BÁO CÁO THUẬT TOÁN ECFE

Mục lục

DANH SÁCH CHỮ VIẾT TẮT, KÍ HIỆU

CHỮ VIẾT TẮT, KÍ HIỆU	Ý NGHĨA
PDFS(Piecemeal DFS)	Duyệt theo chiều sâu chia từng phần
B	Giới hạn năng lượng của mỗi tuyến
DFS	Duyệt theo chiều sâu
C(surPlus)	Lượng dư thừa sau mỗi tuyến
PDFS offline	Duyệt theo chiều sâu biết trước môi trường
ECFE	Bài toán khám phá rừng giới hạn năng lượng

Chương 1

GIỚI THIỆU

Ở [1], nói đến việc khám phá cây (tree) bằng cách là dùng Piecemeal DFS. Và tài liệu này đã cung cấp cho ta về cách duyệt, cách chia tuyến như thế nào để có thể bao phủ toàn bộ cây. Nhưng có một nhược điểm là khám phá chỉ trên một cây và sau mỗi tuyến khám phá hầu hết là không dùng hết năng lượng B đã cho. Cho nên khi về gốc lượng dư thừa này bị bỏ qua.
Giả sử như ta muốn duyệt n cây(rừng) thì theo như tài liệu [1], ta phải duyệt từng cây một, từ đó lượng dư thừa cho việc khám phá này cực lớn. Do đó mà ta phải xem xét dùng lượng dư thừa này hỗ trợ trong quá trình khám phá để tiết kiệm chi phí khám phá.
Bài báo cáo này ra đời với mục đích đó, giúp ta khám phá trên n cây và tối ưu lượng dư thừa này.

Chương 2

ĐẶT VẤN ĐỀ

2.1 BÀI TOÁN ECFE

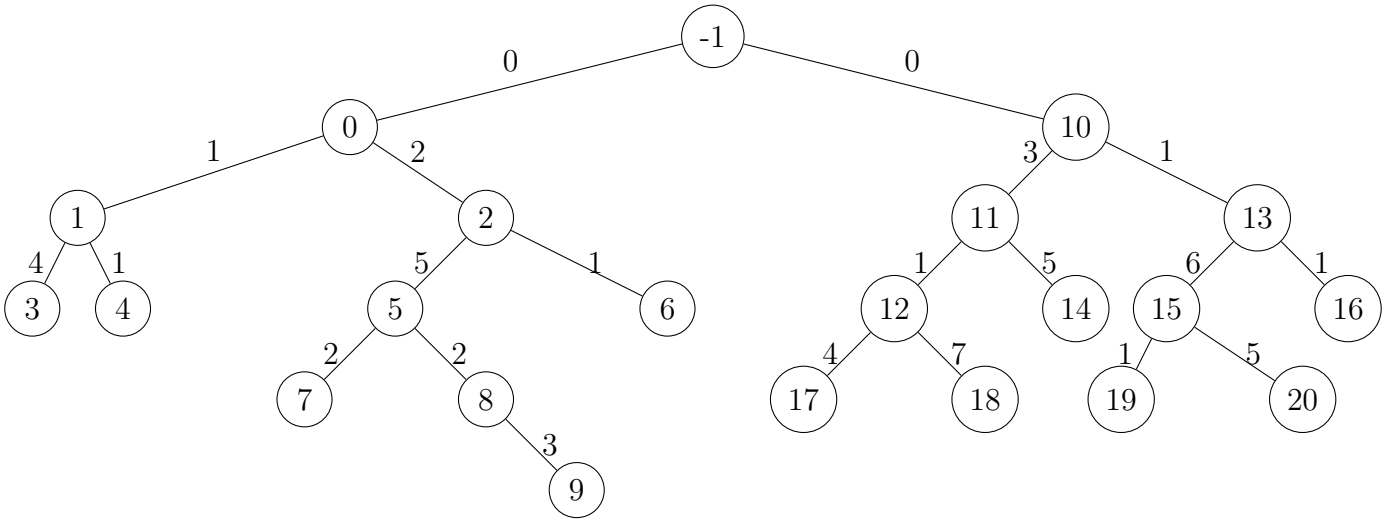
Bài toán ECFE(Energy Constrained Forest Exploration problem) là bài toán khám phá rừng cây có trọng số với giới hạn năng lượng B>1 và tổng trọng từ gốc đến lá xa nhất trong cây là B/2. Bên cạnh đó còn tối ưu lượng dư thừa trong quá trình khám phá
Mục tiêu: Khám phá toàn bộ rừng này với số tuyến k là ít nhất. Mỗi tuyến bắt đầu và kết thúc tại gốc

