THUẬT TOÁN BẦY ONG GIẢI BÀI TOÁN CÂY KHUNG VỚI CHI PHÍ ĐỊNH TUYẾN NHỎ NHẤT
BEES ALGORITHM FOR SOLVING MINIMUM ROUTING COST SPANNING TREE PROBLEM

PHAN TẤN QUỐC[1], NGUYỄN ĐỨC NGHĨA[2]

[1]Khoa công nghệ thông tin, Trường Đại học Sài Gòn, quocpt@sgu.edu.vn
[2]Viện công nghệ thông tin và truyền thông, Trường Đại học Bách Khoa Hà Nội, nghiand@soict.hut.edu.vn

Abstract. Minimum Routing Cost Spanning Tree (MRCST) is one of spanning tree optimization problems having several applications in network design. In general cases, the problem is proved as NP-hard. This paper proposes an algorithm for solving MRCST problem based on the schema of bee algorithm. The computational experiment results show that our proposal algorithm outperforms Wong algorithm, population-based metaheuristics like Max-Min Ant System (MMAS), genetic algorithm (GA), Artificial Bee Colony algorithm (ABC), and some well-known heuristic algorithms.

Tóm tắt. Bài toán cây khung với chi phí định tuyến nhỏ nhất (Minimum Routing Cost Spanning Tree - MRCST) là một trong số các bài toán cây khung tối ưu có nhiều ứng dụng trong lĩnh vực thiết kế mạng. Trong trường hợp tổng quát, bài toán được chứng minh là NP- khó. Bài báo này đề xuất thuật toán giải bài toán MRCST được phát triển dựa trên sơ đồ thuật toán bầy ong. Các kết quả tính toán thực nghiệm cho thấy thuật toán đề xuất cho lời giải với chất lượng tốt hơn thuật toán xấp xỉ Wong và một số thuật toán metaheuristics dạng quần thể như Max-Min Ant System (MMAS), thuật toán di truyền (GA), thuật toán Artificial Bee Colony (ABC) và một số thuật toán heuristic hiện biết.

1. GIỚI THIỆU BÀI TOÁN

Mục này trình bày phát biểu bài toán MRCST và khảo sát một số thuật toán giải MRCST được đề xuất trong những năm gần đây.

1.1. Phát biểu bài toán

Định nghĩa 1. Chi phí định tuyến

Cho $G = (V,E,w)$ là đồ thị vô hướng liên thông có trọng số (chi phí) không âm trên cạnh; trong đó V là tập đỉnh, E là tập cạnh. Giả sử T là một cây khung của G, chi phí định tuyến (routing cost) của T, ký hiệu là $C(T)$, là tổng các chi phí định tuyến giữa mọi cặp đỉnh thuộc cây T, trong đó chi phí định tuyến của một cặp đỉnh (u,v) trên T, ký hiệu là $d_T(u,v)$, là tổng chi phí của các cạnh trên đường đi nối đỉnh u với đỉnh v trên cây T. Như vậy theo định nghĩa ta có:

$$C(T) = \sum_{u,v \in V} d_T(u,v) \quad (1)$$

Bài toán đặt ra là tìm một cây khung với chi phí định tuyến nhỏ nhất trong số tất cả các cây khung của đồ thị G. Xây dựng cây khung chi phí định tuyến nhỏ nhất cũng tương đương với việc xây dựng cây khung sao cho độ dài trung bình giữa mọi cặp đỉnh là nhỏ nhất. Bài toán có ý nghĩa ứng dụng quan trọng trong thiết kế mạng, chẳng hạn trong việc xây dựng các hệ thống mạng; đặc biệt là ở các mạng ngang hàng khi các nút có xác suất truyền tin và độ ưu tiên là như nhau (về xuất xứ và ứng dụng của bài toán có thể xem trong [1,2,3]).

Việc tính toán chi phí định tuyến của một cây khung với n đỉnh trong bài toán MRCST theo đúng định nghĩa sẽ đòi hỏi thời gian $O(n^2)$. Tuy nhiên, trên cơ sở khái niệm "tải năng định tuyến" ("routing load") ta có thể tính chi phí định tuyến một cây khung với độ phức tạp tuyến tính [2].

Định nghĩa 2. Tải năng định tuyến

Cho một cây khung T với tập cạnh $E(T)$. Nếu loại khỏi T một cạnh e thì T sẽ được tách thành hai cây con là T_1 và T_2 với tập đỉnh tương ứng là $V(T_1)$ và $V(T_2)$. Tải năng định tuyến của cạnh e được định nghĩa là: $l(T,e) = 2\,|V(T_1)|\,.\,|V(T_2)|$. Khi đó công thức (1) là tương đương với công thức (2) sau đây:

$$C(T) = \sum_{e \in E(T)} l(T,e).w(e)\ . \qquad (2)$$

Bài toán MRCST được xác định là bài toán thuộc lớp NP– khó [2]. Trọng số trên mỗi cạnh và cấu trúc của cây khung là hai yếu tố tác động lớn đến chi phí định tuyến cây khung, trong đó yếu tố cấu trúc cây khung là quan trọng hơn đối với những đồ thị mà trọng số các cạnh là không quá cách biệt. Hiện nay ngoài phương pháp giải đúng bằng thuật toán nhánh cận [19], đã có nhiều phương pháp giải gần đúng bài toán MRCST dựa trên các cách tiếp cận khác nhau như các xấp xỉ, heuristics, metaheuristics.

