SỬ DỤNG GIẢI THUẬT TỐI ƯU HÓA RỪNG CÂY RỜI RẠC CHO BÀI TOÁN LẬP LỊCH CÁC CÔNG VIỆC ĐỘC LẬP TRONG LƯỚI TÍNH TOÁN VỚI TÌM KIẾM CỤC BỘ

ĐỖ VĨNH TRÚC*

TÓM TẮT

Lưới tính toán (Computational Grid-CG) là bài toán mới xuất hiện gần đây. Việc lập lịch (scheduling) với các công việc độc lập (independent jobs) trên CG với mục tiêu cực tiểu makespan là bài toán khó nhưng hấp dẫn. Đề tài này giới thiệu thuật toán tối ưu hóa rừng cây (Forest Optimization Algorithm – FOA) [5] có hiệu chỉnh và áp dụng để giải quyết bài toán lập lịch các công việc độc lập trên lưới tính toán với mục tiêu cực tiểu hóa makespan (thời gian bắt đầu và kết thúc công việc). Kết quả cho thấy FOA áp dụng tốt cho việc giải bài toán tối ưu hóa trên.

Từ khóa: giải thuật tối ưu hóa rừng cây, lưới tính toán, công việc độc lập, lập lịch, makespan.

ABSTRACT

Using discrete forest optimization algorithm for independent jobs scheduling on computational grids with local search

Computational Grid (CG) is a new problem that has appeared recently. Independent jobs scheduling on CG with the goal of minimizing makespan is a very difficult but fascinating problem. This topic introduces hybrid FOA (Forest Optimization Algorithm) [5] to solve the independent jobs scheduling on CG with the goal of minimizing makespan. The results show that FOA is also a good algorithm for solving the optimization problem.

Keywords: FOA, Computational grid, Independent job, Scheduling, Makespan.

1. Giới thiệu

Một CG là một hệ tính toán phân tán theo địa lí bao gồm một tập hợp các tài nguyên máy tính đa dạng, quy mô rộng lớn và độc lập [8], [15], [4], [2], chúng được nối kết với nhau bởi các mạng băng thông cao [3]. Việc chia sẻ các công việc tính toán là một ứng dụng chính của tính toán lưới. Trong một CG, các nguồn tài nguyên năng động, đa dạng và có thể được thêm vào và rút ra bất kì lúc nào. CG được coi là một tiếp cận hiệu quả để giải quyết các ứng dụng của thế giới thực phân tán, quy mô lớn [12]. Lập điều độ trong môi trường CG, có nghĩa là phân bổ công việc cho các tài nguyên

* ThS, Đại học Quốc tế, ĐHQG TPHCM; *Email: dvtruc@gmail.com*

trên tính toán lưới, đó là công việc rất quan trọng và nhiệm vụ tính toán khó khăn thậm chí ngay cả khi các công việc độc lập nhau [15] do yêu cầu thực tế.

Xét bài toán lập lịch trên tính toán lưới cho các công việc độc lập [12], [9]. Với một số lượng n công việc (job) và m máy móc (machine) cho trước, mục tiêu là tìm ra giải pháp tối ưu để phân bổ công việc cho các máy nhằm cực tiểu hóa makespan C_{max}=max{Ci}, với Ci là thời gian hoàn thành của máy i=1,…m. Trong bài toán này, một công việc chỉ có thể được xử lí chỉ trên một máy và một máy chỉ có thể xử lí một công việc tại một thời điểm nào đó. Các giả định là các công việc độc lập nhau và không có sự ưu tiên.

