ĐỀ XUẤT CHIẾN LƯỢC TÌM KIẾM LÂN CẬN CHO BÀI TOÁN CÂY STEINER NHỎ NHẤT

Trần Việt Chương[*], **Phan Tấn Quốc**[÷], **Hà Hải Nam**[+]

[*] Trung tâm Công nghệ thông tin và Truyền thông, Sở Thông tin và Truyền thông tỉnh Cà Mau
[÷] Khoa Công nghệ thông tin, Trường Đại học Sài Gòn
[+] Viện Công nghiệp phần mềm và nội dung số Việt Nam, Bộ Thông tin và Truyền thông

Tóm tắt - Cây Steiner nhỏ nhất là bài toán tối ưu đồ thị có nhiều ứng dụng quan trọng trong khoa học và kỹ thuật. Đây là bài toán thuộc lớp NP-hard. Hiện đã có nhiều hướng tiếp cận giải bài toán cây Steiner nhỏ nhất như các thuật toán tìm lời giải đúng, các thuật toán tìm lời giải gần đúng cận tỉ lệ, các thuật toán heuristic, các thuật toán metaheuristic…; trong đó, các chiến lược tìm kiếm lân cận có vai trò quyết định đối với việc cải thiện chất lượng lời giải của các thuật toán metaheuristic. Trong bài báo này, chúng tôi đề xuất hai chiến lược tìm kiếm lân cận và chúng tôi sử dụng các chiến lược tìm kiếm lân cận này trong ngữ cảnh của thuật toán tìm kiếm lân cận biến đổi để giải bài toán cây Steiner nhỏ nhất. Kết quả thực nghiệm với hệ thống dữ liệu thực nghiệm chuẩn cho thấy rằng thuật toán đề xuất của chúng tôi cho lời giải với chất lượng tốt hơn so với một số thuật toán heuristic và một số thuật toán metaheuristic hiện biết trên một số bộ dữ liệu.

Từ khóa - Steiner minimal tree, neighbor search, variable neighborhood search algorithm, metaheuristic algorithm, sparse graph.

I. GIỚI THIỆU

A. Một số định nghĩa

Mục này trình bày một số định nghĩa và tính chất căn bản liên quan đến bài toán cây Steiner nhỏ nhất.

Định nghĩa 1. *Cây Steiner* [3]

Cho $G = (V(G), E(G))$ là một đơn đồ thị vô hướng liên thông và có trọng số không âm trên cạnh, $V(G)$ là tập gồm n đỉnh, $E(G)$ là tập gồm m cạnh, $w(e)$ là trọng số của cạnh e với $e \in E(G)$. Cho $L \subseteq V(G)$, cây T đi qua tất cả các đỉnh trong tập L, tức với $L \subseteq V(T)$ được gọi là cây Steiner của L. Tập L được gọi là tập *terminal*, các đỉnh thuộc tập L được gọi là đỉnh *terminal*. Đỉnh thuộc cây T mà không thuộc tập L được gọi là đỉnh *Steiner* của cây T đối với tập L.

Định nghĩa 2**.** *Chi phí cây Steiner* [3]

Cho $T = (V(T), E(T))$ là một cây Steiner của đồ thị G. Chi phí của cây T, ký hiệu là $C(T)$, là tổng trọng số của các cạnh thuộc cây T, tức $C(T) = \sum_{e \in E(T)} w(e)$.

Định nghĩa 3. *Cây Steiner nhỏ nhất* [3]

Cho đồ thị G được mô tả như trên, bài toán tìm cây Steiner có chi phí nhỏ nhất được gọi là bài toán cây Steiner nhỏ nhất (Steiner Minimal Trees Problem - *SMT*) hoặc được gọi ngắn gọn là bài toán cây Steiner (Steiner Trees Problem).

SMT là bài toán tối ưu đồ thị. Trong trường hợp tổng quát, *SMT* đã được chứng minh thuộc lớp *NP-hard* [3,20,36].

Khác với bài toán cây khung nhỏ nhất (Minimum Spanning Trees Problem - *MST*); cây Steiner là cây đi qua tất cả các đỉnh thuộc tập *terminal L* và có thể có thêm một số đỉnh khác nữa thuộc tập $V(G)$ chứ không nhất thiết phải đi qua tất cả các đỉnh $V(G)$ của đồ thị như đối với bài toán *MST*.

Để ngắn gọn, trong bài báo này từ *đồ thị* sẽ được hiểu là đơn đồ thị, vô hướng, liên thông, có trọng số không âm trên cạnh.

B. Ứng dụng của bài toán cây Steiner nhỏ nhất

Bài toán *SMT* có thể được tìm thấy trong các ứng dụng quan trọng như:

Bài toán thiết kế mạng truyền thông: Cho một mạng máy tính được biểu diễn bởi một đồ thị vô hướng $G = (V, E, f)$, một tập con T các đỉnh của V được gọi là nhóm truyền thông và một điểm s được gọi là một máy nguồn. Bài toán yêu cầu tìm một topology nối tất cả các máy trong nhóm

Tác giả liên hệ: Trần Việt Chương;
Email: chuongcm74@gmail.com;
Đến tòa soạn: 25/2/2021, chỉnh sửa: 9/4/2021, chấp nhận đăng: 16/4/2021.

truyền thông $T \cup s$ sao cho tổng số các kênh nối (trong một số trường hợp có thể là tổng độ dài các kênh nối) cần sử dụng là nhỏ nhất mà vẫn thỏa mãn các ràng buộc của mạng như tốc độ đường truyền, dung lượng băng thông, ...