2.2 KIẾN THỨC LIÊN QUAN

2.2.1 MÔI TRƯỜNG KHÁM PHÁ

Môi trường khám phá của ta ở bài này là rừng(n cây), xử lý môi trường khám phá như sau:

- Ta có 1 gốc ảo (-1), những gốc thực của các cây liên kết với gốc ảo với trọng số là 0. Và gốc ảo này cũng chính là trạm sạc để robot r khám phá nạp năng lượng.
- Chỉ số các gốc được đánh từ 1 đến n. Ví dụ:



Hình 2.1: Minh họa rừng 2 cây

Hình 2.1 trên là một minh họa cơ bản cho rừng cây và cách xử lý rừng cây trong việc khám phá.

2.2.2 CƠ SỞ LÝ THUYẾT

Các định nghĩa

- Rừng là tập hợp các cây liên kết với nhau nhưng không có cạnh chung, có sự khác biệt ở đây là rừng khám phá của ta có một gốc ảo(-1) để liên kết các cây rời rạc với nhau(xem hình 2.1).
- Chi phí đường đi $R = (v_0,\ v_1,\ \dots v_n)$: là tổng trọng số của các cạnh thuộc tuyến đó

$$l(R) = \sum_{i=0}^{n} l_i$$

  Với $l(R)$ là tổng chi phí của một tuyến , còn l_i là trọng số của cạnh
- Chiến lược khám phá B là là tập các tuyến đường R_i với $i = (1 \dots k)$, hay $S = (R_1,\ R_2,\ \dots,\ R_k)$, với $R_i \leq B$. Ta kí hiệu $|S| = k$ là số tuyến đường của quá trình khám phá
- Chi phí khám phá:

$$\xi(S) = \sum_{i=1}^{k} R_i$$

  Nói theo cách khác đây là chi phí của chiến lược PDFS(T), kí hiệu $\xi(PDFS(T))$

Quá trình duyệt

Bước 1: Đặt v_{-1} là gốc quay về(r)
Bước 2: Thực hiện duyệt PDFS,

- Tuyến đầu tiên: R_1 duyệt như DFS bình thường nhưng thỏa công thức:

$$d\left(r,\ v_{i-1}\right) + l\left(v_{i-1},\ v_i,\ \dots v_p\right) + d\left(v_p,\ r\right) \leq B \tag{2.1}$$

  Lúc này điểm dừng của ta là tại v_{i-1}. Sau khi kết thúc tuyến R_1 (đã quay về gốc) ta có thêm một lượng dư $C_1 = B - \xi(R_1)$
- Tuyến thứ hai trở đi:
  Sau khi khám phá đã thỏa mãn công thức 2.1, ta chưa quay về gốc ngay mà xem xét là tại nút ta đang dừng đó v_p có thể xét thêm nhánh nào nhờ lượng dư thừa C không
  + Nếu được thì thêm nút có thể duyệt vào tuyến hiện tại, còn nếu không thể thì quay về và lượng C được cộng dồn vào $C_i = C_{i-1} + (B - \xi(R_i))$
- Lặp cho đến khi duyệt hết rừng

Các định lý

Theo [1], ta có hai định lý quan trọng như sau:

Định lý 1: Định lý giới hạn về số tuyến Cho cây T và B, khoảng cách xa nhất từ gốc đến lá bé hơn hoặc bằng B/2, ta có $|PDFS(T)| \leq 12\,|\mathcal{R}|$ với $\mathcal{R}$ là chiến lược khám phá mà có số tuyến nhỏ nhất

Định lý 2: Định lý giới hạn về chi phí Cho T và B/2 là lớn hơn hoặc bằng khoảng cách xa nhất từ gốc đến lá. Ta có $\xi\left(PDFS(T)\right) \leq 12\xi(COPT(T))$.
$\xi\left(COPT(T)\right)$ là chi phí tối ưu nhất trong chiến lược khám phá.
Về cách chứng minh hai định lý này, xem trong tài liệu [1].