Braun và cộng sự [13], so sánh 11 heuristics cho lập lịch tính toán lưới (Grid Computating Scheduling- GCS) với các công việc độc lập. Một tập dữ liệu lớn [13] đã được phát triển dựa trên ma trận thời gian kì vọng cho tính toán (Expected Time to Compute-ETC) nhằm đánh giá hiệu suất của heuristics đã đề xuất. Từ thực nghiệm dựa trên GA, [13] cung cấp hiệu suất tốt nhất trong hầu hết các trường hợp, mà trong đó heuristic Min-Min là một phương pháp tốt để nhanh chóng tạo ra một giải pháp với một makespan ngắn hợp lí. Hai tác giả Page và Naughton [10] đã đề xuất một phương pháp GA khác cho GCS có sử dụng một danh sách heuristic đã lập lịch để khởi tạo ra một tổng thể ngẫu nhiên ban đầu khá tốt. Các toán tử đột biến trong đề xuất này được chuyên biệt hóa nhằm cải thiện hiệu suất GA. Ritchie và Levine [11] đã phát triển giải thuật tối ưu hóa đàn kiến lai (Hybrid ACO) để lập lịch trên tính toán lưới. Tìm kiếm Tabu [6] được dùng để hoàn thiện các giải pháp đạt được bằng ACO. Các ACO lai tạo được đánh giá tốt hơn các phương pháp GA và heuristics khác. Thuật toán cellular memetic (Cellular Memetic Algorithm-CMA) của Xhafa và cộng sự [14] đã giảm thiểu cả makespan và flowtime. Heuristics cục bộ khác cũng đã được kiểm tra trong bài báo [14] này. Kết quả cho thấy CMA là một cách tiếp cận hiệu quả cho GCS. Xhafa và cộng sự [15] đã phát triển một phương pháp tìm kiếm Tabu mới cho GCS. Chiến lược đa dạng hóa được chuyên biệt khác cũng đã được xem xét trong phương pháp tìm kiếm Tabu nhằm nâng cao hiệu quả của nó. Phương pháp này không những nhanh hơn so với giải thuật như Tabu Search [6], ACO lai [11], và CMA [14] mà còn cung cấp các lời giải tốt hơn. Các heuristics khác cũng được đề xuất cho GCS như giải thuật luyện kim (Simulated Annealing-SA) [13], GA đấu tranh (Struggle GA-SGA) [14], sufferage, GA lai.

Gần đây, một số phương pháp PSO [8], [7]đã được đề xuất để giải bài toán GCS với công việc độc lập. Liu và các cộng sự [8] đề xuất một phương pháp PSO liên tục (Continuous PSO-CPSO) cho GCS. Trong phương pháp này, vị trí và tốc độ của một cá thể được biểu diễn như là ma trận số thực (n×m). Để xây dựng một lịch trình vị trí, đầu tiên ma trận phải được chuyển đổi thành một lịch trình bằng cách gán cho mỗi công việc vào máy mà có giá trị chuẩn hóa cao nhất trong cột tương ứng của công việc đó trong ma trận vị trí. Ma trận vị trí trong phương pháp này được coi là một ma trận mờ trong đó một yếu tố đại diện cho mức độ thành viên của công việc và máy tương ứng.

Các thực nghiệm cho thấy rằng CPSO là tốt hơn so với SA và GA. Izakian cùng các cộng sự [7] đưa ra một cách tiếp cận PSO rời rạc (Discrete PSO-DPSO) để lập lịch lưới. Trong phương pháp [7], các cá thể được đại diện là một ma trận vận tốc số thực cho công việc và máy. Giá trị tại một vị trí công việc/máy được cho trước trong ma trận xác định công việc này có liên quan đến máy khác như thế nào. Vị trí của một cá thể là một mảng các số nguyên, ở đó mỗi vị trí trong danh sách là một công việc và các số nguyên ở vị trí đó là các máy mà công việc được phân công đến. Trong giai đoạn cập nhật cho cá thể này, vận tốc của cá thể được thay đổi hướng về phía vận tốc của cá thể tốt nhất toàn cục và tốc độ cá thể tốt nhất dựa trên các thành phần xã hội công nhận và tự nhận, tương ứng. Khi vận tốc được cập nhật, vị trí của cá thể được thiết lập để mỗi công việc được giao cho các máy có giá trị cao nhất trong cột đó của công việc của ma trận vận tốc. Thí nghiệm [7] cho thấy rằng DPSO là tốt hơn so với chẩn đoán khác như Min-Min, GA, và ACO lai. [11]

2. Giải thuật tối ưu hóa rừng cây **[5]**

2.1. Giới thiệu

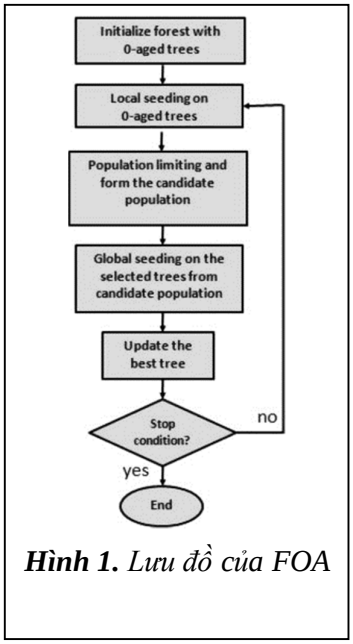
Giải thuật FOA bao gồm ba giai đoạn chính:

1- Gieo mầm cục bộ.

2- Giới hạn quần thể.

3- Gieo mầm toàn cục.

Sơ đồ của thuật toán như sau.

