Kỷ yếu Hội nghị Quốc gia lần thứ X về Nghiên cứu cơ bản và ứng dụng Công nghệ thông tin (FAIR), Đà Nẵng, ngày 17-18/08/2017
DOI: 10.15625/vap.2017.00018

ĐỀ XUẤT THUẬT TOÁN LOCAL SEARCH GIẢI BÀI TOÁN MAX CLIQUE

Huỳnh Thanh Tân[1], Nguyễn Văn Thành[1]

[1]Khoa Công nghệ thông tin, Trường Đại học Sài Gòn

tanhtsgu@gmail.com, thanhnvsgu@gmail.com

TÓM TẮT: *Clique lớn nhất (Max Clique Problem - MCP) là một bài toán tối ưu tổ hợp có nhiều ứng dụng quan trọng trong các lĩnh vực mạng xã hội, thị giác máy tính, nhận biết khuôn mẫu,... MCP là bài toán thuộc lớp NP-Hard và được nghiên cứu rộng rãi trong những năm gần đây. Trong bài báo này, chúng tôi đề xuất một thuật toán Local Search để giải bài toán MCP; thuật toán này được cải tiến từ thuật toán k – opt Local Search (KLS) của nhóm tác giả Kengo Katayama, Akihiro Hamamoto, Hiroyuki Narihisa. Chúng tôi đã so sánh chất lượng của thuật toán cải tiến này với thuật toán k – opt Local Search gốc trên 34 bộ dữ liệu trong hệ thống dữ liệu thực nghiệm chuẩn DIMACS.*

Từ khóa: *Graph Algorithms, Maximum Clique Problem, KLS Local search, Variable Depth Search, Combinatorial Optimization.*

I. GIỚI THIỆU

A. *Một số định nghĩa*

Cho $G=(V,E)$ là đồ thị vô hướng bất kỳ trong đó V là tập của n đỉnh và $E \subseteq V \times V$ là tập các cạnh của G. Tìm tập đỉnh C có kích thước lớn nhất $\subseteq V$ sao cho xây dựng được đồ thị $Gc(Vc,Ec)$ với $Vc = C$ và $Ec = C \times C$, đây được gọi là bài toán *MCP* (Max Clique Problem). Mục này bài báo sẽ trình bày một số định nghĩa và tính chất liên quan.

Định nghĩa 1. *Clique* [7]

Cho $G = (V,E)$ là đồ thị vô hướng bất kỳ trong đó V là tập của n đỉnh và $E \subseteq V \times V$ là tập các cạnh của G. Trong tập con $S \subseteq V$, cho $G(S) = (S, E \cap S \times S)$ là một đồ thị con nằm trong G. Đồ thị $G = (V,E)$ được gọi là đầy đủ nếu tất cả các cặp đỉnh của nó có cạnh nối với nhau, tức là $\forall\, i,j \neq V$ với $i \neq j$, $(i, j) \in E$. Một *clique* C là một tập con của tập đỉnh V sao cho đồ thị $G(C)$ là đồ thị đầy đủ.

Định nghĩa 2. Kích thước *Clique* [9]

Kích thước của một *Clique* C là số đỉnh có trong clique đó, kí hiệu là $|C|$.

Định nghĩa 3. Chỉ số *Clique* [4]

Cho đồ thị vô hướng $G=(V,E)$, chỉ số Clique là số đỉnh có trong Clique lớn nhất thuộc đồ thị G, kí hiệu $\omega(G)$.

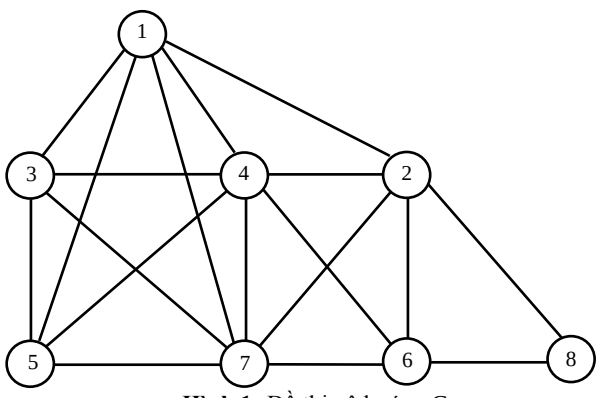
Định nghĩa 4. *Clique* cực đại [4]

Clique cực đại (Maximal Clique) của đồ thị G là Clique mà không bị chứa bởi Clique lớn hơn nó.