Định lý 3: Định lý về giới hạn của C Giới hạn trên của C là B. Nghĩa là $0 \leq C \leq B$.
Chứng minh định lý 3:
Ta có: $\sum_{i=1}^{k} C = k.B - \sum_{i=1}^{k} \xi(R_i)$.
Vì $\xi(R_i) \geq 0$ cho nên $\sum_{i=1}^{k} C \leq k.B$.
Hay ta có: $C_1 + C_2 + C_3 \ldots + C_k \leq k.B$. Dấu bằng xảy ra khi và chỉ khi $C_1 = C_2 = C_3 = \ldots = C_n = B$. Để xảy ra trường hợp này thì buộc $\xi(R_i) = 0$.
Từ đó ta được $C \leq B$.

Chương 3

THUẬT TOÁN

3.1 CƠ SỞ THUẬT TOÁN

Dựa vào nội dung mục 2.2.2, ta có thể thực hiện thuật toán và giải bài toán này trên máy tính với các cấu trúc dữ liệu cây, hàm,... trên ngôn ngữ C++.

Cấu trúc dữ liệu

1. Khai báo biến toàn cục:
   - numTrees: Số lượng cây trong rừng
   - B: Ngân sách năng lượng
   - edgesList: Danh sách cạnh (from, to, weight)
2. Cấu trúc Node:
   - id: Định danh nút
   - children: Danh sách các nút con và trọng số cạnh
   - parent: Nút cha
   - visited: Đánh dấu đã duyệt
3. Cấu trúc PathResult:
   - path: Danh sách các nút từ đích về gốc
4. Lớp Tree:
   - Thuộc tính:
     - root: Nút gốc ảo (-1)
     - B: Ngân sách
     - nodes: Bảng băm lưu các nút theo id
     - distanceFromRoot: Khoảng cách từ gốc đến mỗi nút
   - Phương thức:
     - getOrCreate(id):
       * Nếu nút chưa tồn tại → tạo mới
       * Trả về nút tương ứng
     - assignDistanceFromRoot(node, currentLength, visitedSet):
       * Đệ quy tính khoảng cách từ gốc đến tất cả nút con
     - findRoots(edgeList):
       * Tìm tất cả gốc thực (nút không là con của bất kỳ nút nào)
     - Constructor(edgeList, _B):
       * Xây dựng cây từ danh sách cạnh
       * Tạo gốc ảo (-1) và kết nối với các gốc thực
       * Tính toán khoảng cách từ gốc
     - path_to_root(target):
       * Trả về đường đi từ target về gốc
     - PDFS_offline():

- Khởi tạo:
  - path: Lưu đường đi tạm thời
  - currentRoute: Tuyến đường hiện tại
  - routes: Danh sách các tuyến đường
  - routeCosts: Chi phí từng tuyến
  - surplusEnergy: Năng lượng dư thừa sau mỗi tuyến
- Hàm exploreWithSurplus(node, surplus):
  - Duyệt sâu các nút con chưa thăm nếu đủ năng lượng dư
- Hàm DFS(node):
  - Duyệt tiêu chuẩn:
  - Với mỗi nút con:
  - Nếu không đủ năng lượng → dùng surplus để thăm nút phụ
  - Nếu đủ năng lượng → tiếp tục DFS
  - Backtrack khi cần thiết
- Xử lý tuyến đường cuối cùng
- In kết quả:
  - Thứ tự duyệt DFS đầy đủ
  - Các tuyến đường với chi phí và năng lượng dư