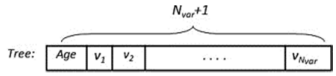


Hình 1. *Lưu đồ của FOA*

FOA bắt đầu với quần thể ban đầu của cây. Mỗi cây đại diện cho một giải pháp khả dĩ của bài toán. Một cây gồm có giá trị các biến và độ tuổi (Age). Ban đầu Age của một cây được gán bằng 0. Sau khi được khởi tạo, các cây con sẽ được nẩy mầm từ những cây có tuổi 0 và chúng được thêm vào rừng. Sau đó, tất cả các cây cũ tăng thêm 1 tuổi.

2.1.1. Khởi tạo cây

Tạo các cây có Age=0. Hình 2 cho thấy một cây có Nvar chiều, là các giá trị của các biến và Age là độ tuổi của cây.



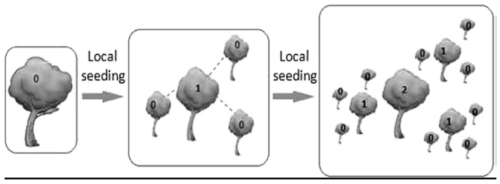
Hình 2. *Biểu diễn lời giải của FOA*

Một cây được coi là một mảng 1x(nvar+1), với nvar là số chiều của bài toán và Age đại diện cho tuổi của cây.Tree=[Age,v_1,v_2,v_3…,v_{nvar}]

Tuổi tối đa cho phép của một cây là một tham số được xác định trước và được đặt tên là tuổi thọ (life time). Tuổi thọ được xác định vào lúc bắt đầu của thuật toán. Khi Age một cây đạt đến "tuổi thọ", cây bị loại khỏi rừng và được thêm vào danh sách cây ứng viên. Nếu tham số này lớn, mỗi lần lặp của thuật toán chỉ làm tăng độ tuổi của cây và rừng sẽ chứa nhiều cây già nua mà không tham gia vào các giai đoạn gieo mầm cục bộ. Nếu tham số này nhỏ thì cây sẽ già đi rất sớm và chúng sẽ bị bỏ qua ở giai đoạn đầu của khởi tạo. Vì vậy, tham số này sẽ cung cấp một cơ hội tốt cho tìm kiếm cục bộ.

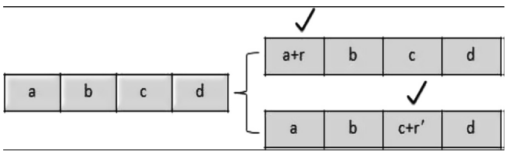
2.1.2. Gieo mầm cục bộ

Gieo mầm cục bộ của cây trong bài cố gắng mô phỏng quá trình tạo cây con của thiên nhiên. Chỉ có cây với Age= 0 mới cho nảy sinh cây non thành cây láng giềng và tạo thành rừng. Hai lần tạo cây non được minh họa như hình 3. Sau một lần như vậy, cây có Age= 0 sẽ thành 1 và cây con có Age=0, trong khi đó cây già hơn sẽ thêm 1 tuổi.



Hình 3. *Ví dụ của gieo mầm cục bộ trên một cây cho 2 lần lặp, với LSC=3.*

Các cây vượt quá "tuổi thọ" sẽ được xem xét. Nếu một cây là đầy tiềm năng, Age của cây đó được gán về 0, và đưa nó thành láng giềng tốt vào rừng. Ngược lại, các cây không hứa hẹn sẽ chết. Số cây non được tạo ra từ 1 cây nào đó do tham số "Sự thay đổi gieo mầm cục bộ" (Local Seeding Changes-LSC) quyết định. Giá trị của tham số LSC này là 3 trong như trong hình 3. Kết quả, thực hiện gieo mầm cục bộ trên một cây với Age 0 sẽ nảy mầm 3 cây non. Tham số này nên được xác định tùy theo kích thước của bài toán. Gieo mầm cục bộ mô phỏng tìm kiếm cục bộ cho thuật toán này. Hình 4 minh họa một ví dụ về công đoạn gieo mầm cục bộ cho bài toán thực trong không gian liên tục 4 chiều và ở đây giá trị của "LSC" được coi là 2. Nếu (a+r) hay (c+r') nằm ngoài giới hạn dưới và trên của biến liên quan nó sẽ được điều chỉnh để để thuộc trong giới hạn cho phép.