Định nghĩa 5. *Clique* lớn nhất [4]

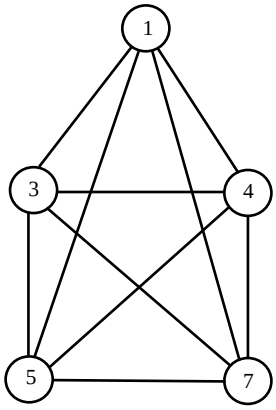
Clique lớn nhất (Maximum Clique) của G là Clique có số đỉnh lớn nhất của G. Một Clique lớn nhất đồng thời là một Clique cực đại, nhưng điều ngược lại chưa chắc đã đúng. Cho đồ thị G được mô tả như trên, bài toán tìm một Clique lớn nhất là bài toán Maximum Clique Problem - *MCP*. *MCP* là bài toán tối ưu tổ hợp trong trường hợp tổng quát, bài toán *MCP* đã được chứng minh là bài toán *NP-hard*. Trong bài báo này, từ đồ thị sẽ được hiểu là đơn đồ thị vô hướng liên thông.

Ví dụ: Cho đồ thị G có 8 đỉnh 18 cạnh như hình vẽ 1



Hình 1. Đồ thị vô hướng G

Khi đó Clique lớn nhất tìm được trên đồ thị G là tập C={1, 3, 4, 5, 7} với $\omega(G)$ = 5 như được minh họa ở hình vẽ 2.



Hình 2. Đồ thị chứa tập đỉnh Clique lớn nhất

Định lý. *Định lý kích thước Clique với một đỉnh bất kỳ trong đồ thị*

Cho đồ thị $G(V, E)$, $o\ v \in V$ thì $|C|$ chứa đỉnh v không vượt quá $deg(v)$ + 1.

B. Ứng dụng

Bài toán *MCP* có những ứng dụng quan trọng trong lĩnh vực khoa học, kĩ thuật; chẳng hạn như: Mạng xã hội [12] , tạo lịch trình thực hiện các tiến trình trên máy tính (Scheduling) [2], đánh nhãn bản đồ (Map Labeling) [13], thị giác máy tính (Computer Vision) [5],…

C. Một số nghiên cứu liên quan bài toán MCP và vấn đề đặt ra cần giải quyết

Do có tính khoa học và tính ứng dụng rộng rãi, bài toán *MCP* đã thu hút được sự quan tâm nghiên cứu liên tục, sâu rộng của nhiều nhà khoa học trên thế giới trong hàng chục năm qua. Hiện nay đã có hàng loạt thuật toán giải bài toán *MCP* được đề xuất và có thể chia chúng làm ba hướng tiếp cận sau:

Hướng thứ nhất là các giải thuật tìm lời giải đúng, chẳng hạn như thuật toán cổ điển *Bron-Kerbosch* [8] của Bron-Kerbosch hay thuật toán *Branch & Bound* [5],... Ưu điểm của hướng tiếp cận này là tìm được lời giải chính xác, nhược điểm của hướng tiếp cận là chỉ tìm được các bài toán có kích thước đồ thị nhỏ. Hướng tiếp cận này là một cơ sở quan trọng để đánh giá mức độ chính xác của các thuật toán giải gần đúng. Việc giải đúng bài toán MCP thực sự là một thách thức lớn trong lý thuyết đồ thị tối ưu tổ hợp.

Hướng thứ hai là các thuật toán heuristic. Thuật toán heuristic chỉ những kinh nghiệm riêng biệt để tìm kiếm lời giải cho một bài toán tối ưu cụ thể. Thuật toán heuristic thường tìm được lời giải có thể chấp nhận được trong thời gian cho phép nhưng không chắc đó là lời giải tốt nhất; thậm chí các thuật toán heuristic không chắc hiệu quả trên mọi loại dữ liệu đối với một bài toán cụ thể. Ưu điểm của các thuật toán heuristic là cho thời gian chạy nhanh. Các thuật toán dùng để tìm các Clique lớn nhất như thuật toán *CLS* và *DLS* của Pullan [14], thuật toán *PLS* [15],… có thể xem là các heuristic cho bài toán *MCP*; tuy nhiên chi phí tìm được các heuristic này không hiệu quả bằng các thuật toán metaheuristic trong việc giải bài toán *MCP*.