Bài toán thiết kế hệ thống mạng cục bộ: Xét một hệ thống mạng cục bộ có dây truyền thống trong một cao ốc. Bộ khung vị trí các điểm phát mới cho WLAN có thể tận dụng hệ thống LAN sẵn có, nhưng không nhất thiết phải thực hiện tương tự. Giả sử đã biết tập các điểm phát được nối với nhau qua LAN và chi phí để kết nối một điểm phát đến một điểm kết nối định nghĩa bởi k_{ij}.

Trong trường hợp này, tập V gồm tất cả các điểm phát và mọi điểm kết nối đến hệ thống LAN sẵn có. Vì chỉ có các điểm phát mới giao tiếp với nhau nên S chỉ gồm những điểm phát và H là một đồ thị đầy đủ. Chi phí kết nối giữa hai điểm kết nối $i, j \in V \setminus S$ được định nghĩa là $k_{i,j} = 0$. Ứng dụng được biết đến như là bài toán Cây Steiner có chi phí nhỏ nhất. Tập con $U \subset V$ được gọi là tập các đỉnh cuối (terminal) liên thông với nhau, trong khi số đỉnh còn lại $V \setminus U$ được xem như là các tải trung tâm (hub) giúp giảm chi phí kết nối.

Bài toán thiết kế VLSI (Very Large Scale Integrated): Trong thiết kế vật lý của vi mạch VLSI, pha đi dây bao gồm việc xác định sơ đồ nối dây cho các điểm của từng khối hay từng cổng trên chip. Một mạng trên chip là tập hợp các điểm cần phải nối với nhau, thường bao gồm một điểm nguồn và nhiều điểm đích. Trong việc đi dây, sơ đồ kết nối được đặc tả cho một số lượng lớn các mạng. Kết nối thường được thực hiện trên một mạng cùng lúc. Mỗi phương án kết nối cho một mạng đơn là một Cây Steiner phủ các điểm nối.

Nhiều yêu cầu tối ưu đặt ra cho việc nối mạng phụ thuộc vào chức năng của từng mạng. Việc thiết kế VLSI, các đường nối chỉ là đường ngang dọc trên bảng mạch. Bài toán tối ưu chiều dài dây nối được đưa về một biến thể của bài toán Cây Steiner.

Các bài toán liên quan đến hệ thống mạng với chi phí nhỏ nhất [1,3,17,39,19]:

C. Một số nghiên cứu liên quan bài toán SMT

Bài toán *SMT* đã thu hút được sự quan tâm nghiên cứu của nhiều nhà khoa học trên thế giới trong hàng chục năm qua [6,7,9,10,11,12,13,23,25,41].

Hiện tại, có nhiều hướng tiếp cận giải bài toán cây Steiner nhỏ nhất như các thuật toán rút gọn đồ thị, các thuật toán tìm lời giải đúng, các thuật toán tìm lời giải gần đúng cận tỉ lệ, các thuật toán heuristic, các thuật toán metaheuristic.

1) Các thuật toán rút gọn đồ thị

Một số nghiên cứu trình bày các kỹ thuật nhằm giảm thiểu kích thước của đồ thị. Chẳng hạn công trình của Jeffrey H.Kingston và Nicholas Paul Sheppard [16], công trình của Thorsten Koch Alexander Martin [31], C. C. Ribeiro, M.C. Souza [5],…

Ý tưởng chung của các thuật toán rút gọn đồ thị là nhằm gia tăng các đỉnh cho tập terminal và loại bỏ các đỉnh của đồ thị mà nó chắc chắn không thuộc về cây Steiner nhỏ nhất cần tìm. Chất lượng các thuật toán giải bài toán *SMT* phụ thuộc vào độ lớn của hệ số $n - |L|$. Do vậy, mục đích của các thuật toán rút gọn đồ thị là làm giảm thiểu tối đa hệ số $n - |L|$.

Các thuật toán rút gọn đồ thị cũng được xem là bước tiền xử lý dữ liệu quan trọng trong việc giải bài toán *SMT*. Hướng tiếp cận này là rất cần thiết khi tiếp cận bài toán *SMT* bằng các thuật toán tìm lời giải đúng.

2) Các thuật toán tìm lời giải đúng

Các nghiên cứu tìm lời giải đúng cho bài toán *SMT* như thuật toán quy hoạch động của Dreyfus và Wagner [27], thuật toán dựa trên phép nới lỏng lagrange của Beasley [15], các thuật toán nhánh cận của Koch và Martin [31], Xinhui Wang [21,38],…

Ưu điểm của hướng tiếp cận này là tìm được lời giải chính xác, nhược điểm của hướng tiếp cận này là chỉ giải được các bài toán có kích thước nhỏ; nên khả năng ứng dụng của chúng không cao. Việc giải đúng bài toán *SMT* thực sự là một thách thức trong lý thuyết tối ưu tổ hợp.

Hướng tiếp cận này là cơ sở quan trọng có thể được sử dụng để đánh giá chất lượng lời giải của các thuật toán giải gần đúng khác khi giải bài toán *SMT*.

3) Các thuật toán gần đúng cận tỉ lệ

Thuật toán gần đúng cận tỉ lệ α nghĩa là lời giải tìm được gần đúng một cận tỉ lệ α so với lời giải tối ưu.

Ưu điểm của thuật toán gần đúng cận tỉ lệ là có sự đảm bảo về mặt toán học theo nghĩa cận tỉ lệ trên, nhược điểm của thuật toán dạng này là cận tỉ lệ tìm được trong thực tế thường là kém hơn nhiều so với chất lượng lời giải tìm được bởi nhiều thuật toán gần đúng khác dựa trên thực nghiệm.

Thuật toán *MST*-Steiner của Bang Ye Wu và Kun-Mao Chao có cận tỉ lệ 2 [3], thuật toán Zelikovsky-Steiner có cận tỉ lệ 11/6 [3]. Cận tỉ lệ tốt nhất tìm được hiện tại cho bài toán *SMT* là 1.39 [4,24,30].

