lập lịch Job Shop” có mã số CS.2025.B3.034 của Trường Đại học Sài Gòn. Nhóm tác giả trân trọng cảm ơn Trường Đại học Sài Gòn đã tài trợ cho Đề tài.

TÀI LIỆU THAM KHẢO

- [1]. Adams J, Balas E, Zawack D, “*The shifting bottleneck procedure for Job Shop scheduling*”, Management Science, vol. 34, no. 3, pp. 391–401, 1988.
- [2]. Brucker P, “*A branch and bound algorithm for the Job Shop scheduling problem*”, Elsevier, 1992.
- [3]. Brucker P, “*Scheduling algorithms*”, Springer, 2007.
- [4]. Carlier J, Pinson E, “*An algorithm for solving the Job Shop problem*”, Management Science, pp.164-176, 1989.

- [5]. Dervis Karaboga, Bahriye Basturk, “*On the performance of artificial bee colony algorithm*”. ELSEVIER, Applied Soft Computing 8, pp.687–697, 2008.
- [6]. Dervis Karaboga, Bahriye Basturk, “*A powerful and efficient algorithm for numerical function optimization: Artificial bee colony algorithm*”, Springer, 2007.
- [7]. Dauzère-Pérès, S. & Lasserre, J. B. A, “*Modified shifting bottleneck procedure for Job Shop scheduling*”, International Journal of Production Research, 31(4), pp.923–932, 1993.
- [8]. Đỗ Ngọc Nhung, “*Nghiên cứu và xây dựng thuật toán bees giải bài toán Job Shop scheduling*” (Luận văn thạc sỹ chuyên ngành khoa học máy tính), Trường Đại học Sài Gòn, 2025.
- [9]. J. A. S. Gromicho et al., “*Solving the job-shop scheduling problem optimally by dynamic programming*”, Computers & Operations Research, vol. 39, pp. 2968–2977, 2012.
- [10]. Laarhoven P. J. M. et al., “*Job Shop scheduling by simulated annealing*”, Openations Research, 1992.
- [11]. Laguna M, Glover F, “*Integrating target analysis and tabu search for improved scheduling system*”, Expert Systems with Application, 1993.
- [12]. Meenambigai R, “*Job Shop scheduling using bee colony optimization algorithm*”, ICCIAMR, 2014.
- [13]. Moraglio A et al., “*Genetic local search for Job Shop scheduling problem*”.
- [14]. Nakano R, Yamada T, “*Conventional genetic algorithm for Job Shop problem*”, In proceedings of international conference on genetic algorithms, 1991.
- [15]. Nowicki E, Smutnicki C, “*A fast taboo search algorithm for the job shop problem*”, Management Science, 1996.
- [16]. Nguyễn Hữu Mùi, “*Thuật toán và các bài toán lịch biểu*” (Luận án tiến sĩ công nghệ thông tin), Trường Đại học Công nghệ, Đại học Quốc gia Hà Nội, 2013.
- [17]. P. Brucker, B. Jurisch & B. Sievers, “*A branch and bound algorithm for the Job Shop scheduling problem*”, Discrete Applied Mathematics, vol. 49, pp. 107–127, 1994.
- [18]. Pham D. T. et al., “*Using the Bees Algorithm to schedule jobs for a machine*”, Manufacturing Engineering Centre, Cardiff University, 2007.
- [19]. Pham D.T. et al., “*The bees algorithm - A novel tool for complex optimisation problems*”. ELSEVIER, Proceedings of IPROMS 2006 Conference, Cardiff University, pp.454–461, (2006).
- [20]. Phan Tân Quốc, Nguyễn Đức Nghĩa, “*Thuật toán bầy ong giải bài toán cây khung với chi phí định tuyến nhỏ nhất*”. ICTFIT (Trường Đại học Khoa học Tự nhiên, Đại học Quốc gia TpHCM), Tuyển tập công trình nghiên cứu Công nghệ thông tin & Truyền thông, Nhà xuất bản Khoa học Kỹ thuật, pp.73-81, 2012.

- [21]. Phan Tấn Quốc, Nguyễn Đức Nghĩa, “*Thuật toán bầy ong giải bài toán cây khung với chi phí định tuyến nhỏ nhất*”, Tạp chí Tin học và Điều khiển học, T.29, S3, 2013, pp.265-276, 2013.
- [22]. Pinedo M, “*Scheduling: Theory, Algorithms, and Systems*”, Springer, 2016.
- [23]. Xin-She Yang (2010). Nature-inspired metaheuristic algorithms. LUNIVER Press, pp.53–62.
- [24]. Yamada T, Nakano R, “*Job Shop scheduling by simulated annealing combined with deterministic local search*”, Metaheuristics International Conference, 1995.
- [25]. Yamamoto K, “*Solving the Job Shop scheduling problem optimally by dynamic programming*”, Academia.edu, 2022.
- [26]. Yin B. Y, Li L, and Zhou H. X, “*An efficient Job Shop scheduling algorithm based on artificial bee colony*”, In Proceedings of the 8th World Congress on Intelligent Control and Automation, 2010.
- [27]. Youssefi M. et al., “*Bottom-up variants for the Job Shop scheduling problem*”, Simulation and Modeling Methodologies, Technologies and Applications, 2024.
- [28]. Zhang C. et al., “*A very fast TS/SA algorithm for the Job Shop scheduling problem*”, Computers and Operations Research, 2008.
- [29]. https://www.eii.uva.es/elena/Elena%20Perez%20Vazquez_archivos/files_optimaJSSP/jobshop1.txt
- [30]. https://github.com/thomasWeise/jsspInstancesAndResults/blob/master/data-raw/results/all_results.txt

Ngày nhận bài: 03/10/2025

Ngày chấp nhận đăng: 18/3/2026