Hướng thứ ba là các thuật toán metaheuristic. Thuật toán metaheuristic sử dụng nhiều heuristic kết hợp với các kỹ thuật phụ trợ nhằm khai phá không gian tìm kiếm; metaheuristic thuộc lớp các thuật toán tìm kiếm tối ưu. Hiện đã có nhiều công trình sử dụng thuật toán metaheuristic giải bài toán *MCP*; chẳng hạn như thuật toán tìm kiếm tabu [11], thuật toán luyện kim [16],… Cho đến hiện tại, hướng tiếp cận metaheuristic cho kết quả tốt nhất trong số các thuật toán gần đúng.

Bài toán *MCP* có hệ thống dữ liệu thực nghiệm chuẩn và có nhiều công bố liên quan đã tiến hành thực nghiệm trên hệ thống dữ liệu này. Các đồ thị trong hệ thống dữ liệu này có tối đa 3321 đỉnh, 5506380 cạnh. Một số công trình đã công bố kết quả thực nghiệm trên các bộ dữ liệu chuẩn này.

II. THUẬT TOÁN LOCAL SEARCH CHO BÀI TOÁN MCP

Trong phần này, chúng tôi trình bày về thuật toán *KLS gốc* và đề xuất những cải tiến cho thuật toán *KLS gốc* để thu được kết quả tối ưu hơn cho bài toán *MCP*.

A. Thuật toán KLS

Input: Đồ thị $G(V,E)$

Output: $\omega(G)$ và tập đỉnh clique

Để thuận tiện cho việc mô tả ý tưởng thuật toán *KLS gốc* và *KLS cải tiến*, chúng tôi có các định nghĩa sau:

CChientai: tập clique hiện tại

CClancan: tập clique tìm kiếm lân cận

CCbestlancan: tập clique lớn nhất tìm được trong tìm kiến lân cận

PA: Tập đỉnh có khả năng thêm vào *CChientai*

Ý tưởng của thuật toán *KLS gốc* gồm các chuỗi thao tác thêm và loại một đỉnh từ clique hiện tại để tìm một clique lớn hơn trong không gian khả thi lân cận, chi tiết thuật toán của tác giả được trình bày chi tiết trong tài liệu [7]. Thuật toán *KLS* gốc gồm 2 giai đoạn:

1. *Giai đoạn khởi tạo (1-opt)*: khởi tạo một clique bằng thao tác thêm từng đỉnh ngẫu nhiên có bậc lớn nhất trong *PA* vào *CChientai*, sau mỗi thao tác thêm đỉnh cập nhập tập đỉnh *PA* và bậc đỉnh G_{PA}, lặp lại thao tác thêm đến khi $PA=\emptyset$.

2. *Giai đoạn tìm kiếm lân cận (k-opt)*: Cập nhật *CClancan=CChientai*, *CCbestlancan=CChientai*. Sau đó, thực hiện chuỗi các thao tác thêm và loại đỉnh liên tiếp, khi $PA \neq \emptyset$, thao tác thêm tìm đỉnh ngẫu nhiên có bậc lớn nhất từ *PA* thêm vào *CClancan* và ngược lại thực hiện thao tác loại đỉnh ngẫu nhiên từ *CClancan* sao cho |*PA*| lớn nhất, các đỉnh thêm và loại được đánh dấu nhằm tránh việc trùng lặp đỉnh trong thao tác thêm loại, sau mỗi thao tác cập nhập tập đỉnh *PA* và bậc đỉnh G_{PA}. Chuỗi thao tác thêm và loại đỉnh dừng khi đã loại hết đỉnh của *CClancan* tồn tại trong *CChientai*, trong thao tác thêm thực hiện so sánh giữa *CClancan* và *CCbestlancan*, nếu |*CClancan*| >|*CCbestlancan*| thì tiến hành cập nhật *CCbestlancan=CClancan* để tìm lân cận tốt nhất. Kết thúc chuỗi thao tác thêm loại nếu |*CCbestlancan*|>|*CChientai*| thì cập nhật *CChientai=CCbestlancan* và tiếp tục tìm kiếm lân cận từ *CChientai*. Thuật toán *k-opt* dừng khi kết thúc quá trình thực hiện các chuỗi thao tác thêm và loại đỉnh, *CCbestlancan* không tìm được clique có kích thước lớn hơn CC*hientai*.

B. Đề xuất cải tiến cho thuật toán KLS

Trong mục này, chúng tôi đề xuất những cải thiện trên thuật toán *KLS* gốc để có những kết quả tối ưu hơn.