1.2. Khảo sát các thuật toán giải MRCST

Thuật toán xấp xỉ

Thứ nhất là thuật toán Wong được đề xuất bởi Richard Wong từ năm 1980, thuật toán Wong có độ chính xác xấp xỉ 2 và có độ phức tạp là $O(nm + n^2\log n)$. Thuật toán Wong sử dụng khái niệm cây đường đi ngắn nhất (shortest path tree – SPT) [1,2]; cây đường đi ngắn nhất có gốc tại đỉnh r là cây được hợp thành từ các đường đi ngắn nhất bắt đầu từ đỉnh r đến tất cả các đỉnh còn lại của đồ thị. Ý tưởng của thuật toán Wong là tìm các SPT xuất phát từ mỗi đỉnh của đồ thị, sau đó chọn ra một SPT có chi phí thấp nhất – đây chính là kết quả của thuật toán Wong. *Thứ hai* là thuật toán dựa trên ý tưởng General Star của nhóm tác giả Bang Ye Wu và Kun-Mao Chao; thuật toán này có độ chính xác xấp xỉ 3/2 và có độ phức tạp là $O(n^4)$. Nhóm tác giả này cũng đưa ra một sơ đồ xấp xỉ thời gian đa thức (Polynomial Time Approximation Scheme - PTAS) cho phép tìm được cây khung có chi phí định tuyến xấp xỉ $(1+\varepsilon)$ lần chi phí định tuyến của cây khung tối ưu, trong đó ε là độ tốt mà ta mong muốn [2]. Độ phức tạp của thuật toán này là $O\left(n^{2\lceil 2/\varepsilon \rceil - 2}\right)$. *Thứ ba* là thuật toán Add được đề xuất bởi Vic Grout vào năm 2005 [3]. Thuật toán Add có độ phức tạp là $O(n \log n)$. *Thứ tư* là thuật toán Campos được đề xuất bởi nhóm tác giả Rui Campos và Manuel Ricardo vào năm 2008; thuật toán này có độ chính xác xấp xỉ 2 và có độ phức tạp là $O(m + n \log n)$[7].

Có thể nói các thuật toán xấp xỉ trên tuy chưa tìm được lời giải có độ chính xác cao nhưng có ưu điểm về thời gian và sự đảm bảo về chất lượng lời giải [1,2,3,7].

Thuật toán heuristic và metaheuristic

Hướng tiếp cận bằng các thuật toán *heuristic* điển hình là các thuật toán *thay thế cạnh* và *xóa dần cạnh* [13][18]. Ý tưởng chung của các heuristic này là từ một cây khung ban đầu nào đó, ta từng bước cải thiện để thu được cây khung tốt hơn; cây khung ban đầu có thể là cây đường đi ngắn nhất tìm được bằng thuật toán Wong.

Hướng tiếp cận bằng các thuật toán metaheuristic đã được áp dụng cho bài toán MRCST như thuật toán di truyền [14,20], thuật toán bầy kiến [15], các thuật toán bầy ong [16],…

Có thể nói các thuật toán metaheuristic là hướng tiếp cận tiềm năng cho bài toán MRCST. Bài báo này trình bày thuật toán giải MRCST được phát triển dựa trên sơ đồ thuật toán bầy ong.

2. THUẬT TOÁN BẦY ONG

Trong tự nhiên bầy ong mật tìm kiếm thức ăn theo quy trình sau: Đầu tiên các ong do thám sẽ được cử đi thăm dò các vùng thức ăn tiềm năng; các ong do thám sẽ di chuyển ngẫu nhiên từ bụi hoa này sang bụi hoa khác; sau đó chúng quay về tổ và thông tin cho cả bầy về hướng di chuyển, khoảng cách từ tổ đến vùng thức ăn và chất lượng của các vùng thức ăn; những thông tin này sẽ làm cho bầy ong bay đến các vùng thức ăn nhanh chóng và chính xác hơn. Một vấn đề tự nhiên nhưng là điểm trọng tâm cho ý tưởng thuật toán bầy ong là: *Nơi nào có thức ăn dồi dào hơn thì nơi đó sẽ có nhiều ong được cử đến hơn.*