3.2 MÃ GIẢ CỦA THUẬT TOÁN

```
function PDFS_offline():
    // Initialization
    path = [(root, 0.0)]
    currentRoute = [root.id]
    fullDFS = [root.id]
    routeEnergy = 0.0
    currentLength = 0.0
    carriedSurplus = 0.0

    function exploreWithSurplus(node, surplus):
        for (child, weight) in node.children:
            if not child.visited and (2 * weight <= surplus):
                child.visited = true
                currentRoute.append(child.id)
                fullDFS.append(child.id)
                routeEnergy += 2 * weight
                exploreWithSurplus(child, surplus - 2 * weight)
                currentRoute.append(node.id)
                fullDFS.append(node.id)

    function DFS(node):
        for (child, weight) in node.children:
            if child.visited:
                continue

            if routeEnergy + weight + (currentLength + weight) > B +
   carriedSurplus:
                surplus = B + carriedSurplus - (routeEnergy + currentLength)
                if surplus > 0:
                    exploreWithSurplus(node, surplus)

                // Finalize current route
                routeEnergy += currentLength
                surplusEnergy.append(B + carriedSurplus - routeEnergy)
                carriedSurplus = surplusEnergy[-1]
```

```
35
36                  // Add return path
37                  for i from len(path)-2 down to 0:
38                      currentRoute.append(path[i].id)
39
40                  routes.append(currentRoute)
41                  routeCosts.append(routeEnergy)
42
43                  // Reset for new route
44                  currentRoute = [root.id]
45                  routeEnergy = 0.0
46
47              // Continue traversal
48              child.visited = true
49              fullDFS.append(child.id)
50              currentLength += weight
51              routeEnergy += weight
52              currentRoute.append(child.id)
53              path.append((child, weight))
54
55              DFS(child)
56
57              // Backtrack
58              fullDFS.append(node.id)
59              path.pop()
60              currentLength -= weight
61              routeEnergy += weight
62              currentRoute.append(node.id)
63
64      // Execute algorithm
65      root.visited = true
66      DFS(root)
```

Chương 4

TRIỂN KHAI THUẬT TOÁN

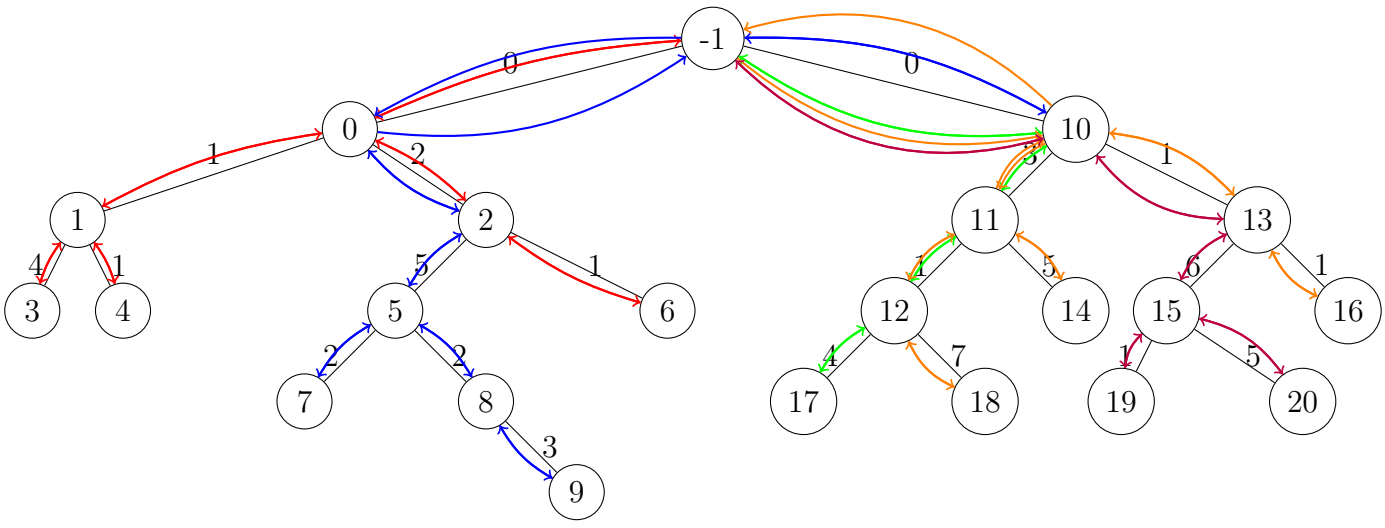
4.1 SOURCECODE

Link Code

4.2 KIỂM THỬ

4.2.1 Kiểm thử thủ công thuật toán

Lưu ý duyệt DFS này sẽ có thêm cả đỉnh quay lui và có thêm đỉnh được duyệt bởi lượng dư thừa.