Hình 4. *Một ví dụ gieo mầm cục bộ cho không gian liên tục, r và r' thuộc [-Δx, Δx]*

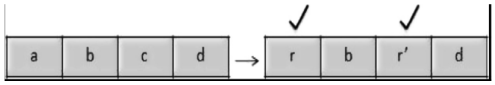
2.1.3. Giới hạn quần thể

Số cây trong rừng phải được giới hạn để ngăn chặn sự mở rộng vô hạn của rừng bằng tham số "giới hạn diện tích" ("area limit"). Xếp hạng các cây từ tốt đến xấu, nếu

số lượng cây lớn hơn "giới hạn diện tích", cây tốt sẽ được giữa lại, cây xấu hơn sẽ bị loại ra khỏi rừng và thêm vào nhóm cây ứng viên. Các cây được khởi tạo ban đầu bằng với "giới hạn diện tích". Sau khi hạn chế số cây của rừng, giai đoạn gieo mầm toàn cục được thực hiện trên tỉ lệ phần trăm của nhóm ứng viên sẽ được mô tả sau.

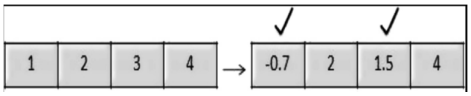
2.1.4. Gieo mầm toàn cục

Động vật, chim trong rừng ăn hạt và trái cây của những cây này và làm hạt giống của cây được phát tán trong toàn bộ khu rừng và kết quả là môi trường sống của cây trở nên rộng hơn. Đó là giai đoạn gieo mầm toàn cục. Giai đoạn này được thực hiện trên tỉ lệ phần trăm xác định qua một tham số có tên là "tỉ lệ lan truyền" (tranfer rate). Đầu tiên, các cây từ nhóm ứng viên được lựa chọn theo "tỉ lệ lan truyền". Sau đó, một số biến của mỗi cây được lựa chọn ngẫu nhiên. Giá trị của mỗi biến được lựa chọn sẽ được thay bằng một giá trị ngẫu nhiên. Bằng cách này, không gian tìm kiếm toàn cục được xem xét và không bị hạn chế. Kết quả là cây có tuổi 0 được thêm vào rừng. Số lượng các biến có giá trị sẽ bị thay đổi là một tham số của thuật toán và được đặt tên là "Thay đổi gieo mầm toàn cục" (Global Seeding Changes-GSC). Hình 5 là một ví dụ về thực hiện công đoạn gieo hạt toàn cầu cho một cây trong không gian liên tục. Trong hình 6, GSC= 2, như vậy 2 biến được lựa chọn ngẫu nhiên và giá trị của chúng được thay bằng 2 giá trị được tạo ra ngẫu nhiên khác như r và r'.



Hình 5. *Ví dụ gieo mầm toàn cục trên 1 cây*

Ví dụ trong hình 6 cho thấy giá trị của tham số GSC=2 và phạm vi là [-5,5]. Kết quả là, giá trị của 2 biến lựa chọn ngẫu nhiên thay bằng 2 giá trị khác trong phạm vi [-5, 5] như -0.7 và 1.5.



Hình 6. *Ví dụ bằng số của gieo mầm toàn cục với GCS=2*

2.1.5. Cập nhật giá trị tối ưu

Sau khi phân loại cây theo giá trị phù hợp của chúng, cây có giá trị phù hợp cao nhất được chọn làm cây tốt nhất, Age của cây tốt nhất sẽ được thiết lập về 0. Như vậy cây tốt nhất có thể đạt tối ưu hóa cục bộ trong giai đoạn gieo mầm cục bộ.

2.1.6. Điều kiện dừng

Ba điều kiện dừng có thể được áp dụng: 1) số bước lặp được xác định trước; 2) không có sự thay đổi trong giá trị tối ưu sau một số lần lặp; 3) đạt đến cấp độ nhất định chính xác. Các giai đoạn chính của FOA được hiển thị như mã giả ở phần sau.

2.2. Giải thuật FOA

Giải thuật FOA("life time", LSC, GSC, "tranfer rate", "area limit")

Nhập: giá trị tối ưu hoặc gần tối ưu của hàm mục tiêu f(x).

Xuất: trả về giá trị gần tối ưu hay tối ưu của f(x).

1.Khởi tạo rừng với cây tạo ngẫu nhiên

Mỗi cây là một vec tơ x có (n+1) chiều, ứng với bài toán n chiều, $x=(age,x_1,x_2,\ldots,x_n)$.

Age=0 lúc khởi tạo.

2. Trong khi điều kiện dùng chưa thỏa

2.1. Thực hiện gieo mầm cục bộ với cây có Age=0

For i=1:LSC

Chọn ngẫu nhiên 1 biến của cây đang xét.

Thêm một giá trị dx thuộc [-Δx, Δx] vào biến ngẫu nhiên trên.