Trong thuật toán *KLS* gốc được trình bày ở phần trên, chúng tôi nhận thấy rằng trong thao tác thêm đỉnh có bậc lớn nhất trong tập *PA* là lựa chọn tối ưu, nhưng không gian tìm kiếm lân cận lại bị hạn chế rất nhiều trong thao tác thêm một đỉnh và không thu được kết quả tối ưu hơn. Đồng thời, hạn chế của thuật toán *KLS* gốc trong giai đoạn *k-opt* ở điều kiện dừng là nếu không tìm thấy clique lớn hơn trong không gian khả thi lân cận thì kết thúc, tuy nhiên với điều kiện dừng này không khai thác hiệu quả không gian khai thác. Với những hạn chế được chúng tôi rút ra và đề cập ở trên, tiếp theo chúng tôi sẽ trình bày những cải tiến cụ thể trong các đoạn mã giả sau.

Gọi |*CCgoc*| là kích thước clique thu được qua thuật toán *KLS* gốc, vấn đề này sẽ giúp khởi tạo kích thước clique tối ưu bằng giải thuật *KLS* gốc và sau đó cải tiến không gian lân cận để tìm clique tối ưu hơn.

1. Mã giả cải tiến

Giai đoạn 1. *Khởi tạo một clique ngẫu nhiên (1-opt cải tiến)*

one_opt_advanced (V, E)

1. *CChientai* = ∅; // Khởi tạo tập clique ban đầu là rỗng
2. **while** ($PA \neq \emptyset$){
3. **if** ($\exists\ v \in PA$: *deg(v)* >= |*CCgoc*|) Chọn ngẫu nhiên một đỉnh *v* trong *PA* có *deg(v)* >= |*CCgoc*|;
4. **else** Chọn ngẫu nhiên một đỉnh *v* trong *PA* có bậc lớn nhất;
5. *CChientai=CChientai* ∪ *v*;
6. Cập nhật *PA* và bậc từng đỉnh trong G_{PA};
7. }
8. **return** *CChientai*;

Giai đoạn 2. *Tìm kiếm lân cận tốt hơn (k-opt cải tiến)*

kopt_advanced (V, E, CChientai)

1. **do**{
2. *CClancan=CChientai*;
3. *CCbestlancan=CChientai;* // lân cận tốt nhất

4. *count_vertex_delete=0;* // đếm đỉnh loại trong *CClancan* có tồn tại trong *CChientai*
5. *find_neightbor=false;* //biến kiểm tra có tìm thấy clique lân cận tốt hơn clique hiện tại
6. Cho mảng một chiều *color[v]*, đặt *color[v]=0,* $\forall v \in V$
7. //tất cả đỉnh ban đầu chưa được đánh dấu, mảng color để tránh các thao tác thêm loại bị lặp lại
8. **do**{
9. **if** (PA ≠ ∅) {
10. **if** ($\exists$ *v* ϵ *PA: deg(v)* >= |*CCgoc*|)
11. Chọn ngẫu nhiên một đỉnh *v* trong *PA* có *deg(v)* >=|*CCgoc*|;
12. **else** Chọn ngẫu nhiên một đỉnh *v* trong *PA* có bậc lớn nhất;
13. *color[v]=1;* // đánh dấu đỉnh *v*
14. *CClancan= CClancan* $\cup${*v*};
15. **if** (|*CClancan*| > |*CCbestlancan*|) {
16. *CCbestlancan=CClancan*;
17. *find_neightbor*=true; // tìm thấy lân cận tốt hơn hiện tại
18. *CCSAME*=∅; //cập nhật để chứa các lân cận khác tốt hơn
19. }
20. **else if** (|*CClancan*| = |*CCbestlancan*| *và CClancan* $\notin$ *CCSAME*)
21. *CCSAME=CCSAME* $\cup$ *CClancan*; // lưu lân cận tốt nhất tìm được
22. }
23. **else**{
24. Tìm 1 đỉnh *v* ϵ *CClancan* chưa được đánh dấu sao cho |*PA*| là lớn nhất;
25. *color[v]=1;* // đánh dấu đỉnh v
26. *CClancan=CClancan* \ {*v*};
27. **if** (*v* $\in$ *CChientai*) *count_vertex_delete*++;
28. }
29. Cập nhật tập *PA* và bậc từng đỉnh trong G_{PA}; // các đỉnh trong *PA* không bị lặp lại, tức $\forall$ *i* ϵ *PA, c[i]=0*
30. }
31. **while** (*demloai* < |*CChientai*|); // điều kiện dừng khi loại hết tất cả đỉnh *CClancan* có trong *CChientai*
32. **if** (*find_neightbor*=true) *CChientai=CCbestlancan*; // cập nhật lân cận tốt nhất.
33. }
34. **while** (*find_neightbor*=true);
35. **return** *CChientai*; // kết quả tốt nhất tìm được trong lân cận