4) 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 chính xác. Các thuật toán heuristic cũng không chắc hiệu quả trên mọi loại dữ liệu đối với một bài toán cụ thể.

Các thuật toán heuristic điển hình cho bài toán *SMT* như: SPT [32], PD Steiner [32] (bài báo này sẽ gọi tắt là PD), SPH [5], Heu [31], Distance network heuristic của Kou, Markowsky và Berman [18].

Ưu điểm của các thuật toán heuristic là thời gian chạy nhanh hơn nhiều so với các thuật toán metaheuristic. Giải pháp này thường được lựa chọn với các đồ thị có kích thước lớn [17,32,29,26].

5) 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 [40].

Hiện đã có nhiều công trình sử dụng thuật toán metaheuristic giải bài toán *SMT*. Chẳng hạn như thuật toán local search sử dụng các chiến lược tìm kiếm lân cận Node-Based Neighborhood [28], Path-Based Neighborhood [28], thuật toán Local Search [8], thuật toán tìm kiếm với lân cận biến đổi [35], thuật toán di truyền [2], thuật toán Tabu Search [5], thuật toán di truyền song song [20],…

Đóng góp chính của chúng tôi trong bài báo này là đã đề xuất hai chiến lược tìm kiếm lân cận mới để giải bài toán cây Steiner nhỏ nhất.

II. KHẢO SÁT MỘT SỐ CHIẾN LƯỢC TÌM KIẾM LÂN CẬN GIẢI BÀI TOÁN CÂY STEINER NHỎ NHẤT

A. Chiến lược Chèn cạnh - Xóa cạnh

Cho đồ thị vô hướng liên thông có trọng số G. Bắt đầu từ cây Steiner T của G được khởi tạo ngẫu nhiên, chèn lần lượt từng cạnh $e \in E(G) - E(T)$ vào cây Steiner T. Nếu cây Steiner T không chứa chu trình thì cạnh e không cần được xem xét; nếu $E(T) \cup \{e\}$ chứa chu trình thì tìm một cạnh e' trên chu trình này sao cho việc loại nó dẫn đến cây Steiner T' có chi phí là nhỏ nhất. Tiếp theo, nếu $C(T') < C(T)$ thì thay T bằng T'.

Thao tác tìm chu trình trong cây Steiner T sau khi chèn thêm một cạnh e được tiến hành như sau: Khi chèn cạnh $e = (u, v)$ vào T, duyệt cây Steiner T theo chiều sâu bắt đầu từ u, lưu vết trên đường đi bằng mảng p (đỉnh trước của một đỉnh trong phép duyệt). Tiếp theo, bắt đầu từ đỉnh v, truy vết theo mảng p đến khi gặp u thì kết thúc, các cạnh trên đường truy vết chính là các cạnh trong chu trình cần tìm [34].

B. Chiến lược tìm lân cận tốt hơn

Thủ tục tìm kiếm lân cận bắt đầu từ cây Steiner T được tiến hành như sau: Loại ngẫu nhiên khỏi T một cạnh e, chọn ngẫu nhiên cạnh e' từ tập $E - E(T)$, nếu tập cạnh $E(T) - \{e\} \cup \{e'\}$ cho ta cây Steiner T' có chi phí tốt hơn T thì ghi nhận cây Steiner này. Thủ tục trên sẽ được lặp lại k lần đối với cây Steiner T, trong số các cây Steiner được ghi nhận chọn ra T^* là cây Steiner tốt nhất, nếu cây Steiner T có chi phí tốt hơn T^* thì đặt $T = T^*$ [33].

C. Chiến lược tìm lân cận ngẫu nhiên tốt hơn

Tìm kiếm ngẫu nhiên cho cây Steiner T được tiến hành tương tự như tìm kiếm lân cận nhưng không quan tâm đến chi phí trên cạnh: Loại ngẫu nhiên cạnh e trong T, tìm ngẫu nhiên một cạnh e' từ tập $E - E(T)$ sao cho $E(T) - \{e\} \cup \{e'\}$ cho ta cây Steiner T' (không quan tâm đến việc T' có tốt hơn T hay không). Cập nhật $T = T$' và lặp lại k lần thao tác trên [33].

D. Chiến lược tìm lân cận Node-base

Cho cây Steiner T. Xét một đỉnh ngẫu nhiên $u \in T$ mà không thuộc tập đỉnh terminal L. Xóa đỉnh u và các cạnh liên quan đến đỉnh u trong T; khi đó T được phân rã thành nhiều thành phần liên thông; gọi đây là đồ thị H. Lần lượt bổ sung các cạnh theo thứ tự trọng số tăng dần của G - H vào đồ thị H cho đến khi H là một cây. Xóa các cạnh dư thừa trong H để H trở thành một cây Steiner T'. Nếu cây T' có chi phí tốt hơn T thì cập nhật T bằng T'; ngược lại hoặc là không tạo được cây T giữ nguyên [28].

E. Chiến lược tìm lân cận Path-based

Một key-node là một đỉnh của cây Steiner với bậc ít nhất là 3.

Một key-path là một đường đi với tất cả các đỉnh trung gian (không là đỉnh terminal) đều có bậc là 2 và hai đỉnh đầu cuối của đường đi đó hoặc thuộc tập terminal hoặc là đỉnh key-nodes.

Việc tìm key-path ngẫu nhiên được thực hiện như sau: Chọn ngẫu nhiên một cạnh trong T; nếu hai đỉnh đầu cuối có bậc 2 và là đỉnh Steiner thì thêm cạnh tiếp theo kề của đỉnh đó cho đến khi 2 đỉnh đầu cuối không phải bậc 2 và không phải Steiner thì dừng lại, và kiểm tra đường dẫn đó có phải key-path không. Nếu trong lúc thực hiện không thỏa mãn thì dừng.