Các thuật toán mô phỏng theo quá trình tìm kiếm thức ăn của loài ong mật (gọi chung là các thuật toán bầy ong) đã được biết đến với các đề xuất ban đầu của nhóm tác giả Dusan Teodorovic (2010) [17] và hiện đã có nhiều nhóm tác giả phát triển như Karaboga và Basturk (2006) [10], Alok Singh và Shyam Sundar (2011) [16]; Duc Truong Pham (2005) [4,5]; Xing-She Yang (2010)[11,12],… Sơ đồ thuật toán ong cơ bản mà chúng tôi phát triển trong bài báo này là thuật toán của các nhóm tác giả Duc Truong Pham (2005) [4,5] và Xing-She Yang (2010)[11,12]. Để thấy được sự khác biệt của một nhóm tác giả khác khi tiếp cận giải bài toán MRCST cũng bằng thuật toán dựa trên sơ đồ thuật toán bầy ong (ABC) [16], chúng tôi nêu ngắn gọn lại ý tưởng của thuật toán ABC như sau: Đầu tiên chọn n_e lời giải ngẫu nhiên $E_1, E_2,\ldots, E_{n_e}$. Gọi *best_sol* là lời giải tốt nhất trong số n_e lời giải đó. Lặp lại ba công đoạn sau đến khi nào điều kiện dừng được thỏa: T*hứ nhất,* với mỗi i ($i = 1..n_e$), gọi E' là một lân cận của E_i. Nếu $E' = \varnothing$ thì E_i được thay bằng một lời giải ngẫu nhiên nào đó. Nếu $E' \neq \varnothing$ thì xét các trường hợp sau: Nếu E' tốt hơn E_i thì $E_i = E'$; ngược lại, nếu E_i không thay đổi qua một số lần lặp xác định thì E_i sẽ được thay bằng một lời giải ngẫu nhiên nào đó. Và nếu E_i tốt hơn *best_sol thì best_sol* = E_i. T*hứ hai,* với mỗi i ($i = 1..n_o$), gọi P_i là một số ngẫu nhiên trong tập chỉ số $\{1,2,\ldots n_e\}$. Gọi S_i là lời giải lân cận của E_{P_i}. Nếu $S_i = \varnothing$ thì S_i được gán bằng một lời giải tùy ý có chi phí kém hơn chi phí của E_{P_i}. Nếu S_i tốt hơn *best_sol* thì đặt *best_sol* = S_i. T*hứ ba, v*ới mỗi i ($i = 1..n_o$), nếu S_i tốt hơn E_{P_i} thì đặt $E_{P_i} = S_i$.

Thuật toán bầy ong sử dụng hai chiến lược trọng tâm là tìm kiếm lân cận và tìm kiếm ngẫu nhiên. Thuật toán bầy ong được cho là có khả năng thoát khỏi tối ưu cục bộ và do đó nó có thể tìm được lời giải tối ưu toàn cục; và do đó nó được đánh giá là một trong những thuật toán metaheuristic thích hợp cho việc giải các bài toán tối ưu tổ hợp khó. Sơ đồ tổng quát của thuật toán bầy ong được mô tả như sau.

Thuật toán bầy ong

1. Khởi tạo quần thể ban đầu với n vùng; mỗi vùng chứa duy nhất một cá thể. Các cá thể này được chọn ngẫu nhiên hoặc bằng một thuật toán heuristic.
2. Đánh giá độ thích nghi cho mỗi cá thể của quần thể.
3. **while** (*điều kiện dừng* chưa thoả)
4. Trong n vùng ban đầu, chọn ngẫu nhiên p vùng để thực hiện tìm kiếm lân cận ($p < n$) và trong p vùng này chọn tiếp ra e vùng có độ thích nghi cao nhất.
5. Mỗi vùng trong e vùng sẽ được tuyển thêm *nep* ong để tìm lân cận quanh nó.
6. Mỗi vùng trong $p - e$ vùng được chọn sẽ được tuyển thêm *nsp* ong để tìm lân cận (trong đó *nep* > *nsp*).
7. Mỗi ong trong $n - p$ vùng còn lại được thay thế bằng một ong ngẫu nhiên khác.
8. Đánh giá độ thích nghi cho tất cả các ong ở mỗi vùng.
9. Mỗi vùng chọn ra duy nhất một ong có độ thích nghi cao nhất.

   endwhile

Sơ đồ tổng quát thuật toán bầy ong

Trong thuật toán trên, *điều kiện dừng* được chọn là số lần lặp định trước.

Thuật toán bầy ong có một loạt các tham số:

- n là số vùng được thăm, mỗi vùng tương ứng với một ong do thám - mỗi vùng được *hiểu là chứa một cá thể hoặc một số cá thể được sinh ra từ một cá thể nào đó.*
- p là số vùng được chọn trong số n vùng được xét.
- e là số vùng tốt nhất trong số p vùng được chọn. e vùng viết tắt là e-vùng.
- *nep* là số ong cử đến mỗi vùng thuộc e-vùng.
- *nsp* là số ong cử đến mỗi vùng thuộc $p-e$ vùng được chọn còn lại; $p-e$ vùng viết tắt là *pe*-vùng.
- $n-p$ vùng viết tắt là *np*-vùng.

Thuật toán bầy ong bắt đầu với n ong do thám được khởi tạo ngẫu nhiên hoặc bằng các hàm heuristic. Sau đó thuật toán đánh giá độ thích nghi (giá trị của hàm mục tiêu) của các cá thể cho từng vùng. Ở mỗi bước lặp, các ong do thám sẽ được phân công vào ba loại vùng: loại e-vùng, loại *pe*-vùng và loại *np*-vùng. Mỗi vùng thuộc e-vùng sẽ được tìm thêm *nep* lân cận, mỗi vùng thuộc *pe*-vùng sẽ được tìm thêm *nsp* lân cận và mỗi vùng thuộc *np*-vùng sẽ được thay thế trực tiếp bằng một lân cận ngẫu nhiên. Sau đó mỗi vùng chỉ chọn đúng một ong có độ thích nghi cao nhất để xây dựng quần thể ong ở thế hệ tiếp theo. Trong quá trình thực hiện thuật toán, ta cần lưu lại cá thể tốt nhất tìm được để khi thuật toán kết thúc nó chính là kết quả cần tìm.