Giai đoạn 3. *Khai thác hiệu quả lân cận tốt nhất*

KLS(V, G, iter)

1. *CCSAME=*∅; *//* lưu các clique lân cận có kích thước giống với *CChientai*
2. one_opt_advanced(*V*,*E*); // khởi tạo clique ngẫu nhiên
3. *CCbest=CChientai;* // clique tốt nhất được tìm thấy
4. *count=0;* // đếm số lần không tìm được lân cận tốt hơn
5. **do{**
6. **while** (*count*<*iter*) {
7. kopt_advanced(*V, E, CChientai*)*;*
8. **if** (|*CChientai*| > |*CCbest*|) {

9. *count=0*;
10. *CCbest=CChientai*;
11. *CCgoc=CChientai*; // cập nhật lại giá trị clique tốt nhất cho *CCgoc*.
12. }
13. **else** *count++;*
14. }
15. **if** (*CCSAME* ≠ ∅) {
16. *CChientai*=Một clique *CC* ϵ *CCSAME*;
17. *CCSAME=CCSAME* \ *CC*;
18. }
19. } **while** (*CCSAME* ≠ ∅); // tiếp tục tìm kiếm xung quanh lân cận cho đến khi khai thác hết.
20. **return** *CCbest*;

2. Nội dung cải tiến

Chúng tôi trình bày 2 cải tiến cụ thể thu được kết quả tốt hơn *KLS gốc* như sau:

Cải tiến 1: Mở rộng không gian thêm đỉnh

Trong *KLS gốc*, thao tác thêm thực hiện chọn đỉnh ngẫu nhiên có bậc lớn nhất trong *PA* và thêm vào clique có khuyết điểm sau: Hạn chế lớn về tính đa dạng của giai đoạn *khởi tạo clique* từ đó không tìm kiếm hiệu quả từ không gian lân cận; cụ thể trong một số trường hợp, các clique được khởi tạo có sự giống nhau chứa các đỉnh bậc lớn nhất, như vậy không gian *khởi tạo clique* chỉ xung quanh những đỉnh bậc lớn và không tiếp cận được lời giải tốt nhất cần tìm, đồng thời tìm kiếm lân cận xung quanh không hiệu quả, hạn chế nói trên được chúng tôi thực nghiệm *20* lần trên các bộ *test brock* bằng cách kiểm tra giai đoạn khởi tạo. Hạn chế tính đa dạng này bao gồm cả chuỗi thao tác thêm trong tìm kiếm lân cận ở giai đoạn 2.

Qua hạn chế được rút ra, chúng tôi đề ra phương pháp: trong *PA*, chọn đỉnh ngẫu nhiên có bậc không vượt quá clique tốt nhất tìm được hiện tại, nếu không tồn tại đỉnh như vậy thì chọn đỉnh ngẫu nhiên có bậc lớn nhất, giải pháp này có 2 ưu điểm:

- Tăng tính đa dạng trong *khởi tạo clique* và giai đoạn thêm đỉnh, khắc phục hạn chế của *KLS gốc*.
- Đảm bảo không gian thêm vào không quá lớn dẫn đến thuật toán không khả thi và các đỉnh thêm vào clique từ *PA* có *khả năng* tìm được clique lớn hơn hiện tại tối thiểu là một đỉnh.

Cải tiến 1 này được trình bày chi tiết trong giai đoạn 1: dòng 3,4; giai đoạn 2: dòng 9,10.

Cải tiến 2: Khai thác hiệu quả các lân cận

Dựa trên điều kiện dừng của thuật toán *KLS gốc* là: không tìm được một clique có kích thước lớn hơn từ không gian tìm kiếm khả thi lân cận của clique hiện tại thì dừng lại, với điều kiện dừng này có nhiều hạn chế, cụ thể như:

- Chưa khai thác hiệu quả clique hiện tại bằng thao tác tìm kiếm lân cận.
- Không khai thác các clique tốt tìm được trong quá trình tìm kiếm lân cận.