T là một cây Steiner. Giả sử p một key- path ngẫu nhiên trên T; tiến hành loại bỏ p; khi đó T được tách thành hai thành phần liên thông T_1 và T_2;

Chọn cạnh ngắn nhất có thể nối hai thành phần liên thông T_1 và T_2 với nhau; giả sử khi đó ta được cây mới là T'.

Nếu cây T' có chi phí tốt hơn T thì cập nhật T bằng T'; ngược lại thì giữ nguyên T [28].

III. ĐỀ XUẤT MỘT SỐ CHIẾN LƯỢC TÌM KIẾM LÂN CẬN CHO CÂY STEINER

A. Chiến lược tìm kiếm lân cận tham lam (NS1)

Cho cây Steiner T. Loại ngẫu nhiên khỏi T một cạnh e, chọn ngẫu nhiên cạnh e' từ tập E - $E(T)$, nhưng cạnh e' thêm vào cây T thì ta lấy ngẫu nhiên từ tập những cạnh kề với tập đỉnh của cây T trong tập $E - E(T)$ trên tiêu chí là chi phí càng nhỏ thì tỉ lệ chọn càng cao. Sau khi loại cạnh e và thêm cạnh e' ta được cây Steiner T' có chi phí nhỏ hơn thì ghi nhận $T = T'$. Lặp lại k lần thủ tục này, ta được cây T có chi phí tốt nhất.

Mã giả của chiến lược tìm kiếm NS1 có thể được viết như sau :

Algorithm: ***NS1***

Input: Cây Steiner T ban đầu, k là số lần lặp.

Output: Cây Steiner T.

1: Khởi tạo tập cạnh xóa *e_del* từ tập cạnh của cây T:
e_del = $E(T)$;

2: Khởi tạo tập cạnh thêm *e_add* từ tập $E - E(T)$ và các cạnh này kề với tập đỉnh của cây T:
e_add = *adjacentEdges* (E – E(T));

3: Sắp xếp các cạnh trong *e_add* theo thứ tự tăng dần của trọng số cạnh;
4: ***for*** (*i = 1; i <= k; i++*) {
5: Chọn cạnh xóa *e = randomChoice*(*e_del*);
6: Chọn cạnh thêm *e'* = *priorityChoice*(*e_add*);
7: Hai cạnh được chọn *e, e'* thỏa mãn trọng số cạnh *e* lớn hơn trọng số *e'* (*e > e'*);
8: Sau khi loại cạnh *e* và thêm cạnh *e'* ta được cây Steiner *T'*;
9: ***if*** (*isSteinerTree*(*T'*) && *reduceTreeValue*(*T'*) < *reduceTreeValue*(*T*)) {
10: *T = T'*;
11: Cập nhập lại hai tập cạnh *e_del, e_add*;
12: }
13: }
14: Output *T*;

B. Chiến lược tìm kiếm lân cận có xác xuất (NS2)

Cho cây Steiner *T*, mỗi cạnh e thuộc *T* được gán cho một xác suất chọn *p(e)*; nếu trọng số của cạnh *e* càng bé thì xác xuất chọn *p(e)* càng cao. Loại ngẫu nhiên khỏi *T* một cạnh *e, e* là cạnh được chọn ngẫu nhiêu với việc ưu tiên chọn cạnh có chi phí lớn (chi phí cạnh càng lớn thì tỉ lệ chọn càng cao). Sau đó tìm ngẫu nhiên một cạnh *e'* từ tập *E – E(T)*, với *e'* là cạnh chọn ngẫu nhiêu với việc ưu tiên chọn cạnh có chi phí nhỏ (chi phí cạnh càng nhỏ thì tỉ lệ chọn càng cao) sao cho *E(T) – {e} ∪ {e'}* cho ta cây Steiner *T'* có chi phí tốt hơn *T* thì ghi nhận cây Steiner này *T = T'*. Lặp lại *k* lần thủ tục này, ta thu được cây *T* có chi phí tốt nhất.

Mã giả của chiến lược tìm kiếm NS2 có thể được viết như sau :

Algorithm: ***NS2***

Input: Cây Steiner *T* ban đầu, *k* là số lần lặp.

Output: Cây Steiner *T*;

1: Khởi tạo tập cạnh xóa *e_del* từ tập cạnh của cây *T*:
e_del = E(*T*);
2: Khởi tạo tập cạnh thêm *e_add* từ tập *E – E*(*T*) và các cạnh này kề với tập đỉnh của cây *T*:
e_add = adjacentEdges (*E – E*(*T*));
3: Sắp xếp các cạnh trong *e_add* theo thứ tự tăng dần của trọng số cạnh;
4: Sắp xếp các cạnh trong *e_del* theo thứ tự giảm dần của trọng số cạnh;
5: for (*i = 1; i <= k; i++*) {
6: Chọn cạnh xóa *e = priorityChoice*(*e_del*) và *e* không là cạnh treo;
7: chọn cạnh thêm *e'* = *priorityChoice*(*e_add*) và *e'* chứa 2 đỉnh thuộc T;
8: Sau khi loại cạnh *e* và thêm cạnh *e'* ta được cây *Steiner T'*;
9: if (*isSteinerTree*(*T'*) && *reduceTreeValue*(*T'*) < *reduceTreeValue*(*T*)) {
10: *T = T'*;
11: Cập nhập lại hai tập cạnh *e_del, e_add*;
12: }
13: }
14: Output *T*;

Trong đó:

- Hàm *adjacentEdges*() trả về tập cạnh kề với tập đỉnh *T* từ tập cạnh đầu vào.
- Hàm *randomChoice*() dùng để chọn ngẫu nhiên một cạnh từ tập cạnh đầu vào.
- Hàm *isSteinerTree*() dùng để kiểm tra có phải là cây Steiner hay không
- Hàm *reduceTreeValue*() dùng để tính chi phí của cây sau khi đã loại bỏ những cạnh dư thừa (cạnh treo chứa đỉnh treo không thuộc tập *terminal*)
- Hàm *priorityChoice*() dùng để chọn ưu tiên một cạnh từ tập cạnh đầu vào.