Tăng tuổi của các cây lên 1 tuổi ngoại trừ cây con mới mọc.

2.2. Giới hạn quần thể

Loại bỏ cây có tuổi vượt quá "Life time" và đưa chúng vào nhóm cây ứng viên.

Sắp xếp cây theo độ phù hợp fitness.

Loại bỏ các cây nằm ngoài "area limit" có độ phù hợp thấp nhất và đưa chúng vào nhóm cây ứng viên.

2.3. Gieo mầm toàn cục

Chọn tỉ lệ "tranfer rate" từ nhóm cây ứng viên.

Với mỗi cây ứng viên trên

Chọn ngẫu nhiên "GCS" biến

Thay giá trị của mỗi biến với giá trị ngẫu nhiên trong miền giới hạn và tạo cây con với Age=0, sau đó đưa vào rừng.

2.4. Cập nhật giá trị tối ưu

Sắp xếp cây theo fitness.

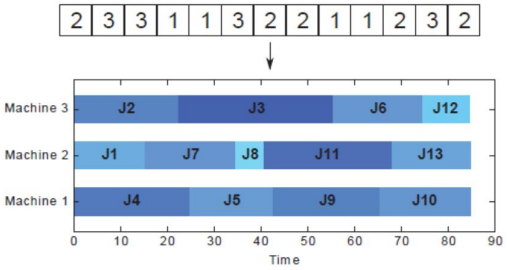
Gán trị Age=0 cho cây tốt nhất.

3. Trả về giá trị tốt nhất

3. Giải thuật đề nghị

3.1. Mô tả bài toán

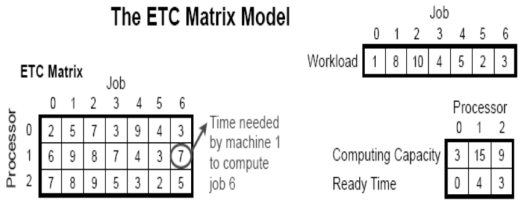
Trong giải thuật đề nghị này, mỗi cây x=(Age,x_1,…,x_n) là một mảng số nguyên. Mỗi phần tử x_j đại diện cho công việc j được gán cho một máy cụ thể nào đó. Bất kì công việc j nào cũng có thể được xử lí trên máy nào cũng được. Vì các công việc là độc lập và mục tiêu của bài toán là cực tiểu hóa makespan nên lời giải của bài toán là các công việc không bị ràng buộc theo một thứ tự nào. Giả sử có 13 công việc j_1,j_2, …. j_{13} với thời gian cần hoàn thành từng công việc tương ứng là p_1,p_2,…,p_{13} trên máy có công suất đơn vị. Chúng được bố trí thực hiện trên 3 máy m_1, m_2, m_3 với công suất tương ứng từng máy là e_1, e_2, và e_3. Công việc j_1 với p_1 thực trên máy m_1, m_2 hay m_3 sẽ tốn thời gian tương ứng là $p_{11}=p_1/e_1$, $p_{12}=p_1/e_2$ hoặc $p_{13}=p_1/e_3$. Giải thuật này sẽ tìm cách bố trí các công việc trên các máy sao cho tổng thời gian làm việc trên các máy là nhỏ nhất. Xét một ví dụ kết quả như hình sau, x_1=2, x_2=3 có nghĩa công việc 1, 2 được xử lí trên máy 2, và 3 với thời gian xử lí là p_{12} và p_{23} và được biểu diễn như sơ đồ trong hình 7.



Hình 7. *Biểu diễn lời giải của FOA đề xuất*

3.2. Mô hình Expected Time to Compute-ETC

Để việc gán công việc cho máy phù hợp cho bài toán lập lịch, mô hình ma trận ETC [1] được sử dụng cho mục đích này. Xét một tập các công việc độc lập để phân bổ nguồn lực, mỗi ETC[j][m] chỉ ra thời gian ước lượng để hoàn thành công việc j trên máy m. Mỗi hàng của ma trận lưu trữ thời gian ước lượng để hoàn thành từng công việc trên 1 máy. Mỗi cột cho biết thời gian dự tính hoàn thành 1 công việc trên từng máy. Giá trị ETC[j][m] bằng khối lượng công việc của công việc j cho khả năng làm việc của máy m. Giả định rằng khối lượng công việc là cho trước dựa các thông số kĩ thuật cung cấp từ người dùng, từ dữ liệu quá khứ, hoặc dự đoán.



Hình 8. *Mô hình ma trận ETC*

Hình trên là ví dụ về ma trận ETC (njob x nmachine), với ETC[j][m] là giá trị của thời gian dự kiến để tính toán công việc j trên máy m. Workload cho biết khối lượng công việc của từng công việc. Computing Capacity cho biết công suất làm việc của từng máy. Ready time là thời gian để một máy bắt đầu làm việc.