Do vậy, chúng tôi thực hiện cải tiến: khai thác không gian khả thi lân cận của clique hiện tại trong *iter* lần, lưu trữ các clique lân cận tốt nhất vào mảng *CCSAME*, sau *iter* lần nếu không tìm thấy clique tốt hơn trong lân cận thì tiến hành khai thác không gian của các clique trong mảng *CCSAME* như trong clique hiện tại đến khi *CCSAME* = ∅ thì kết thúc. Cải tiến này làm tăng xác suất khả thi để tìm kiếm một clique tốt hơn và cho kết quả tối ưu hơn bằng khai thác một cách hiệu quả các lân cận tốt nhất tìm được xung quanh giai đoạn tìm kiếm lân cận, tuy nhiên sẽ làm tăng thời gian chạy nhiều. Trong mã giả *k-opt cải tiến* khi thực hiện cải tiến thay đổi giai đoạn 2: dòng 13 đến 19 để lưu trữ và cập nhật các clique tốt nhất trong lân cận để khai thác; giai đoạn 3 để thực hiện chọn khai thác không gian clique lân cận tốt nhất tìm được xung quanh clique hiện tại.

Tóm lại, với những cải tiến được đề xuất cho thuật toán *KLS* là *mở rộng không gian lựa chọn đỉnh thêm vào trong thao tác thêm* và *khai thác hiệu quả lận cận tốt nhất tìm được* đã thu được một số kết quả tối ưu và vượt trội so với *KLS* gốc trong giải quyết bài toán *MCP*, chúng tôi thực nghiệm và so sánh kết quả với 34 bộ test *DIMACS* giữa *KLS* gốc và *KLS* cải tiến.

III. THỰC NGHIỆM VÀ ĐÁNH GIÁ

A. Dữ liệu thực nghiệm

Chúng tôi sử dụng tổng cộng 34 bộ dữ liệu đồ thị thưa trong hệ thống dữ liệu thực nghiệm chuẩn *DIMACS* cho bài toán *MCP*: URL *http://iridia.ulb.ac.be/~fmascia/maximum_clique/* [3]. Thông tin chi tiết về 34 bộ dữ liệu này được trình bày ở Bảng 1; trong đó các cột lần lượt ghi các thông tin về tên tập tin trong hệ thống dữ liệu thực nghiệm chuẩn, số đỉnh và số cạnh.

Bảng 1a. Thông tin dữ liệu thực nghiệm nhóm 1

Test	N	M
C125.9	125	6 963
C250.9	250	27 984
C500.9	500	112 332
C1000.9	1 000	450 079
C2000.5	2 000	999 836
C4000.5	4 000	4 000 268
DSJC1000_5	1 000	499 652
DSJC500_5	500	125 248
MANN_a27	378	70 551
MANN_a45	1 035	533 115

Bảng 1b. Thông tin dữ liệu thực nghiệm nhóm 2

brock200_2	200	9 876
brock200_4	200	13 089
brock400_2	400	59 786
brock400_4	400	59 765
brock800_2	800	208 166
brock800_4	800	207 643
gen200_p0.9_44	200	17 910
gen200_p0.9_55	200	17 910
gen400_p0.9_55	400	71 820
gen400_p0.9_65	400	71 820
gen400_p0.9_75	400	71 820
hamming8-4	256	20 864
hamming10-4	1 024	434 176

Bảng 1c. Thông tin dữ liệu thực nghiệm nhóm 3

keller4	171	9 435
keller5	776	225 990
p_hat300-1	300	10 933
p_hat300-2	300	21 928
p_hat300-3	300	33 390
p_hat700-1	700	60 999
p_hat700-2	700	121 728
p_hat700-3	700	183 010
p_hat1500-1	1 500	284 923
p_hat1500-2	1 500	568 960
p_hat1500-3	1 500	847 244

B. Môi trường thực nghiệm

Các thuật toán được cài đặt bằng ngôn ngữ C++ sử dụng môi trường C-Free 5.0, CPU Intel(R) Core(R) i5-5200U 2.20GHz, RAM 4GB, hệ điều hành Windows 10, 64 bit.

C. Kết quả thực nghiệm và đánh giá

1. Kết quả thực nghiệm

Trong kết quả thực nghiệm này, chúng tôi chọn ***iter***=100 và tiến hành chạy ngẫu nhiên 300 lần. Thời gian trung bình được tính là thời gian của 300 lần chạy trung bình (Đơn vị giây: s).