C. Thuật toán tìm kiếm lân cận biến đổi

Vấn đề lớn nhất mà các thuật toán metaheuristic gặp phải là nó dễ rơi vào bẫy tối ưu cục bộ. Để giải quyết vấn đề này, chúng tôi đề xuất việc kết hợp thuật toán tìm kiếm lân cận biến đổi với một số chiến lược tìm kiếm lân cận để hy vọng nâng cao chất lượng lời giải của thuật toán [37].

Ý tưởng của thuật toán tìm kiếm lân cận biến đổi (Variable Neighborhood Search - VNS) là thực hiện lần lượt từng chiến lược tìm kiếm lân cận, hết chiến lược tìm kiếm lân cận này đến chiến lược tìm kiếm lân cận khác; trong quá trình thực hiện thuật toán VNS, ta luôn ghi nhận lời giải tốt nhất (kỷ lục),

Khi thực hiện một chiến lược tìm kiếm lân cận, nếu tìm được kỉ lục mới thì ta quay trở lại thực hiện thuật toán VNS từ đầu. Ngược lại, ta chuyển sang chiến lược tìm kiếm lân cận tiếp theo; quá trình được tiếp tục cho đến khi thực hiện hết tất cả các chiến lược tìm kiếm lân cận mà lời giải tốt nhất không được cải thiện [22].

Mã giả của thuật toán VNS cơ bản có thể viết như sau:

- Gọi *S* là toàn bộ (hoặc một phần) của không gian lời giải của bài toán;
- Khởi tạo ngẫu nhiên giải pháp *s* thuộc *S*;
- Gọi $LS_1, LS_2, ..., LS_k$ là các thuật toán tìm lân cận của bài toán đang xét;

while (điều kiện dừng chưa được thỏa)
{
for (*i = 1; i <= k; i++*)
{
+ Gọi *s'* là lận cận của *s* với *s'* được tìm thấy bằng LS_i;
+ Nếu *s'* tốt hơn *s* thì break; // thoát khỏi vòng lặp for;
}
}

- *s* là lời giải tìm được của bài toán;

Trong đó, điều kiện dừng có thể sử dụng là: Khi thực hiện hết tất cả các chiến lược tìm kiếm lân cận mà lời giải tốt nhất không được cải thiện.

D. Sơ đồ thuật toán VNS giải bài toán SMT

T là cây Steiner ngẫu nhiên;
while (điều kiện dừng chưa thỏa) {
- Lần lượt thực hiện các chiến lược tìm kiếm lân cận *NS1, NS2* trên;

- Trong quá trình thực hiện các chiến lược tìm kiếm lân cận trên, ta ghi nhận lời giải tốt nhất;
- Khi thực hiện một chiến lược tìm kiếm lân cận, nếu tìm được kỉ lục mới thì ta quay trở lại thực hiện thuật toán VNS từ đầu (sau vòng lặp while); ngược lại, ta chuyển sang chiến lược tìm kiếm lân cận tiếp theo;
- Thuật toán VNS kết thúc khi điều kiện dừng được thỏa; điều kiện dừng được chúng tôi lựa chọn trong thuật toán này *10*n* với *n* là số đỉnh của đồ thị.

}

Trả về lời giải tốt nhất tìm được;

Mã giả của thuật toán VNS-NS có thể được viết như sau :

Algorithm: ***VNS-NS***

Input: *T* là cây Steiner ngẫu nhiên, *k* là số lần lặp cho mỗi chiến lược tìm kiếm lân cận, *k_vns* là số lần lặp cho thuật toán VNS-NS.

Output: Cây Steiner *T*;

```
1: T là cây Steiner ngẫu nhiên;
2: loop = 0;
3: while (loop < k_vns) {
4:         NS1(k);
5:         if (T được cải thiện) {
6:             loop = 0;
7:             continue;
8:             }
9:         NS2(k);
10:        if (T được cải thiện) {
11:            loop = 0;
12:            continue;
13:            }
14:            loop++;
15:            }
16:  Output T;
```

IV. THỰC NGHIỆM VÀ ĐÁNH GIÁ

Phần này, chúng tôi mô tả việc thực nghiệm hai chiến lược lân cận do chúng tôi đề xuất trên với hai metaheuristic đơn giản là thuật toán tìm kiếm lân cận (local search) và thuật toán tìm kiếm lân cận biến đổi (VNS); đồng thời đưa ra một số đánh giá về các kết quả đạt được.

A. Dữ liệu thực nghiệm

Để thực nghiệm các thuật toán liên quan, chúng tôi sử dụng 30 bộ dữ liệu gồm 15 đồ thị nhóm *steinc và* 15 đồ thị nhóm *steind* trong hệ thống dữ liệu thực nghiệm chuẩn dùng cho bài toán cây Steiner tại địa chỉ URL:

http://people.brunel.ac.uk/~mastjjb/jeb/orlib/steininfo.html [14].