3. THUẬT TOÁN BẦY ONG GIẢI BÀI TOÁN CÂY KHUNG VỚI CHI PHÍ ĐỊNH TUYẾN NHỎ NHẤT

Mục này sẽ đề xuất thuật toán BEE-MRCST để giải bài toán MRCST. Trước hết ta sẽ trình bày một số vấn đề liên quan khi áp dụng thuật toán bầy ong vào bài toán MRCST.

3.1. Mã hóa cây khung

Một phương pháp mã hóa cây khung được xem là hiệu quả nếu nó thỏa mãn được các tiêu chí như: tận dụng được những tính chất đặc trưng của cây khung, quá trình mã hóa và giải mã cây khung là không quá phức tạp, thời gian để xử lý các phép toán liên quan trên cây khung cũng như không gian để lưu trữ các cây khung là hợp lý, khả năng di truyền các gen tốt cho thế hệ sau cũng phải được đảm bảo [6,8,20]. Hiện có nhiều phương pháp mã hoá cây khung được sử dụng: mã hóa dạng nhị phân (binary-based encoding), mã hóa dạng cạnh (edge-based encoding), mã hóa dạng đỉnh (node-based encoding).

Mã hóa nhị phân: Mỗi cây khung được mã hóa thành một chuỗi gồm $n-1$ bít, trong đó bít thứ i của dãy mã mang giá trị 0 hoặc 1 tùy theo cạnh thứ i của đồ thị có tham gia vào cây khung tương ứng với dãy mã đó hay không.

Mã hóa dạng đỉnh: Mỗi cây khung được mã hóa thành một chuỗi gồm $n-1$ số nguyên, trong đó số nguyên thứ i là chỉ số đỉnh đầu của các cạnh trong phép duyệt cây khung theo chiều rộng (hoặc theo chiều sâu) và chỉ số cuối của các cạnh được cho cố định từ 2 đến n.

Mã hóa dạng cạnh: Mỗi cây khung được mã hóa thành một chuỗi gồm $n-1$ số nguyên, trong đó mỗi số nguyên là chỉ số cạnh của đồ thị có tham gia vào cây khung.

Các cách mã hóa này không quan tâm đến trọng số trên các cạnh của cây khung [8] và việc giải mã cho các cách mã hóa này là đơn giản. Đối với các bài toán có tính đến trọng số trên cạnh thì phương pháp mã hóa dạng cạnh được đánh giá là thích hợp nhất. Thuật toán đề xuất trong bài báo này sẽ sử dụng phương pháp mã hóa dạng cạnh.

3.2. Tạo quần thể ban đầu

Các cá thể trong quần thể ban đầu cần được phân bố sao cho chúng có thể lấy mẫu được phần lớn các vùng của không gian tìm kiếm nhằm tạo được tính đa dạng của quần thể. Đây là yếu tố quan trọng đối với các thuật toán metaheuristic nói chung. Quần thể ban đầu thường được tạo bằng các thuật toán heuristic hoặc sinh ngẫu nhiên đơn giản hoặc kết hợp cả hai cách này.

Trong bài báo này, chúng tôi sử dụng hai cách sau để tạo quần thể ban đầu.

- Mỗi cá thể là một cây đường đi ngắn nhất (SPT) có gốc xuất phát từ mỗi đỉnh của đồ thị (tìm được nhờ sử dụng thuật toán Dijkstra).
- Mỗi cá thể được tạo thành theo thuật toán *tựa* Prim: việc chọn cạnh tiếp theo để đưa vào cây *T* không dựa vào trọng số của cạnh như thuật toán Prim chuẩn mà chọn ngẫu nhiên một cạnh nào đó kề với tập cạnh đã được chọn sao cho không tạo thành chu trình trong *T*.

Cách đầu tiên tạo ra các cá thể cây khung bảo đảm được các yếu tố về trọng số của cạnh và cấu trúc của cây khung, nhưng chúng chỉ tạo được quần thể với kích thước không vượt quá số đỉnh của đồ thị. Trong khi đó thuật toán *tựa* Prim cho phép tạo quần thể với kích thước không hạn chế, có thể đảm bảo tính đa dạng của quần thể nhưng chất lượng của quần thể sẽ không cao khi *n* là đủ lớn. Trong thực nghiệm chúng tôi chọn cách thứ nhất để tạo quần thể; còn cách thứ hai dùng để tạo các lời giải ngẫu nhiên được sử dụng trong thuật toán.

Đoạn mã giả tạo quần thể ban đầu có *n* cá thể bởi thuật toán thứ nhất như sau:

```
initializePopulation(G,n,m)          // Khởi tạo quần thể
{
   population = ∅ ;
   for (i=1; i<=n; i++)
        { T=SPT(G,i); population = population ∪ T; }
}
```

3.3. Độ thích nghi của cá thể

Độ thích nghi của mỗi cá thể là chi phí định tuyến của cây khung tương ứng với cá thể đó. Để tính chí phí định tuyến của mỗi cây khung ta áp dụng công thức (2).