Bảng 2a. *Kết quả thực nghiệm nhóm 1*

Test	KLS gốc	KLS cải tiến		
	|CC|	|CC|	|CC| trung bình	Thời gian trung bình
C125.9	34	34	34.000	5.12
C250.9	44	44	44.000	12.29
C500.9	57	57	56.627	80.08
C1000.9	68	68	66.550	423.11
C2000.5	16	16	15.430	1646.34
C4000.5	18	18	16.352	3804.35
DSJC1000_5	15	15	15.000	10.35
DSJC500_5	13	13	13.000	6.35
MANN_a27	126	126	125.923	14.55
MANN_a45	345	345	343.312	40.35

Bảng 2b. Kết quả thực nghiệm nhóm 2

brock200_2*	11	12	10.840	2.04
brock200_4*	16	17	16.530	3.26
brock400_2*	25	29	23.347	5.03
brock400_4*	25	33	23.583	12.51
brock800_2*	21	24	21.356	20.53
brock800_4	21	21	20.465	22.56
gen200_p0.9_44	44	44	43.063	10.82
gen200_p0.9_55	55	55	55.000	12.60
gen400_p0.9_55*	53	55	55.000	24.53
gen400_p0.9_65	65	65	65.000	32.22
gen400_p0.9_75	75	75	74.227	56.43
hamming10-4	40	40	39.391	152.34
hamming8-4	16	16	15.352	18.23

Bảng 2c. Kết quả thực nghiệm nhóm 3

keller4	11	11	11.000	15.08
keller5	27	27	26.747	28.34
p_hat300-1	8	8	8.000	3.46
p_hat300-2	25	25	25.000	6.29
p_hat300-3	36	36	36.000	9.43
p_hat700-1	11	11	11.000	15.36
p_hat700-2	44	44	44.000	32.10
p_hat700-3	62	62	62.000	39.01
p_hat1500-1	12	12	12.000	123.34
p_hat1500-2	65	65	64.253	233.40
p_hat1500-3	94	94	93.533	845.12

2. Đánh giá và nhận xét

Kết quả thực nghiệm của thuật toán được ghi nhận trên 34 bộ dữ liệu *DIMACS* chi tiết trong *Bảng 2*, thông tin về mỗi kết quả của từng dữ liệu lần lượt ứng với tên của bộ dữ liệu, clique tốt nhất của thuật toán *KLS gốc* ghi nhận theo nguồn tài liệu của tác giả trong *tài liệu tham khảo* [7] và thông tin của *KLS cải tiến* gồm: clique tốt nhất tìm được (được đánh dấu * ghi chú những bộ dữ liệu kết quả tốt hơn so với *KLS gốc*), mật độ clique trung bình, thời gian chạy trung bình của thuật toán.

Dựa vào kết quả ở *Bảng 2*, chúng tôi có một số nhận xét sau:

Thuật toán *KLS cải tiến* thu được 6 bộ dữ liệu có kết quả tốt hơn so với *KLS gốc*; đặc biệt, thuật toán mang tính hiệu quả cao trên những bộ dữ liệu *brock*, các bộ dữ liệu được bài báo nghiên cứu tổng quan [8] đánh giá có kết quả không tốt khi thực hiện khảo sát bằng thuật toán *KLS gốc*, mật độ trung bình của thuật toán *KLS cải tiến* cao, tuy nhiên các bộ *test brock* có mật độ thấp chứng tỏ tần suất xuất hiện kết quả tối ưu là không nhiều, thời gian chạy của *KLS cải tiến* gấp nhiều lần so với *KLS* gốc vì khai thác hiệu quả lần lượt các lân cận để đạt đến kết quả tốt nhất. Tóm lại, thuật toán *KLS gốc* được chúng tôi cải tiến thu được kết quả tốt hơn, những cải tiến được đề xuất dựa trên những hạn chế của *KLS gốc* và từ đó đưa ra giải pháp tốt hơn.