B. Môi trường thực nghiệm

Thuật toán VNS-NS được chúng tôi cài đặt bằng ngôn ngữ C++ sử dụng môi trường DEV C++ 5.9.2; được thực nghiệm trên một máy chủ ảo Hệ điều hành Windows server 2008 R2 Enterprise, 64bit, Intel(R) Xeon (R) CPU E5-2660 @ 2.20 GHz, RAM 4GB.

C. Kết quả thực nghiệm và đánh giá

Thông tin dữ liệu thực nghiệm [14] được ghi nhận ở Bảng 1 và Bảng 2 có cấu trúc như sau: Cột đầu tiên (Test) là tên các bộ dữ liệu trong hệ thống dữ liệu thực nghiệm; số đỉnh (*n*), số cạnh (*m*) và số đỉnh thuộc tập *terminal* (|*L*|) của từng đồ thị.

Bảng 1. Thông tin về các đồ thị nhóm *steinc*

Test	*n*	*m*	|*L*|
steinc1.txt	500	625	5
steinc2.txt	500	625	10
steinc3.txt	500	625	83
steinc4.txt	500	625	125
steinc5.txt	500	625	250
steinc6.txt	500	1000	5
steinc7.txt	500	1000	10
steinc8.txt	500	1000	83
steinc9.txt	500	1000	125
steinc10.txt	500	1000	250
steinc11.txt	500	2500	5
steinc12.txt	500	2500	10
steinc13.txt	500	2500	83
steinc14.txt	500	2500	125
steinc15.txt	500	2500	250

Bảng 2. Thông tin về các đồ thị nhóm *steind*

Test	*n*	*m*	|*L*|
steind1.txt	1000	1250	5
steind2.txt	1000	1250	10
steind3.txt	1000	1250	167
steind4.txt	1000	1250	250
steind5.txt	1000	1250	500
steind6.txt	1000	2000	5
steind7.txt	1000	2000	10
steind8.txt	1000	2000	167
steind9.txt	1000	2000	250
steind10.txt	1000	2000	500
steind11.txt	1000	5000	5
steind12.txt	1000	5000	10
steind13.txt	1000	5000	167
steind14.txt	1000	5000	250
steind15.txt	1000	5000	500

Kết quả thực nghiệm của thuật toán được ghi nhận ở Bảng 3 và Bảng 4.

Bảng 3 ghi kết quả thực hiện của từng thuật toán MST [3], SPH [31], TS[31], NB[28], PB[28] và cột tiếp theo ghi nhận giá trị chi phí cây Steiner tương ứng với chiến lược tìm kiếm lân cận NS1, NS2 trong ngữ cảnh thuật toán tìm kiếm lân cận biến đổi được đặt tên là VNS-NS.

Bảng 3. Kết quả thực nghiệm trên nhóm đồ thị *steinc*

Test	MST	SPH	TS	NB	PB	VNS-NS
steinc1.txt	88	105	85	85	85	85
steinc2.txt	144	165	144	144	144	144
steinc3.txt	779	776	754	754	754	754
steinc4.txt	1114	1092	1079	1079	1079	1079
steinc5.txt	1599	1593	1579	1579	1579	1579
steinc6.txt	60	70	55	55	55	55
steinc7.txt	115	113	102	102	103	102
steinc8.txt	531	527	509	509	509	509
steinc9.txt	728	723	707	707	707	707
steinc10.txt	1117	1109	1093	1093	1093	1093
steinc11.txt	37	42	32	32	33	32
steinc12.txt	49	55	46	46	46	46
steinc13.txt	274	272	258	258	258	258
steinc14.txt	337	335	324	323	323	323
steinc15.txt	571	563	556	556	556	556

Bảng 4. Kết quả thực nghiệm trên nhóm đồ thị *steind*

Test	MST	SPH	TS	NB	PB	VNS-NS
steind1.txt	107	109	106	106	106	106
steind2.txt	237	243	220	220	220	220
steind3.txt	1636	1606	1567	1565	1565	1565
steind4.txt	2012	1975	1935	1935	1935	1936
steind5.txt	3310	3286	3250	3250	3254	3252
steind6.txt	74	85	70	68	70	67
steind7.txt	105	105	103	103	103	103
steind8.txt	1138	1106	1078	1072	1077	1073
steind9.txt	1540	1504	1450	1448	1449	1448
steind10.txt	2163	2148	2112	2110	2111	2113
steind11.txt	31	38	30	29	29	29
steind12.txt	43	49	42	42	42	42
steind13.txt	531	523	502	501	502	502
steind14.txt	702	699	667	669	667	671
steind15.txt	1151	1137	1117	1117	1120	1117

D. Đánh giá kết quả thực nghiệm

Mục này nhằm so sánh chất lượng lời giải của thuật toán VNS-NS với các thuật toán heuristic: MST [3], SPH [31], TS [31] và hai thuật toán local search NB [28], PB [28].

Kết quả có được do thực nghiệm thuật toán với số lần lặp $k = (n*m)/2$ (n là số đỉnh, m là số cạnh) cho mỗi chiến lược tìm kiếm lân cận NS1, NS2 và điều kiện để thuật toán VNS-NS kết thúc là sau $k\_vns = 10*n$ lần lặp mà kết quả của cây Steiner không được cải thiện (không tìm được kỷ lục mới thì dừng lại).

Với 30 bộ dữ liệu nhóm steinc và steind, thuật toán **VNS-NS** cho chất lượng lời giải (tốt hơn, bằng, kém hơn) so với các thuật toán **MST** [3], **SPH** [31], **TS** [31], **NB** [28], **PB** [28] lần lượt là: (96.7%, 3.3%, 0.0%), (100%, 0.0%, 0.0%), (20%, 66.7%, 13.3%), (3.3%, 76.7%, 20.0%), (23.3%, 66.7%, 10.0%).