Đoạn mã giả để tính chi phí định tuyến cho các cá thể của quần thể được mô tả như sau:

```
evaluateFitness (population)         // Tính độ thích nghi của mỗi cá thể thuộc quần thể
{
  for (mỗi cây khung T thuộc quần thể)
       routingCost(T);
}

routingCost(T)                       // Tính chi phí của cây khung T
{
   s = 0;
   for (mỗi cạnh e ∈ T)
       s = s + routingLoad(T,e)*w(e) ;
   return s;
}
```

3.4. Chọn cá thể cho mỗi loại vùng

Chọn ngẫu nhiên *p* vùng từ *n* vùng ban đầu, tiếp theo là sắp xếp *p* vùng được chọn này theo độ thích nghi giảm dần. Khi đó trong *p* cá thể được sắp thì *e* cá thể đầu tiên sẽ thuộc ***e*-vùng**, *p* – *e* cá thể tiếp theo sẽ thuộc ***pe*-vùng**, các cá thể còn lại (chưa được sắp xếp) sẽ thuộc ***np-vùng***.

Sau đây là đoạn mã giả chọn cá thể cho mỗi loại vùng.

```
selectSties(population,n,p,e)
{
   d=0; populationtemp=∅; dachon[n]; // mảng này được khởi tạo bằng giá trị 0
   while (d < p)
   {
        k=random(p)+1;
        if (dachon[k]=0) // cá thể thứ k chưa được chọn
                 {populationtemp = populationtemp ∪ Tk; dachon[k]=1;}
```

```
        d++;
    }
        Sắp xếp các cá thể trong quần thể populationtemp và chọn các vùng như sau:
        for (i=1; i<=e ;i++)        e-vùng=e-vùng ∪ Ti;
        for (i=e+1; i<=p ;i++)      pe-vùng=pe-vùng ∪ Ti;
        for (i=1; i<=n ;i++)        if (dachon[i]=0)   np-vùng=np-vùng ∪ Ti;
    }
```

3.5. Tìm kiếm lân cận

Việc tìm kiếm lân cận *T*' cho một cây khung *T* được thực hiện như sau: Loại ngẫu nhiên khỏi *T* một cạnh *e*, sau đó tìm một cạnh *e*' tốt nhất từ tập $E - T$ để thay thế và nếu $T - e + e$' tốt hơn *T* thì đặt $T' = T - e + e$'.

Sau đây là các đoạn mã giả cho việc tìm kiếm lân cận của mỗi loại vùng.

```
eSitesNeighSearch(T,e,nep)          // Tìm kiếm lân cận để sinh thêm nep cây khung
{
    for mỗi cây khung T thuộc vùng e-vùng
    for (i =1 ;i<=nep ;i++)
    {
        findNeighST(T,T');          // T' là lân cận của T
        if (routingCost (T') < routingCost (T))   T=T';
    }
}

peSitesNeighSearch(T,pe,nsp)        // Tìm kiếm lân cận để sinh thêm nsp cây khung
{
    for mỗi cây khung T thuộc vùng pe-vùng
    for (int i =1 ;i<=nsp ;i++)
    {
        findNeighST (T,T');         // T' là lân cận của T
        if (routingCost (T') < routingCost (T))   T=T';
    }
}

findNeighST (T,T')          // Tìm cây khung lân cận
    {
        Xóa ngẫu nhiên một cạnh e thuộc T;
        Tìm cạnh e' tốt nhất trong E – T để thay thế;
        if (routingCost (T – e + e') < routingCost (T))
                T = T – e + e';
        T=T';
    }
```

3.6. Tìm kiếm ngẫu nhiên

Việc tìm kiếm ngẫu nhiên diễn ra ở giai đoạn cuối của mỗi bước lặp khi $n - p$ ong được cử đi tìm kiếm ngẫu nhiên. Mỗi cá thể trong số *np*-vùng này được thay thế bằng một cá thể được sinh ngẫu nhiên mà bỏ qua điều kiện là cá thể ngẫu nhiên này tốt hơn cá thể trước đó ở mỗi vùng đó.

Sau đây là đoạn mã giả việc tìm kiếm ngẫu nhiên.

```
npSitesRandSearch(T,np)             // Tìm cây khung ngẫu nhiên
{
        for (mỗi cây khung T thuộc vùng np-vùng)
                {randSearch(T'); T=T' ;}
```

```
}
randSearch(T)  // Tạo cây khung ngẫu nhiên theo thuật toán tựa Prim
{
   T={u};        // u được chọn ngẫu nhiên
   while (|T|<n)
       {
          Chọn ngẫu nhiên đỉnh v∉T sao cho v có kề với một đỉnh u nào đó với u ∈T
          T = T ∪ {v};
       }
}
```

3.7. Tìm cá thể tốt nhất của quần thể

```
findBestIndividual(population,Tbest)        // Tìm cá thể tốt nhất của quần thể
{
       for (mỗi cây khung T của quần thể population)
            if (routingCost (T) < routingCost (Tbest))
                 Tbest=T ;
}
```

Cuối cùng thuật toán BEE-MRCST được mô tả như sau:

```
BEE-MRCST()
{
initializePopulation(G,n,m);
evaluateFitness(population);
t=0;
while (t<maxn)
     {
          selectSties(population,n,p,e);
          eSitesNeighSearch(T,e,nep);
          peSitesNeighSearch(T,pe,nsp);
          npSitesRandSearch(T,np); t++;
     }
findBestIndividual(population,Tbest) ;
return Tbest ;
}
```

4. KẾT QUẢ THỰC NGHIỆM

4.1.Môi trường thực nghiệm, các tham số thực nghiệm

Thuật toán BEE-MRCST được cài đặt trên ngôn ngữ *C++* sử dụng trình biên dịch CFREE 5.0 và chạy trên máy tính cấu hình 4GB RAM, CPU INTEL 2.20GHz.