3.3. Giải thuật Min-Max

Trong Max-min heuristic, xét tập hợp U chứa tất cả các nhiệm vụ chưa được gán cho máy nào. Sau đó tìm ra được tập hợp thời gian hoàn thành lâu nhất của các công việc thuộc U. Tiếp theo, các công việc với thời gian hoàn thành lâu nhất từ M được chọn và gán cho máy tương ứng (do đó có tên Min-Max). Cuối cùng, lấy công việc mới được gán cho máy ra khỏi U, và lặp lại cho đến khi tất cả các công việc được gán (U sẽ rỗng).

3.4. Tìm kiếm cục bộ

GCS thường liên quan đến nhiều công việc và các loại máy khác nhau, nhưng không chắc rằng có thể tìm thấy được giải pháp tốt từ lời giải ngẫu nhiên ban đầu. Vì vậy, tìm kiếm các giải pháp gần tối ưu thông qua FOA sẽ khó khăn. Rất hữu ích nếu áp dụng tìm kiếm cục bộ thật hiệu quả để nhanh chóng tìm ra giải pháp hứa hẹn hơn. Một tìm kiếm cục bộ mới được đề xuất trong bài báo này để giúp tinh chỉnh các giải pháp GCS tìm thấy bằng FOA mà không làm tăng đáng kể thời gian tính toán. Tìm máy có thời gian hoàn thành chậm nhất (max of Completion time)-Machine$_{max}$. Tìm máy có thời gian hoàn thành nhanh nhất (min of Completion time)-Machine$_{min}$. Chuyển processing time nhỏ nhất ở Machine$_{max}$ cho Machine$_{min}$ xử lí. Quá trình này lập lại cho đến khi makespan không còn được cải tiến nữa. Mã giả của heuristic tìm kiếm cục bộ được trình bày trong giải thuật sau đây.

Cho x=(Age,x_1,x_2,…,x_n) với hàm makespan f(x)

Trong khi bài toán chưa được cải thiện

x_{temp}=x

max=f(x_{temp})

l=machine_max(x_{temp})

s=machine_min(x_{temp})

j=job_min(x_{temp},L)

x_j=l

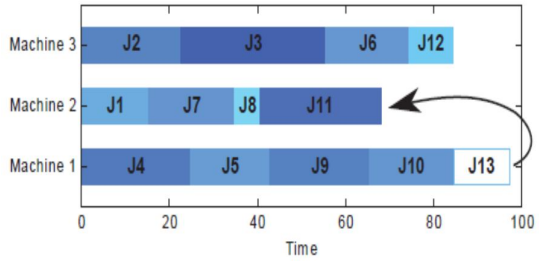
if f(x_{temp})<f(x)

x=x_{temp}

f(x)=f(x_{temp})

Kết thúc

Trong thuật toán này, f(x) là các makespan thu được từ lời giải x. Hàm max_machine(x) (hoặc min_machine (x)) cung cấp cho các chỉ số của máy với thời gian hoàn thành dài nhất (hoặc ngắn nhất) từ lời giải x. Hàm min_job(x,d,c) cho phép công việc hiện tại được gán vào máy c với thời gian xử lí ngắn nhất từ máy d. Với một lời giải x cho trước, giải thuật heuristics này sẽ xác định máy c với thời gian hoàn thành lâu nhất (bằng với makespan). Sau đó, công việc j với thời gian xử lí ngắn nhất trên máy c sẽ được gỡ bỏ và chèn vào máy d với thời gian hoàn thành nhanh nhất để tạo ra các giải pháp mới x'. Một ví dụ về phỏng đoán này được thể hiện trong hình 3. Trong trường hợp này, công việc 13 từ máy 1 được chuyển đến máy 2. Nếu tác vụ này cải thiện makespan tổng thể của bài toán thì thay đổi được chấp nhận (x* ← x'). Và quá trình này được lặp đi lặp lại cho đến khi không cải thiện được thực hiện. Nguyên tắc chính của Heuristic này là để cân bằng khối lượng công việc của máy để giảm các makespan. Bởi vì chỉ có một thay đổi đơn giản được thực hiện trong mỗi lần lặp, giải pháp mới x này có thể được đánh giá lại một cách nhanh chóng (bằng cách trừ đi thời gian xử lí pjc từ Cc và thêm pjd từ Cd và tìm Cmax). Một ví dụ được trình bày trong hình sau. Máy 1 và 2 có makespane dài nhất và ngắn nhất. Ở máy 1, j_{13} có thời gian xử lí là ngắn nhất nên nó được chuyển cho máy 2 xử lí. Nếu việc chuyển đổi này làm cải thiện makespan of lời giải thì thực hiện x*←x’ và tiếp tục quá trình này cho đến không cải thiện được makespan.