V. KẾT LUẬN

Trong bài báo này, chúng tôi đã đề xuất hai chiến lược tìm kiếm lân cận mới và áp dụng chúng trong thuật toán tìm kiếm lân cận biến đổi để giải bài toán cây Steiner nhỏ nhất. Chúng tôi đã thực nghiệm thuật toán đề xuất này với 30 bộ dữ liệu là các đồ thị trong hệ thống dữ liệu thực nghiệm chuẩn. Kết quả thực nghiệm cho thấy rằng các chiến lược tìm kiếm lân cận này là hiệu quả và có thể sử dụng trong các thuật toán metaheuristic khác nhằm nâng cao chất lượng lời giải cho bài toán cây Steiner nhỏ nhất.

TÀI LIỆU THAM KHẢO

[1] ARIE M.C.A. KOSTER, XAVIER MUNOZ (Eds), "Graphs and algorithms in communication networks", Springer, pp.1-177, 2010.

[2] ANKIT ANAND, SHRUTI, KUNWAR AMBARISH SINGH, "An efficient approach for Steiner tree problem by genetic algorithm", International Journal of Computer Science and Engineering (SSRG-IJCSE), vol.2, pp.233-237, 2015.

[3] BANG YE WU, KUN-MAO CHAO, "Spanning trees and optimization problems", Chapman&Hall/CRC, pp.13-139, 2004.

[4] CHI-YEH CHEN, "An efficient approximation algorithm for the Steiner tree problem", National Cheng Kung University, Taiwan, 2018.

[5] C. C. RIBEIRO, M.C. SOUZA, "Tabu Search for the Steiner problem in graphs", Networks, 36, pp.138-146, 2000.

[6] CHI-YEH CHEN, "*An efficient approximation algorithm for the Steiner tree problem*", National Cheng Kung University, Taiwan, 2018.

[7] DAVIDE BILÒ (University of Sassari), "*New algorithms for Steiner tree reoptimization*", ICALP, 2018.

[8] EDUARDO UCHOA, RENATO F. WERNECK, "Fast local search for Steiner trees in graphs", SIAM, 2010.

[9] FICHTE JOHANNES K., HECHER MARKUS, and SCHIDLER ANDRÉ, **"***Solving the Steiner Tree Problem with few Terminals"*, University of Potsdam, Germany, 2020.

[10] HADRIEN CAMBAZARD, NICOLAS CATUSSE, "*Fixed-parameter algorithms for rectilinear steiner tree and rectilinear traveling salesman problem in the plane"*, Univ. Grenoble Alpes, France, 2019.

[11] IVANA LJUBIC, "*Solving Steiner Trees – Recent Advances*", Challenges and Perspectives, 2020.

[12] JACK HOLBY, "*Variations on the Euclidean Steiner Tree Problem and Algorithms*", St. Lawrence University, 2017.

[13] JOB KWAKERNAAK, "*Pipeline network optimization using Steiner nodes*", Wageningen University and Research Centre, The Netherlands, 2020.

[14] J. E. BEASLEY, OR-Library: URL http://people.brunel.ac.uk/~mastjjb/jeb/orlib/steininfo.html

[15] J. E. BEASLEY, "An SST-Based algorithm for the Steiner problem in graphs", Networks, 19, pp.1-16, 1989.

[16] JEFFREY H. KINGSTON, NICHOLAS PAUL SHEPPARD, "On reductions for the Steiner problem in graphs", Basser Department of Computer Science the University of Sydney, Australia, pp.1-10, 2006.

[17] JON WILLIAM VAN LAARHOVEN, "Exact and heuristic algorithms for the euclidean Steiner tree problem", University of Iowa, 2010 (Doctoral thesis).

[18] L. KOU, G. MARKOWSKY, L. BERMAN, "A Fast Algorithm for Steiner Trees", acta informatica, Vol.15, pp.141-145, 1981.