IV. KẾT LUẬN

Bài báo này đề xuất những cải tiến cho thuật toán *k – opt Local Search (KLS*) của nhóm tác giả Kengo Katayama, Akihiro Hamamoto, Hiroyuki Narihisa và được thực nghiệm so sánh qua 34 bộ dữ liệu *DIMACS*. Những cải tiến được đề xuất thu được 6 kết quả tốt hơn so với thuật toán *KLS gốc*, phần lớn ở các bộ test *brock*, kết quả đạt được trên dữ liệu thực nghiệm *brock* là tối ưu hiện nay [3], góp phần thay đổi những đánh giá của các nhà nghiên cứu bài toán *MCP* về thuật toán *KLS* không hiệu quả trên những bộ dữ liệu *brock* [8]. Thuật toán *KLS cải tiến* và kết quả thực nghiệm là thông tin hữu ích cho việc tiếp cận giải bài toán *MCP*, đồng thời đưa ra những hướng nghiên cứu mới cho các thuật toán Local Search. Thực tế hiện nay, có nhiều thuật toán gần đúng giải bài toán *MCP*, mỗi thuật toán đều có ưu và khuyết điểm, riêng với thuật toán *KLS – một metaheuristic*, trong bài báo này đã đề cập chi tiết một số hạn chế của thuật toán *KLS* và đưa ra những phương hướng giải quyết cho từng trường hợp, những cải tiến được đề ra là tốt hơn về kết quả cho thuật toán *KLS* bằng thực nghiệm so sánh, qua đó đóng góp một thuật toán giải hiệu quả bài toán *MCP* dựa trên thuật toán *k – opt Local Search (KLS).*

TÀI LIỆU THAO KHẢO

[1] Đàm Thanh Chương, Ngô Mạnh Tưởng, Khoa Thu Hoài, "Bài toán clique lớn nhất - ứng dụng và những thách thức tính toán", Tạp chí Khoa học và Công nghệ, 26-3-2013.

[2] David Helmbold and Ernst Mayr, "Applications of Parallel Scheduling to perfect graphs", Stanford University, pp.1-14, June 1986.

[3] http://iridia.ulb.ac.be/~fmascia/maximum_clique/

[4] https://vi.wikipedia.org/wiki/Clique.

[5] J. Jeffry Howbert và Jacki Roberts, "The Maximum Clique Problem", CSEP 521 Applied Algorithms, 2007.

[6] Janez Konc and Duˇsanka Janeˇziˇc, "An improved branch and bound algorithm for the maximum clique problem", Received June 21, 2007.

[7] Kengo Katayama, Akahiro Hamamoto và Hiroyuki Narakisa, "An Effective Local Search for the Maximum Clique Problem" , Information Processing Letters 95 (2005) 503–511, 2005.

[8] Krisna Kumar Signh, Dr. Ajeet Kumar Pandey, "Survey of Algorithms on Maximum Clique Problems", International Advanced Research Journal in Science, Engineering and Technology, Vol. 2, Issue 2, pp.18, 19, 2015.

[9] Patrick Prosser, "Exact Algorithms for Maximum Clique", A Computational Study TR-2012-333, 2012.

[10] Qinghua Wu, Jin-Kao Hao, "A review on algorithms for maximum clique problems", October 15, 2014.

[11] Qinghua Wu, Jin-Kao Hao, "An Adaptive Multistart Tabu Search Approach to solve the Maximum Clique Problem", 17 November 2011.

[12] Spyridon Pollatos, "Solving the maximum clique problem on a class of network graphs, with application to social networks", Naval postgraduate school, pp.5-43, June 2008.

[13] Tycho Strijk, Bram Verweij and Karen Aardal, "Algorithms for Maximum Independent Set Applied to Map Labelling", pp.1-38, September 27, 2000.

[14] Wayne Pullan, Franco Mascia, Mauro Brunato, "Cooperating local search for the maximum clique problem", Received: 9 March 2009.

[15] Wayne Pullan, W.J Comb Optim, "Phased local search for the maximum clique problem", 2006.

[16] Xiutang Geng, Jin Xu, Jianhua Xiao, Linqiang Pan, "A simple simulated annealing algorithm for the maximum clique problem", 15 November 2007.

A IMPROVED LOCAL SEARCH FOR THE MAXIMUM CLIQUE PROBLEM

Huynh Thanh Tan, Nguyen Van Thanh

ABSTRACT: *The Maximum Clique Problem (MCP) is a combinatorial optimization problem which has important practical applications like social networking, computer vision, pattern recognition,… The MCP is known to be NP-hard and recently has received much attention from the scientific community. In this paper we propose an improved version of the k – opt Local Search (KLS), which is a variable depth search based algorithm suggested by Kengo Katayama, Akihiro Hamamoto and Hiroyuki Narihisa for the maximum clique problem. Extensive testing on 34 DIMACS benchmark graphs shows good results in comparison to the original KLS.*