Hình 4.1: Minh họa rừng 2 cây với các tuyến đường

Dầu tiên tuyến DFS ta có khi áp dụng thuật toán trên là:
`-1 0 1 3 1 4 1 0 2 6 2 5 7 5 8 9 8 5 2 0 -1 10 11 12 17 12 18 12 11 14 11 10 13 16 13 15 19 15 20 15 13 10 -1`
Thực hiện chia tuyến theo mục 2.2.2 ta có được là 5 tuyến như hình vẽ:

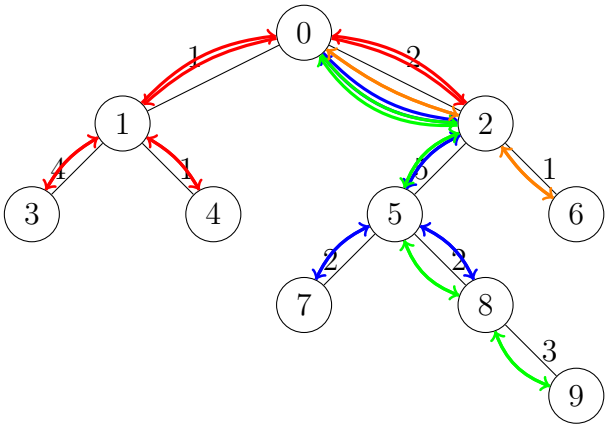
- **Route 1** (cost = 22, surplus = 4): `-1 0 1 3 1 4 1 0 2 6 2 0 -1`
- **Route 2** (cost = 30, surplus = 0): `-1 0 2 5 7 5 8 9 8 5 2 0 -1 10 -1`

- **Route 3** (cost = 16, surplus = 10): `-1 10 11 12 17 12 11 10 -1`
- **Route 4** (cost = 36, surplus = 0): `-1 10 11 12 18 12 11 14 11 10 13 16 13 10 -1`
- **Route 5** (cost = 26, surplus = 0): `-1 10 13 15 19 15 20 15 13 10 -1`

4.2.2 Nhận xét

Nếu như dùng theo cách truyền thống ta phải tách rừng này thành 2 cây tương ứng với bên phải và bên trái gốc ảo (-1). Lúc này ta sẽ có các tuyến đường như sau:

Cây bên phải



Route 1 (cost=20)
Route 2 (cost=24)
Route 3 (cost=26)
Route 4 (cost=8)

Hình 4.2: Cây bên phải các tuyến đường tối ưu

Tuyến DFS: `0 1 3 1 4 1 0 2 5 7 5 8 9 8 5 2 6 2 0`
Các tuyến được chia:

- **Route 1** (cost = 20): `0 1 3 1 4 1 0 2 0`
- **Route 2** (cost = 24): `0 2 5 7 5 8 5 2 0`
- **Route 3** (cost = 26): `0 2 5 8 9 8 5 2 0`
- **Route 4** (cost = 8): `0 2 6 2 0`

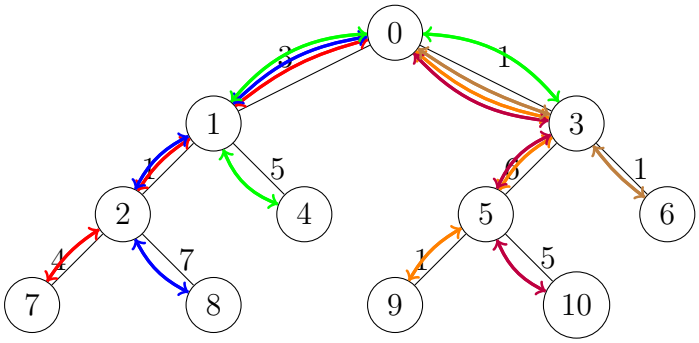
Cây bên trái

Tuyến DFS: `0 1 2 7 2 8 2 1 4 1 0 3 5 9 5 10 5 3 6 3 0`
Các tuyến được chia:

