(khả thi về thời gian chạy). Tiếp theo, chúng tôi sẽ trình bày ngắn gọn ý tưởng của các thuật toán này.

Thuật toán WONG: Tìm tất cả các cây đường đi ngắn nhất (*Shortest Path Tree - SPT*) [4] có gốc lần lượt tại các đỉnh của đồ thị, sau đó chọn ra trong số chúng một cây đường đi ngắn nhất có chi phí định tuyến nhỏ nhất [4].

Thuật toán SHC: Trước hết khởi tạo một cây khung lời giải ngẫu nhiên; sau đó từng bước cải thiện dần chất lượng lời giải. Ở mỗi bước lặp, *SHC* sinh ra một hoặc một số lời giải lân cận và sau đó chọn một lời giải tốt nhất trong số đó để làm lời giải hiện tại cho bước lặp kế tiếp. Việc tìm kiếm lân cận *T*' của cây khung *T* được thực hiện như sau: Loại ngẫu nhiên một cạnh *e* thuộc *T*, sau đó tìm cạnh *e*' tốt nhất từ tập cạnh $E - T$ để thay thế cho *e* (trong ký hiệu $E - T$; *E* là tập cạnh của đồ thị *G*, còn *T* là tập cạnh của cây khung *T*. Ký hiệu $G - T$ cũng được hiểu tương tự). Một lần lặp của thuật toán *SHC* đòi hỏi thời gian tính $O(mn)$. Thuật toán *SHC* ấn định số bước lặp là $10000n$. Vậy *SHC* đòi hỏi thời gian tính cỡ $10000n \times mn = 10000mn^2$ đối với mỗi bộ dữ liệu [5].

Thuật toán PBLS: Điểm khác biệt duy nhất của thuật toán *PBLS* so với thuật toán *SHC* là *PBLS* có sử dụng chiến lược đa dạng hóa lời giải - theo nghĩa là thay một số cạnh của cây khung bằng một số cạnh ngẫu nhiên khác. Thuật toán *PBLS* ấn định số bước lặp là 50000 và khi chất lượng lời giải không được cải thiện sau $2n$ bước lặp thì tiến hành đa dạng hóa lời giải. Một lần lặp của thuật toán *PBLS* đòi hỏi thời gian tính $O(mn)$. Vậy *PBLS* đòi hỏi thời gian tính cỡ $50000 \times mn = 50000mn$ đối với mỗi bộ dữ liệu [1].

Thuật toán REPRI: Bắt đầu từ một cây khung *T* được khởi tạo bằng kết quả của thuật toán *WONG*. Chiến lược tìm kiếm lân cận: Loại lần lượt từng cạnh $e \in T$, với mỗi cạnh *e* như vậy, tìm một cạnh $e' \in E - T$ sao cho $T' = (T - \{e\}) \cup \{e'\}$ có chi phí định tuyến nhỏ nhất. Nếu $C(T') < C(T)$ thì thay *T* bằng *T'* (thay cạnh *e* trong *T* bằng cạnh *e'* trong $E - T$). Thuật toán dừng nếu trong một lần duyệt qua tất cả các cạnh $e \in T$ mà không tìm được cạnh *e*' để cải thiện chi phí định tuyến của cây khung *T* [8].

Thuật toán REPIR: Bắt đầu từ một cây khung *T* được khởi tạo bằng cây khung nhỏ nhất theo thuật toán *KRUSKAL* hoặc thuật toán *PRIM*. Chiến lược tìm kiếm lân cận: Chèn lần lượt từng cạnh $e \in E-T$ vào cây khung *T,* khi đó $T \cup \{e\}$ sẽ chứa một chu trình, tìm một cạnh *e'* trên chu trình này sao cho việc loại nó dẫn đến cây khung *T'* có chi phí định tuyến là nhỏ nhất. Nếu $C(T') < C(T)$ thì thay *T* bằng *T'* (hoán đổi cạnh *e* trong $G - T$ với cạnh *e'* trong *T*). Thuật toán dừng nếu trong một lần duyệt qua tất cả các cạnh $e \in G - T$ mà không cải thiện được chi phí định tuyến của cây khung *T* [8].

1.3 Vấn đề tồn tại cần giải quyết

Tất cả các thuật toán đã khảo sát trên đều có chung cách làm là khi cần thay thế một cạnh nào đó thì phải duyệt qua tập tất cả các cạnh ứng viên để tìm ra cạnh tốt nhất; chính điều này làm cho các thuật toán trên không khả thi về thời gian khi số cạnh của đồ thị là đủ lớn.

Bài báo này đề xuất một thuật toán *HCST* dạng tìm kiếm leo đồi giải bài toán *MRCST*. Đề xuất này luôn cho chất lượng lời giải tốt hơn hoặc tương đương với các thuật toán tốt nhất hiện biết giải bài toán *MRCST* trên các đồ thị thưa. Đề xuất này thực sự có ý nghĩa đối với các đồ thị thưa có kích thước lớn.

2 ĐỀ XUẤT THUẬT TOÁN *HCST* GIẢI BÀI TOÁN *MRCST* TRONG TRƯỜNG HỢP ĐỒ THỊ THƯA

Thuật toán đề xuất *HCST* dựa vào chiến lược tìm kiếm lân cận như đã được đề cập trong *REPIR* [8]*; HCST* được bổ sung một số chiến lược mới nhằm tăng chất lượng lời giải và làm cho thuật toán khả thi khi làm việc với các đồ thị thưa có kích thước lớn.

2.1 Tạo lời giải ban đầu bằng thuật toán *tựa PRIM*

Khác với thuật toán *REPIR* cho khởi tạo cây khung ban đầu bằng thuật toán tìm cây khung nhỏ nhất của đồ thị theo thuật toán *KRUSKAL* hoặc thuật toán *PRIM*; *HCST* cho khởi tạo cây khung ngẫu nhiên bằng thuật toán *tựa* PRIM: Bắt đầu từ cây chỉ gồm một đỉnh nào đó của đồ thị, tiếp theo, thuật toán sẽ thực hiện n-1 bước lặp. Ở mỗi bước lặp, trong số các đỉnh chưa được chọn để tham gia vào cây, ta chọn một đỉnh kề với ít nhất một đỉnh nằm trong cây đang được xây dựng mà không quan tâm đến trọng số của cạnh. Đỉnh được chọn và cạnh nối nó với đỉnh của cây đang được xây dựng sẽ được bổ sung vào cây [9].

2.2 Chiến lược tìm kiếm lân cận

HCST thực hiện chiến lược tìm kiếm lân cận như đã trình bày trong thuật toán *REPIR* [8].

2.3 Giới hạn kích thước tập cạnh ứng viên

Trong các giáo trình về lý thuyết đồ thị và cấu trúc dữ liệu, nếu một đồ thị có số lượng cạnh lớn