Hình 9. *Kết quả tìm kiếm cục bộ*

3.5. Giải thuật tổng thể

Khởi tạo rừng bằng các cây được tạo ngẫu nhiên

Mỗi cây là 1 vec tơ x có (d+1) chiều, x=(age,x_1,x_2,…,x_D)

Gán age=0

Trong khi điều kiện dừng chưa thỏa

Thực hiện gieo mầm cục bộ với cây có Age=0 bằng các số rời rạc phối hợp giải thuật Min-Max

Sử dụng tìm kiếm cục bộ

Thực hiện giới hạn quần thể

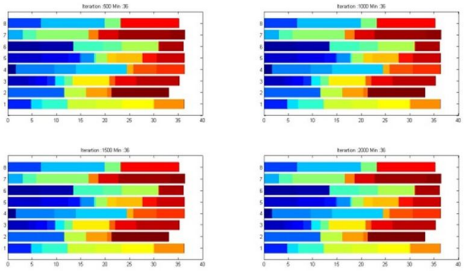
Gieo mầm toàn cục

Cập nhật cây tốt nhất

Trả về cây tốt nhất

4. Thiết kế thực nghiệm và kết quả

Để minh họa, bài báo sử dụng dữ liệu được lấy từ [8]. Thực nghiệm bắt đầu với bài toán có 3 máy và 13 công việc, kí hiệu là (3,13) đã được [8] đề cập. Tốc độ xử lí của 3 máy là 4,3,2 đơn vị thời gian và 13 công việc có thời gian xử lí là 6, 12, 16, 20, 24, 28, 30, 36, 40, 42, 48, 52, 60 đơn vị thời gian. Kết quả thực tế từ giải thuật này với 10 lần chạy là (46, 47, 46, 47, 47, 47, 47, 47, 47, 46). Giá trị nhỏ nhất và giá trị trung bình là 46 và 46.67. Thông số giải thuật cho bài toán với số chiều <5 là "life time"=15, LSC=1, GSC=1, "tranfer rate"=10, "area limit"=10. Còn với số chiều >=5 thì "life time"=15, LSC=20%*số chiều của bài toán, GSC=10%*số chiều của bài toán, "tranfer rate"=10, "area limit"=10). Hình dưới đây cho thấy kết quả của bài toán (3,13).



Hình 10. *Kết quả bài toán (3,13)*

Với bài toán (5,100) , thực hiện 10 lần thử nghiệm kết quả là (100, 100, 100, 101, 100, 100, 101, 100, 100, 100), giá trị nhỏ nhất và giá trị trung bình là 100 và 100.2. Sau đây là bảng so sánh giải thuật đề nghị và giải thuật PSO trên bộ dataset [8] với giá trị tốt nhất mà mỗi bài toán đạt được. "(3,13) 1" có nghĩa là số liệu của 3 máy, 13 công việc của bộ 1, "(3,13) 2" là của bộ thứ 2 …

Bảng 1. *So sánh FOA và PSO*

Dữ liệu	**(3,13) 1**	**(3,13) 2**	**(3,13) 3**	**(5,100) 1**	**(5,100) 2**	**(5,100) 3**	**(8,60) 1**	**(8,60) 2**	**(8,60) 3**
FOA	87.75	77.75	67	97.5	100	100	37.40	36.24	40
PSO	87.75	77.75	67	97.5	100	100	37.40	36.00	40

Dữ liệu	**(10,50) 1**	**(10,50) 2**	**(10,50) 3**	**(10,100) 1**	**(10,100) 2**	**(10,100) 3**	**(60,500) 1**	**(60,500) 2**	**(60,500) 3**
FOA	42.24	39.00	43.72	71.91	68.61	71.84	51.20	52.82	54.61
PSO	42.5	40.00	42.00	70.00	65.00	75.00	52.00	53.00	55.00