Kết quả thực nghiệm của thuật toán đề xuất BEE-MRCST được so sánh với các thuật toán Wong [15], MMAS [15], GA [14] và ABC [16].

Bài báo sẽ tiến hành thực nghiệm thuật toán trên đồ thị tổng quát, đồ thị đồng nhất, đồ thị có các cạnh phân bố đều giữa các đỉnh, đồ thị có các cạnh phân bố không đều giữa các đỉnh, đồ thị đầy đủ. Dữ liệu thực nghiệm được phát sinh ngẫu nhiên như đề xuất từ các bài báo [15,16]. Với mỗi bộ dữ liệu, chương trình BEE-MRCST được chạy 10 lần và kết quả tốt nhất của 10 của lần chạy được ghi nhận làm kết quả của thuật toán.

Trong thực nghiệm BEE-MRCST chúng tôi đề xuất các giá trị của tham số là: Số cá thể trong quần thể là *n,* số vùng được chọn để tìm kiếm lân cận là $p = 2n/3$, số vùng được khai phá lân cận

nhiều hơn là $e=p/3$, số ong được cử đến mỗi vùng thuộc e-vùng là $nep=5$, số ong được cử đến mỗi vùng thuộc pe-vùng là $nsp=3$. Thực nghiệm được tiến hành với đồ thị có số đỉnh nằm trong đoạn [20..100] và số cạnh nằm trong đoạn [50..1200]. Chi phí định tuyến trong các bảng thực nghiệm được ghi nhận bằng ½ giá trị tính theo công thức (2).

4.2.Xây dựng bộ dữ liệu

Các bộ dữ liệu thực nghiệm (INPUT) và kết quả thực nghiệm (OUTPUT) được sử dụng trong bài báo này được lưu trữ chi tiết tại trang WEB[1].

Đồ thị tổng quát

Các đồ thị tổng quát $G = (V,E,w)$ được sinh như sau: đầu tiên xây dựng ngẫu nhiên một cây khung gồm $n = |V|$ đỉnh và $n - 1$ cạnh, sau đó lần lượt thêm ngẫu nhiên $m - (n - 1)$ cạnh hợp lệ khác; trọng số các cạnh của đồ thị là số nguyên ngẫu nhiên trong đoạn [1..2500].

Đồ thị đồng nhất

Đồ thị đồng nhất và đồ thị gần đồng nhất (là loại đồ thị mà trọng số các cạnh khác nhau không đáng kể - trong bài báo này sẽ gọi chung là đồ thị đồng nhất).

Các đồ thị đồng nhất $G = (V,E,w)$ được sinh như sau: đầu tiên chọn một giá trị ngẫu nhiên làm giá trị đồng nhất cho các cạnh, giả sử là $\Delta \in [6..2500]$. Sau đó xây dựng ngẫu nhiên một cây khung gồm $n = |V|$ đỉnh và $n - 1$ cạnh, và cuối cùng là thêm ngẫu nhiên $m-(n - 1)$ cạnh khác. Tất cả các cạnh của đồ thị được gán cho một giá trị nguyên dương $\Delta \pm \mu$ trong đó với μ là số nguyên nhỏ.

Đồ thị có các cạnh được phân bố đều

Đồ thị có các cạnh được phân bố đều giữa các đỉnh là đồ thị mà bậc của các đỉnh là tương đương nhau hoặc chênh lệch là nhỏ, không đáng kể.

Các đồ thị có các cạnh được phân bố đều $G = (V,E,w)$ được sinh như sau: đầu tiên ta xác định tham số $r = 2 \times [m/n] + 1$ gọi là số lượng cạnh trung bình của mỗi đỉnh; sau đó xây dựng ngẫu nhiên một cây khung gồm $n = |V|$ đỉnh và $n - 1$ cạnh sao cho bậc của một đỉnh bất kỳ không vượt quá r, tiếp tục là thêm ngẫu nhiên $m - (n - 1)$ cạnh hợp lệ khác và đảm bảo bậc của một đỉnh không vượt quá r; trọng số các cạnh của đồ thị là số nguyên ngẫu nhiên trong đoạn [1..2500].

Đồ thị có các cạnh được phân bố không đều

Các đồ thị có các cạnh phân bố không đều $G = (V,E,w)$ được sinh như sau: đầu tiên, ta chọn ngẫu nhiên k đỉnh ($[n/2] + 1 \le k \le n-1$), gán cho các đỉnh này một số nguyên $r \in \{1, 2\}$ cho biết các đỉnh này sẽ không được có quá r cạnh nối trong đồ thị; tiếp theo, ta xây dựng cây khung và các cạnh như trong trường hợp đồ thị tổng quát. Chú ý nếu không xây dựng được đủ m cạnh thì ta lặp lại quá trình từ đầu.

Đồ thị đầy đủ

Dễ dàng tạo các đồ thị ngẫu nhiên cho các đồ thị đầy đủ này. Đồ thị có n đỉnh có số cạnh là $m = (n - 1)*n/2$, trọng số các cạnh của đồ thị là số nguyên ngẫu nhiên trong đoạn [1..2500].