- **Route 1** (cost = 16): `0 1 2 7 2 1 0`
- **Route 2** (cost = 22): `0 1 2 8 2 1 0`
- **Route 3** (cost = 18): `0 1 4 1 0 3 0`
- **Route 4** (cost = 26): `0 3 5 9 5 10 5 3 0`
- **Route 5** (cost = 4): `0 3 6 3 0`

4.2.3 Kiểm thử bằng thuật toán C++

Chạy thuật toán C++ từ file đính kèm ta có kết quả như sau:



Route 1 (cost=16)
Route 2 (cost=22)
Route 3 (cost=18)
Route 4 (cost=16)
Route 5 (cost=24)
Route 6 (cost=4)

Hình 4.3: Cây bên trái với các tuyến đường tối ưu

```
2 26
10
0 1 2
0 2 3
1 3 4
1 4 1
2 5 5
2 6 1
5 7 2
5 8 2
8 9 3
11
10 11 3
10 13 1
11 12 1
11 14 5
12 17 4
12 18 7
13 15 6
13 16 1
15 19 1
15 20 5
-1 0 1 3 1 4 1 0 2 6 2 5 7 5 8 9 8 5 2 0 -1 10 11 12 17 12 18 12 11 14 11 10 13 16 13 15 19 15 20 15 13 10 -1
Route 1 (cost = 22, surplus = 4): -1 0 1 3 1 4 1 0 2 6 2 0 -1
Route 2 (cost = 30, surplus = 0): -1 0 2 5 7 5 8 9 8 5 2 0 -1 10 -1
Route 3 (cost = 16, surplus = 10): -1 10 11 12 17 12 11 10 -1
Route 4 (cost = 36, surplus = 0): -1 10 11 12 18 12 11 14 11 10 13 16 13 10 -1
Route 5 (cost = 26, surplus = 0): -1 10 13 15 19 15 20 15 13 10 -1
```

Hình 4.4: Kết quả kiểm tra thuật toán bằng C++

Chương 5

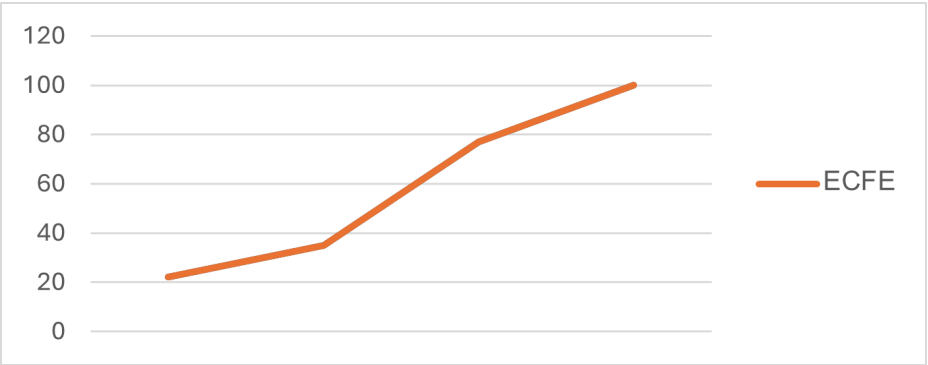
PHÂN TÍCH THUẬT TOÁN

5.1 PHÂN TÍCH

5.1.1 Độ phức tạp

Theo như thuật toán ta có độ phức tạp của thuật toán là $O(n)$. $O(n)$ thì độ phức tạp vẫn cao nhưng với sự phát triển công nghệ như ngày nay thì với $O(n)$ không phải là độ phức tạp quá lớn. Ta có bảng, biểu đồ về thời gian chạy như sau:

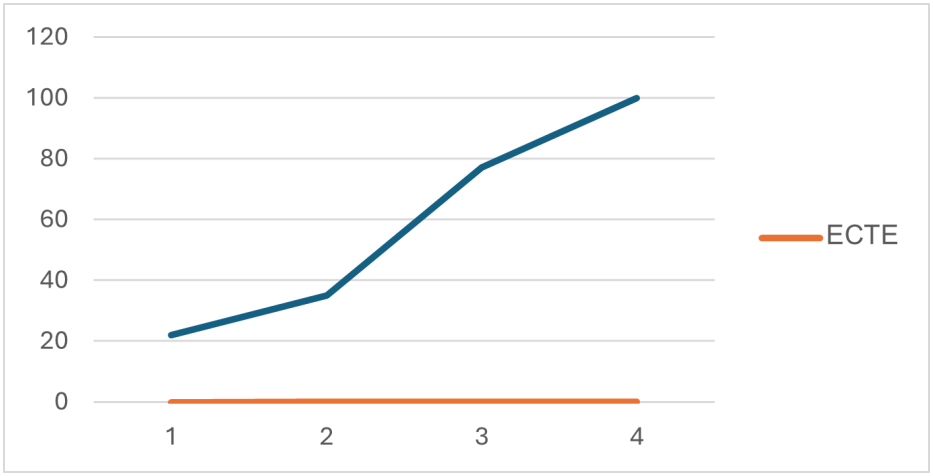
Số đỉnh	22	35	77	100
Thời gian (s)	0.02	0.040	0.07	0.11



Hình 5.1: Biểu đồ minh họa thời gian chạy code

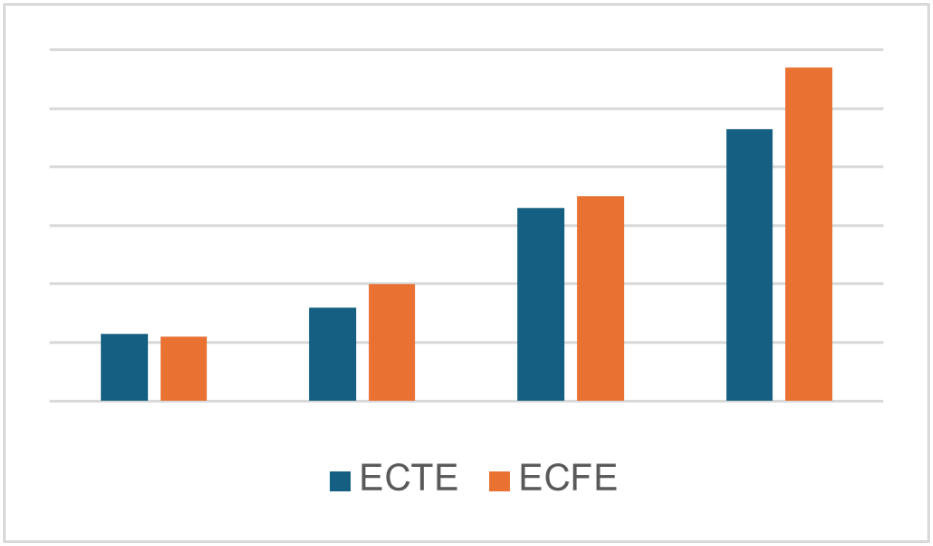
Nếu như ta dùng thuật toán truyền thống thì ta sẽ duyệt từng cây một tương ứng, ta có bảng thời gian khi chạy theo thuật toán truyền thống như sau:

Số đỉnh	22	35	77	100
Thời gian	0.02	0.03	0.07	0.09



Hình 5.2: Biểu đồ minh họa thời gian chạy code

Dựa vào hai biểu đồ trên ta có biểu đồ so sánh về thời gian như sau:



Hình 5.3: Biểu đồ minh họa thời gian chạy code

Nhận xét: Thời gian chạy code của hai thuật toán xét về thời gian thì không có sự chênh lệch quá lớn, nên nếu xét về thời gian thì khi dùng theo cách nào cũng mang tính tối ưu về thời gian duyệt. Bên cạnh đó thời gian chạy rất nhanh với 100 đỉnh thời gian chỉ với 0.11 giây

5.1.2 Số tuyến duyệt và chi phí

Tiêu chí	ECFE	ECTE
Số tuyến	5	9
Chi phí tổng	130	234
Chi phí dùng	130	164
Lượng dư	0	70

Bảng 5.1: So sánh testcase 1

Tiêu chí	ECFE	ECTE
Số tuyến	4	4
Chi phí tổng	344	344
Chi phí dùng	328	272
Lượng dư	16	72

Bảng 5.2: So sánh testcase 2

Tiêu chí	ECFE	ECTE
Số tuyến	5	5
Chi phí tổng	430	430
Chi phí dùng	396	352
Lượng dư	34	78

Bảng 5.3: So sánh testcase 3

Tiêu chí	ECFE	ECTE
Số tuyến	11	13
Chi phí tổng	1100	1300
Chi phí dùng	1058	1036
Lượng dư	42	264

Bảng 5.4: So sánh testcase 4

Dựa vào các bảng trên minh họa ta có thể thấy:
Chi phí tổng cho khám phá thì khi khám phá trên rừng n cây $\leq$ chi phí khám riêng từng cây.