5. Kết luận

Bài báo này giới thiệu giải thuật mới, tối ưu hóa rừng cây [5] để giải quyết các bài toán tối ưu liên tục. Sau đó chúng tôi đã chỉnh sửa để có thể áp dụng cho bài toán rời rạc. Ý tưởng chính của bài này dùng giải thuật tối ưu hóa rừng cây với các biến rời rạc kết hợp giải thuật Min-Max và tìm kiếm cục bộ để giải bài toán lập lịch lưới tính toán cho các công việc độc lập. Giải thuật Min-Max dùng để tạo ra lời giải ban đầu có thể phân bổ các công việc vào các máy sao cho makespan hy vọng là sớm đến cực tiểu. Giải thuật tìm kiếm cục bộ nhanh chóng sắp xếp các công việc cho đến khi việc cực tiểu hầu như không được cải thiện nữa. Kết quả thực nghiệm cho thấy giải thuật cho kết quả tốt và nhanh chóng trên các bài toán (3,13), (5,100), (8,60) và (10,50). Với bài toán (3,13) thì kết quả cũng xấp xỉ như [8]. Các kết quả khác chưa có so sánh nhưng thời gian hội tụ là rất nhanh. Kết quả cho thấy cơ chế Min-Max và tìm kiếm cục bộ đóng vai trò quan trọng trong việc đạt đến kết quả nhanh chóng cho bài GCS. Hy vọng rằng FOA cũng như FOA chỉnh sửa sẽ được áp dụng cho các bài toán của lập lịch nói riêng hay các lĩnh vực khác nói chung.

TÀI LIỆU THAM KHẢO

1. Ali S., Siegel H.J., Maheswaran M., Hensgen D., Sedigh S.. Ali S. (2000), "Representing task and machine heterogeneities for heterogeneous computing systems", *Tamkang Journal of Science and Engineering*, 3(3), pp.195-207.
2. Dong F., Akl S.G. (2006), "Technical report no. 2006-504 scheduling algorithms for grid computing: State of the art and open problems", *Queen's University*, Tech. Rep. 2006-504.
3. Foster I.,Kesselman C. (2003), "The Grid 2: Blueprint for a New Computing Infrastructure",*Morgan Kaufmann Publishers* Inc., San Francisco, CA, USA.
4. Foster I., Kesselman C., Tuecke S. (2001), "The anatomy of the grid: Enabling scalable virtual organization",*International Journal of HighPerformance Computing Applications*, 15(3), pp.200–222.
5. Ghaemi M., Feizi-Derakhshi M.R. (2014), "Forest Optimization Algorithm",*Expert Systems with Applications* 41, pp.6676–6687.
6. Graham R. (2003), "A fast, effective local search for scheduling independent jobs in heterogeneous computing environments" , in *PLANSIG'03: Proceedings of the 22nd Workshop of the UK Planning and Scheduling Special Interest Group*, pp.178-183.
7. Izakian H., Ladani B.T., Abraham A.,Snasel V. (2010), "A Discrete Particle Swarm Optimization Approach for Grid Job Scheduling",*International Journal of Innovative Computing, Information and Control*, 6(9).
8. Liu H., AbrahamcA., Hassanien A.E. (2010), "Scheduling jobs on computational grids using a fuzzy particle swarm optimisation algorithm", *Future Generation Computer Systems*, 26, pp.1336–1343.

9. Nguyen S.B., Zhang M. (2014), "A hybrid discrete particle swarm optimisation method for grid computation scheduling", in: *Evolutionary Computation* (CEC), 2014 IEEE Congress on. IEEE, pp.483–490.

10. Page A.J., Naughton T.J. (2005), "Framework for Task Scheduling in Heterogeneous Distributed Computing Using Genetic Algorithms",*Artifical Intelligence Review*, 24(3-4), pp.415–429.

11. Ritchie G., Levine J. (2003), "A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments", in *PLANSIG'04: Workshop of the UK Planning and Scheduling Special Interest Group*.

12. Talbi E.G., Zomaya A.Y. (2007), "Wiley Series on Bioinformatics: Computational Techniques and Engineering", in:*Grid Computing for Bioinformatics and Computational Biology*. John Wiley & Sons, Inc., pp. 393–393.

13. Tracy D.B., Howard J.S., Beck N. (2001), "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems", *Journal of Parallel and Distributed Computing*, 61(6), pp.810–837.

14. XhafaF., AlbaE., DorronsoroB. (2007), "Efficient Batch Job Scheduling in Grids using Cellular Memetic Algorithms", in: *Parallel and Distributed Processing Symposium*, 2007. IPDPS 2007. IEEE International. Presented at the Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, pp.1–8.

15. Xhafa F., Carretero J., Dorronsoro B., Alba E. (2009), "A tabu search algorithm for scheduling independent jobs in computational grids", *Computing and informatics* 28(2), pp.237–250.

(Ngày Tòa soạn nhận được bài: 12-01-2015; ngày phản biện đánh giá:12-5-2015; ngày chấp nhận đăng: 18-5-2015)