4.3.Phân tích và đánh giá kết quả thực nghiệm

Chi phí định tuyến

Nội dung các bảng thực nghiệm gồm các cột: Mã số các bộ test, số lượng đỉnh, số lượng cạnh của đồ thị tương ứng, kết quả cho bởi các thuật toán Wong (xấp xỉ), MMAS, GA, ABC, BEE-MRCST (đều ghi kết quả tốt nhất của 10 lần chạy; số thế hệ là 350).

Bảng 1.a (WEB[1]) cho kết quả thực nghiệm qua 18 bộ test và so sánh kết quả thu được từ thuật toán BEE-MRCST với các thuật toán Wong, MMAS, GA, ABC. Chúng tôi cũng đánh giá thuật toán qua 18 đồ thị tổng quát ngẫu nhiên được lấy tại trang WEB[2] và kết quả cho như ở bảng 1.b (WEB[1]). Tương tự, các bảng 2,3,4,5 (WEB[1]) là kết quả thực nghiệm ứng với các đồ thị đồng nhất,

đồ thị có các cạnh được phân bố đều giữa các đỉnh, đồ thị có các cạnh được phân bố không đều giữa các đỉnh và đồ thị là đầy đủ.

Các kết quả thu được trong các bảng 1-5 được tổng kết trong bảng 6 dưới đây. Nội dung của bảng 6 cho biết số lượng (SL) bộ test cho kết quả tốt hơn/bằng nhau/kém hơn khi so sánh thuật toán BEE-MRCST với các thuật toán Wong, MMAS, GA, ABC. Bảng 6 có so sánh riêng cho từng dạng đồ thị và tổng hợp cho tất cả các dạng đồ thị đã đề cập. Bảng 6 cho thấy với hầu hết các bộ test thuật toán BEE-MRCST cho kết quả tốt hơn các thuật toán Wong, MMAS, GA; còn so sánh với thuật toán ABC (với bản cài đặt của chúng tôi) thì BEE có nhỉnh hơn về chi phí định tuyến nhưng hiệu quả hơn về thời gian chạy. BEE-MRCST cũng cho kết quả tốt hơn các heuristic giải bài toán MRCST trước đây như thuật toán *thay thế cạnh* và thuật toán *xóa dần cạnh* [13].

Bảng 6. So sánh thuật toán BEE-MRCST với các thuật toán khác

Kết quả	**Wong**		**MMAS**		**GA**		**ABC**	
	SL	*%*	*SL*	*%*	*SL*	*%*	*SL*	*%*
Dạng các đồ thị tổng quát (36 test)								
Tốt hơn	34	94.4	24	66.7	15	41.7	1	2.8
Bằng nhau	2	5.6	12	33.3	21	58.3	35	97.2
Kém hơn	0	0.0	0	0.0	0	0.0	0	0.0
Dạng các đồ thị đồng nhất (15 test)								
Tốt hơn	15	100.0	15	100.0	15	100.0	3	20.0
Bằng nhau	0	0.0	0	0.0	0	0.0	12	80.0
Kém hơn	0	0.0	0	0.0	0	0.0	0	0.0
Dạng các đồ thị có các cạnh được phân bố đều (15 test)								
Tốt hơn	14	93.3	11	73.3	8	53.3	1	6.7
Bằng nhau	1	6.7	4	26.7	7	46.7	14	93.3
Kém hơn	0	0.0	0	0.0	0	0.0	0	0.0
Dạng các đồ thị có các cạnh được phân bố không đều (15 test)								
Tốt hơn	14	93.3	11	73.3	3	20.0	0	0.0
Bằng nhau	1	6.7	4	26.7	12	80.0	15	100.0
Kém hơn	0	0.0	0	0.0	0	0.0	0	0.0
Dạng các đồ thị đầy đủ (15 test)								
Tốt hơn	15	100.0	10	66.7	4	26.7	0	0.0
Bằng nhau	0	0.0	5	33.3	11	73.3	15	100.0
Kém hơn	0	0.0	0	0.0	0	0.0	0	0.0
Tổng hợp cho các dạng đồ thị (96 test)								
Tốt hơn	92	95.8	71	74.0	45	46.9	5	5.2
Bằng nhau	4	4.2	25	26.0	51	53.1	91	94.8
Kém hơn	0	0.0	0	0.0	0	0.0	0	0.0

Thời gian thực hiện các thuật toán

Thời gian chạy trung bình mỗi test theo thuật toán MMAS, GA, ABC, *thay thế cạnh, xóa dần cạnh*, BEE-MRCST lần lượt là 92.7 giây, 15.8 giây, 175.4 giây, 6.6 giây, 70.6 giây, 26.1 giây.

4.4. Thực nghiệm trên một hệ thống test khác

Ngoài hệ thống test đã được mô tả như trên, chúng tôi cũng đã thực nghiệm trên một hệ thống test khác (WEB[1]). Hệ thống test này vẫn duy trì 78 bộ test cho năm dạng đồ thị như trên (hệ thống này không có các đồ thị ở bảng 1.b; đây là đồ thị mẫu được sử dụng rộng rãi cho việc thử nghiệm bài toán cây steiner); trong đó mỗi đồ thị được giữ nguyên số đỉnh và số cạnh còn cấu trúc và trọng số các cạnh thì được cho ngẫu nhiên. Bảng 7 so sánh chi phí định tuyến và phần trăm tương ứng của BEE-MRCST so với các thuật toán đã đề cập.

