hơn 5 lần số lượng đỉnh thì có thể xem đó là đồ thị dày. Với các đồ thị dày thì *HCST* làm việc như sau: Thay vì duyệt tất cả các cạnh ứng viên để tìm ra một cạnh tốt nhất thì *HCST* chỉ cần duyệt trong một tập con (tối đa 5*n* cạnh) ngẫu nhiên gồm các cạnh được chọn từ tập cạnh ứng viên. Trong khi đó, trước đây, thuật toán *REPIR* làm việc kém hiệu quả đối với các đồ thị dày có nhiều đỉnh.

2.4 Tăng tính ngẫu nhiên cho thuật toán

Hiệu quả của thuật toán *HCST* có thể được cải thiện khi ta thay đổi thứ tự các cạnh được duyệt trong tập $E - T$; nghĩa là ta sẽ duyệt tập cạnh này theo một hoán vị được sinh ngẫu nhiên chứ không theo một thứ tự cố định ở tất cả các lần duyệt (như thuật toán *REPIR*).

2.5 Cho thực hiện thuật toán nhiều lần trên mỗi bộ dữ liệu

Thuật toán *REPIR* thực hiện một lần duy nhất đối với mỗi bộ dữ liệu (do lời giải khởi tạo của *REPIR* không có tính ngẫu nhiên nên việc chạy nhiều lần sẽ không có tác dụng). Thuật toán *HCST* cho thực hiện nhiều lần đối với mỗi bộ dữ liệu.

2.6 Sơ đồ thuật toán *HCST*

Thuật toán *REPIR* [8] chỉ nêu ngắn gọn ý tưởng chiến lược tìm kiếm lân cận và không phân tích độ phức tạp tính toán của thuật toán. Trong bài báo này chúng tôi đã trình bày chi tiết nội dung thuật toán *HCST* và phân tích độ phức tạp tính toán của thuật toán *HCST* một cách chi tiết.

HCST (V,E)

Đầu vào: Đồ thị $G=(V, E)$

Đầu ra: Cây khung có chi phí định tuyến nhỏ nhất tìm được

1. stop=false;
2. T là cây khung được khởi tạo ngẫu nhiên bằng thuật toán *tựa PRIM*;
3. **while** (!stop){
4. stop=true;
5. $S= E - T$;
6. **for** (với mỗi cạnh $e \in S$){
7. $min= +\infty$;
8. $F=T \cup \{e\}$; // F có một chu trình
9. Xác định chu trình *Cycle* trong F chứa cạnh e;
10. **for** (với mỗi cạnh $e' \in Cycle$)
11. **if** ($C(F - \{e'\}) < min$){
12. $min= C(F - \{e'\})$;
13. Ghi nhận $T'= F - \{e'\}$;
14. }
15. **if** ($min < C(T)$){
16. $T=T'$;
17. stop=false;
18. } // end if dòng 15
19. } // end for dòng 6
20. } // end while dòng 3
21. **return** T;

2.7 Đánh giá độ phức tạp của *HCST*

Độ phức tạp một lần lặp của thuật toán *HCST* được đánh giá bởi thời gian tính hai vòng lặp **for** lồng nhau ở dòng 6 và 10. Câu lệnh **if** dòng 11 có độ phức tạp $O(n)$, vòng lặp **for** ở dòng 10 chứa câu lệnh **if** trên lặp $O(n)$ lần; do đó có độ phức tạp là $O(n^2)$. Thao tác tìm các cạnh trong chu trình ở dòng 9 có độ phức tạp $O(n)$. Vòng lặp **for** ở dòng 6 lặp m-(n-1) lần. Do đó vòng **for** này có độ phức tạp là $O(n^2)$. Gọi k là giá trị lớn nhất mà vòng lặp **while** ở dòng 3 có thể thực hiện, thì độ phức tạp một lần lặp của thuật toán *HSCT* là $O(kn^2m)$.

Trong khi các thuật toán *SHC*, *PBLS* đưa ra số lần lặp lần lượt là 10000 n^2m và 50000 nm để đạt được kết quả như công bố (đã được khảo sát ở trên); thì thuật toán *HCST* với cách thức tìm lân cận đã nêu có giá trị k khá nhỏ. Với đồ thị thưa; chẳng hạn là 1000 đỉnh, qua thống kê trên các thực nghiệm của chúng tôi thì giá trị k tối đa chỉ bằng 10. Thời gian tính của các thuật toán *HCST* nhanh hơn so với nhiều thuật toán metaheuristic khác, đặc biệt là khi làm việc với các đồ thị thưa có kích thước lớn.

3 THỰC NGHIỆM VÀ ĐÁNH GIÁ

Chúng tôi tiến hành thực nghiệm thuật toán *HCST* và so sánh chất lượng lời giải và thời gian tính của *HCST* với các thuật toán tốt nhất hiện biết giải bài toán *MRCST* trong trường hợp đồ thị thưa như *WONG, CAMPOS, SHC, PBLS*.

Do hiện tại chưa có một công trình nào công bố chất lượng lời giải đối với các đồ thị thưa có kích thước lớn, nên chúng tôi đã cài đặt lại các thuật toán *WONG, SHC, PBLS* trên cùng môi trường thực nghiệm với thuật toán *HCST*. Để kiểm tra chất lượng các chương trình do chúng tôi cài đặt, chúng tôi đã đối sánh output từ các chương trình do chúng tôi cài đặt với output của các công trình khác đối với các đồ thị đầy đủ euclid và đồ thị đầy đủ ngẫu nhiên. Riêng thuật toán *CAMPOS* chúng tôi được