Chi phí dùng cho việc khám phá trên rừng n cây có thể $\geq$ chi phí khám phá riêng cho từng cây.

Lượng dư thừa của việc khám phá trên rừng n cây luôn luôn bé hơn lượng dư khám phá trên từng cây vì thuật toán đề xuất này có tối ưu lượng dư thừa. Và đây cũng là lý do tại sao chi phí dùng trong rừng n cây lớn hơn. Bởi vì khi duyệt trên rừng trên n cây lượng dư thừa sẽ duyệt thêm những nhánh có thể dùng bởi lượng dư thừa. Sau đó lại duyệt thêm nhánh đó lần nữa cho tuyến khác
Số tuyến của thuật toán khám phá trên n cây $\leq$ số tuyến của thuật toán khám phá riêng từng cây

5.2 ĐÁNH GIÁ

Thuật toán được đề xuất trên đã thể hiện được bản chất của Piecemeal DFS, và đảm bảo tính đúng đắn của thuật toán và có nhiều ưu điểm bên cạnh cũng có những nhược điểm. Cụ thể hơn

- Về bản chất: thuật toán đề xuất vẫn thực duyệt từng phần(PDFS), đảm bảo theo công thức 2.1.
- Về tính đúng đắn: thuật toán đã thực hiện đúng theo yêu cầu ban đầu và kết quả thuật toán trùng khớp với kiểm thử thủ công, bên cạnh đó còn rất nhiều testcase khác cũng đảm bảo đúng
- Về ưu điểm:
  - Thuật toán duyệt rất nhanh, với 100 đỉnh nhưng chỉ có 0.11(s)
  - Thuật toán giúp giảm số tuyến duyệt và tối ưu lượng dư thừa
  - Thuật toán giúp tiết kiệm năng lượng khám phá nhờ vào cơ chế tối ưu lượng dư thừa

  - Thuật toán thích hợp ứng dụng trong thực tế khám phá
- Về nhược điểm:
  - Thuật toán với độ phức tạp thời gian vẫn lớn. Nên nếu đầu vào lớn thì sẽ thực hiện khá chậm
  - Lượng dư thừa chưa hoàn toàn được tối ưu trong các trường hợp sau:
    * Cây khám phá có trọng số lớn hơn so với lượng dư thừa qua từng tuyến
    * Lượng dư thừa được dùng không đúng lúc, nếu như thế sẽ dẫn đến chi phí sẽ tăng, vì sẽ thăm những nhánh nhiều lần
    * Không dùng được trong trường hợp giá trị B được dùng hết sau mỗi tuyến hoặc giá trị B quá lớn so với cây đó, lúc này số tuyến vẫn sẽ giữ nguyên

KẾT LUẬN

Thuật toán được đề xuất trên đã thực hiện đúng và đáp ứng được mục đích ban đầu mà ta đặt ra, đó là giải quyết được vấn đề duyệt trên một rừng trên n cây, Và cụ thể trong bài đã chứng minh lần lượt là 2 cây, 3 cây, 4 cây, 5 cây. Qua đó ta thấy cả về số tuyến lẫn chi phí đều được tối ưu hơn so với việc khi duyệt trên một cây. Hơn thế khi duyệt trên một rừng ta còn có tối ưu lượng dư thừa.

Song bên cạnh vẫn còn nhược điểm là độ phức tạp là tuyến tính vẫn còn quá cao so với mong đợi, và lượng dư thừa khi dùng không đúng lúc sẽ ảnh hưởng đến kết quả chi phí.

Ứng dụng của thuật toán này cực kì cao trong thực tế khám phá vì chúng thể hiện được tính tối ưu về chi phí và số tuyến, do đó tương lai thuật toán này mong đợi sẽ được ứng dụng trong thực tế