[19] M. HAUPTMANN, M. KARPINSKI (Eds), "A compendium on Steiner tree problems", pp.1-36, 2015.
[20] MARCELLO CALEFFI, IAN F. AKYILDIZ, LUIGI PAURA, "On the solution of the Steiner tree NP-Hard problem via physarum bionetwork", IEEE, pp.1092-1106, 2015.
[21] ONDRA SUCHÝ, "Exact algorithms for Steiner tree", Faculty of Information Technology, Czech Technical University in Prague, Prague, Czech Republic, 2014.
[22] Pierre Hansen, Nenad Mladenovic, Variable neighborhood search, GERAD and HEC Montreal, Canada, 2004.
[23] PROSENJIT BOSE, ANTHONY D'ANGELO, STEPHANE DUROCHER, "*On the Restricted 1-Steiner Tree Problem*", Funded in part by the Natural Sciences and Engineering Research Council of Canada, 2020.
[24] REYAN AHMED and et al, "Multi-level Steiner trees", ACM J. Exp. Algor., 1(1), 2018.
[25] **Radek Hušek, Dušan Knop, Tomáš Masaríik**, "*Approximation Algorithms for Steiner Tree Based on Star Contractions: A Unified View*", International Symposium on Parameterized and Exact Computation (IPEC 2020), ACM, 2020.
[26] STEFAN HOUGARDY, JANNIK SILVANUS, JENS VYGEN, "Dijkstra meets Steiner: a fast-exact goal-oriented Steiner tree algorithm", Research Institute for Discrete Mathematics, University of Bonn, pp.1-59, 2015.
[27] S. E. DREYFUS, R. A. WAGNER, "The Steiner Problem in Graphs", Networks, Vol.1, pp.195-207, 1971.
[28] S.L. MARTINS, M.G.C. RESENDE, C.C. RIBEIRO, AND P.M. PARDALOS, "A parallel grasp for the Steiner tree problem in graphs using a hybrid local search strategy", 1999.
[29] THOMAS BOSMAN, "A solution merging heuristic for the Steiner problem in graphs using tree decompositions", VU University Amsterdam, The Netherlands, pp.1-12, 2015.
[30] THOMAS PAJOR, EDUARDO UCHOA, RENATO F. WERNECK, "A robust and scalable algorithm for the Steiner problem in graphs", Springer, 2018.
[31] THORSTEN KOCH, ALEXANDER MARTIN, "Solving Steiner tree problems in graphs to optimality", Germany, pp.1-31, 1996.
[32] TRẦN VIỆT CHƯƠNG, PHAN TẤN QUỐC, HÀ HẢI NAM, "Đề xuất một số thuật toán heuristic giải bài toán cây Steiner nhỏ nhất", Kỷ yếu Hội nghị Quốc gia lần thứ X về Nghiên cứu cơ bản và ứng dụng Công nghệ thông tin (FAIR), Trường Đại học Đà Nẵng, ngày 17-18/08/2017, ISBN: 978-604-913-614-6, trang 138-147, 2017.
[33] TRẦN VIỆT CHƯƠNG, PHAN TẤN QUỐC, HÀ HẢI NAM, "Thuật toán Bees giải bài toán cây Steiner nhỏ nhất trong trường hợp đồ thị thưa". Tạp chí khoa học công nghệ thông tin và truyền thông, Học viên Công nghệ Bưu chính Viễn thông, Bộ Thông tin và Truyền thông, ISSN: 2525-2224, tháng 12, trang 24-29, 2017.
[34] TRẦN VIỆT CHƯƠNG, PHAN TẤN QUỐC, HÀ HẢI NAM, "Thuật toán tìm kiếm hill climbing giải bài toán cây Steiner nhỏ nhất", Các công trình nghiên cứu phát triển Công nghệ thông tin và Truyền thông, Bộ Thông tin truyền thông, 2020.
[35] TRAN VIET CHUONG, HA HAI NAM, "A variable neighborhood search algorithm forsolving the Steiner minimal tree problem in sparse graphs", Context-Aware Systems and Applications, and Nature of Computation and Communication, EAI, Springer, pp.218-225, 2018.
[36] VŨ ĐÌNH HÒA, "Bài toán Steiner", http://math.ac.vn.
[37] WASSIM JAZIRI (Edited). "Local Search Techniques: Focus on Tabu Search". In-teh, Vienna, Austria, pp.1-201, 2008.
[38] XINHUI WANG, "Exact algorithms for the Steiner tree problem", doctoral thesis, ISSN 1381-3617, 2008.
[39] XIUZHEN CHENG, DING-ZHU DU, "Steiner trees in industry", Kluwer Academic Publishers, vol.5, pp.193-216, 2004.
[40] XIN-SHE YANG. "Engineering optimization". WILEY, pp.21-137, 2010.
[41] YAHUI SUN, "*Solving the Steiner Tree Problem in Graphs using Physarum-inspired Algorithms*", 2019.

PROPOSE SOME NEIGHBORHOOD SEARCH STRATEGIES FOR THE STEINER MINIMUM TREE PROBLEM

Abstract: Steiner Minimum Tree (*SMT*) is a graph theory optimization problem that has many important applications in science and technology. This is a NP-hard problem. Many approaches have been developed to solve the Steiner Minimum Tree such as algorithms for finding exact solutions, Approximation Algorithms, algorithms heuristic algorithms, metaheuristic algorithms. In which, neighborhood search strategies are decisive for improving the quality solution of metaheuristic algorithms. In this paper, we propose two neighborhood search strategies for the Steiner Minimum Tree problem, and we use these neighborhood search strategies in the context of a variable neighbor search algorithm. Experimental results with the standard experimental data system show that our proposed algorithm offers better quality solution than some existing heuristic algorithms and metaheuristic algorithms on some data sets.

Trần Việt Chương

Sinh ngày: 01/12/1974

Cơ quan: Trung tâm CNTT và Truyền thông, Sở Thông tin và Truyền thông tỉnh Cà Mau.

Điện thoại: 0913 688 477

E-mail: chuongtv@camau.gov.vn

Tác giả nhận bằng Thạc sỹ Khoa học máy tính năm 2005 tại Trường Đại học Sư phạm I Hà Nội.

Lĩnh vực nghiên cứu: Thuật toán tiến hóa và tối ưu.

Phan Tấn Quốc

Sinh ngày: 12/10 /1971

Cơ quan: Bộ môn Khoa học Máy tính, khoa Công nghệ Thông tin, trường Đại học Sài Gòn.

Điện thoại: 0918 178 052

E-mail: quocpt@sgu.edu.vn

Tác giả nhận bằng Thạc sỹ Tin học tại trường Đại học KHTN - ĐHQG

TP.HCM năm 2002. Tác giả nhận bằng Tiến sỹ chuyên ngành Khoa học máy tính tại trường Đại học Bách Khoa Hà Nội năm 2015.

Lĩnh vực nghiên cứu: Thuật toán gần đúng.

Hà Hải Nam

Sinh ngày: 10/02/1975

Cơ quan: Viện Công nghiệp phần mềm và nội dung số Việt Nam, Bộ Thông tin và Truyền thông.

Điện thoại: 091 66 34567

E-mail: namhh@ptit.edu.vn

Tác giả nhận bằng Thạc sỹ Tin học năm 2004 tại Trường Đại học Bách Khoa Hà Nội, nhận bằng Tiến sỹ năm 2008 tại Vương quốc Anh. Được phong học hàm PGS năm 2014.

Lĩnh vực nghiên cứu: Hệ thống phân tán và tối ưu hóa.