Bảng 7. So sánh chi phí BEE-MRCST với các thuật toán khác

Kết quả	Wong		MMAS		GA		ABC	
	SL	*%*	*SL*	*%*	*SL*	*%*	*SL*	*%*
Tốt hơn	70	89.7	50	64.1	34	43.6	6	7.7
Bằng nhau	8	10.3	27	34.6	44	56.4	72	92.3
Kém hơn	0	0.0	1	1.3	0	0.0	0	0.0

5. KẾT LUẬN

Bài báo đề xuất thuật toán BEE-MRCST được phát triển dựa trên sơ đồ thuật toán bầy ong để giải bài toán MRCST. Thuật toán BEE-MRCST đã được cài đặt và thử nghiệm trên 2 hệ thống test được sinh ngẫu nhiên với 174 bộ test. Kết quả thực nghiệm cho thấy thuật toán BEE-MRCST cho chất lượng lời giải cao hơn các thuật toán hiện biết. Tuy nhiên, việc tăng hiệu quả thời gian của BEE-MRCST là vấn đề cần tiếp tục được xem xét. Đó là vấn đề mà chúng tôi sẽ giải quyết trong những nghiên cứu tiếp theo.

TÀI LIỆU THAM KHẢO

[1] R. Wong, Worst-case analysis of network design problem heuristics, *SIAM J. Algebra. Discr*., pp.51–63, 1980.

[2] Bang Ye Wu, Kun-Mao Chao, Spanning Trees and Optimization Problems, *Chapman&Hall/CRC*, 2004, pp.13–139.

[3] V. Grout, Principles of cost minimization in wireless networks, *Journal of Heuristics 11*, 2005, pp.115–133.

[4] Duc Truong Pham, Ghanbarzadeh A, Koc E, Otri S, Rahim Sand Zaidi M, The Bees Algorithm, *Technical Note, Manufacturing Engineering Centre, Cardiff University, UK*, 2005.

[5] Duc Truong Pham, Ghanbarzadeh A., Koc E., Otri S., Rahim S., and M.Zaidi, The Bees Algorithm - A Novel Tool for Complex Optimisation Problems, *Proceedings of IPROMS 2006 Conference*, pp.454–461.

[6] Lin Lin, Mitsuo Gen, Node-Based Genetic Algorithm for Communication Spanning Tree Problem, *IEICE TRANS. COMMUN, VOL.E89–B, NO.4 APRIL* 2006, pp.1091-1098.

[7] Rui Campos, Manuel Ricardo, A fast Algorithm for Computing Minimum Routing Cost Spanning Trees, *Computer Networks, Volume 52, Issue 17*, 2008, pp.3229-3247.

[8] Mitsuo Gen, Runwei Cheng, Lin Lin, Network Models and Optimization - Multiobjective Genetic Approach, *Springer*, 2008, pp.1-96.

[9] Karaboga D, An idea based on honey bee swarm for numerical optimization, *Erciyes University, Turkey*, 2005.

[10] Basturk B, Karaboga D, An artificial bee colony (ABC) algorithm for numeric function optimization, *IEEE Swarm Intelligence Symposium*, 2006.

[11] Xing-She Yang, Engineering Optimization, *WILEY*, 2010, pp.197–202.

[12] Xing-She Yang, Nature-Inspired Metaheuristic Algorithms, *LUNIVER PRESS*, 2010, pp.53–62.

[13] Phan Tan Quoc, A Heuristic Approach for Solving the Minimum Routing Cost Spanning Tree Problem, *IJMLC 2012, IACSIT*, pp.V2.406-409.

[14] Phan Tan Quoc, A Genetic Approach for Solving the Minimum Routing Cost Spanning Tree Problem, *IJMLC 2012, IACSIT*, pp.V2.410-414.

[15] Nguyen Minh Hieu, Phan Tan Quoc, Nguyen Duc Nghia, An Approach of Ant Algorithm for Solving Minimum Routing Cost Spanning Tree Problem, *SoICT 2011*, ACM, pp.5-10.

[16] Alok Singh, Shyam Sundar, An artificial bee colony algorithm for the minimum routing cost spanning tree problem, *Soft Computing - A Fusion of Foundations, Methodologies and Applications, Volume 15, Number 12*, 2489-2499, DOI: 10.1007/s00500-011-0711-,6, Springer-Verlag 2011.

[17] Dusan Teodorovic, Bee Colony Optimization, *Belgrade, Serbia*, 2010.

[18] Alok Singh, A New Heuristic for the Minimum Routing Cost Spanning Tree Problem, *International Conference on Information Technology, IEEE*, 2008.

[19] M. Fischetti, G. Lancia, and P. Serafini, Exact algorithms for minimum routing cost trees, *Networks, vol.39*, 2002, pp.161-173.

[20] B. A. Julstrom, The Blob code is competitive with edgesets in genetic algorithms for the minimum routing cost spanning tree problem, *Proceedings of the Genetic and Evolutionary Computation Conference 2005 (GECCO-2005), Hans-Georg Beyer et al.. Eds, vol. 1,ACM Press, New York*, 2005, pp. 585–590.

[1] http://my.opera.com/phantanquoc

[2] http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files