Đại Học Bách Khoa Hà Nội

Lớp Kĩ Sư Tài Năng - Công Nghệ Thông Tin K62

Tìm đường thoát hiểm

Ngày 8 tháng 5 năm 2019

Author

1 Tiêu chuẩn đề đánh giá trọng số con đường

Xét đoạn đường với chiều dài L, chiều ngang H, chỉ số an toàn T, và hiện tại có N người trên đoạn đường đó. Giả sử trong điều kiện bình thường, đoạn đường không có người, độ an toàn T = 1, thì với một người di chuyển vào con đường đó sẽ có vận tốc là V. Nhưng với ngoại cảnh cụ thể, thì vận tốc người đó sẽ nhỏ hơn, có thể coi như vận tốc thực tế của người đó tỉ lệ thuận với độ an toàn và tỉ lệ nghịch với mật độ người trên con đường đó. Ta có thể xét

$$v = \frac{V * f(T)}{g(D)}$$

$D = \frac{L*H}{X*Y}$ là mật độ người trên đoạn đường, g và f là hai hàm đồng biến trên khoảng giá trị tương ứng với D và T. Như vậy, thời gian để những người đó đi hết đoạn đường là:

$$t = \frac{L}{v} \Leftrightarrow t = \frac{L * g(D)}{V * f(T)}$$

Đặt trọng số $w = \frac{L*g(D)}{f(T)}$ thì $t = \frac{w}{V}$.
Như vậy, trong công thức trên, nếu V là vận tốc chuẩn của một người thì w giống như chuẩn độ dài của con đường trong ngoại cảnh cụ thể, và thời gian đi qua con đường sẽ được tính bởi hai đơn vị trên.
Ta sẽ giả sử tất cả mọi người trong tòa nhà đều có cùng vận tốc trung bình là v_{tb}, và khi đó thì để so sánh xem hai con đường nào tốt hơn, tức là ta phải so sánh thời gian thoát hiểm của hai con đường đó, đồng nghĩa với việc ta phải so sánh trọng số w của quãng đường.
Ở đây, ta sẽ sử dụng hàm

$$f(T) = T, g(D) = D + 1 \tag{1}$$

2 Ý tưởng thuật toán

Xét đoạn đường đi từ một Indicator u trong tòa nhà tới một exit node e, gồm các corridor theo thứ tự là $p =< p_1, p_2, ..., p_k >$. Với mỗi corridor p_i sẽ có tương ứng n_i người.

Trong số w của đoạn đường được tính: $w = \sum w_i$ với $w_i \frac{L_i * g(D_i)}{f(T_i)}$ là trọng số của quãng đường tương ứng tại thời điểm hiện tại. Với công thức như trên thì ta thấy trọng số của đường đi từ Indicator u sẽ phụ thuộc vào lượng người trên mỗi cạnh tại thời điểm tính, là t_0. Tại thời điểm t_1 những người ở cạnh p_1 đi đến được p_i, lúc đó lượng người trên cạnh p_i đã thay đổi. Ta có nhận xét rằng khi mà nhóm người A ở cạnh p_{i1} tới được được cạnh p_{i2} (i2 > i1) thì những người ở cạnh p_{i2} đã do chuyển sang cạnh khác, tức là họ không ảnh hưởng tới thời gian di chuyển của nhóm người A trên cạnh p_{i2} Như vậy, nếu ta muốn lấy trọng số w để đặc trưng cho thời gian thoát hiểm của một nhóm người trên một đoạn đường thì không thể tính bởi công thức như trên. Để có thể tính chi li, xét dãy $t =< t_1, t_2, ..., t_k >$, t_i là thời điểm mà số người đó bắt đầu đi vào quãng đường p_i, thì trọng số w sẽ được tính

$$w = \sum w_i \tag{2}$$

với w_i là trọng số của quãng đường p_i tại thời điểm t_i.
Vấn đề là với mỗi đường đi khác nhau sẽ có cách chia thời gian khác nhau, và để tính toán với từng đoạn đường như vậy là không thể về khía cạnh thời gian tính toán.
Với ý tưởng tương tự trên, ta sẽ chia đường đi làm 2 phần. Phần đầu tiên là quãng đường đi được trong t(s) tiếp theo, phần tiếp theo là đoạn còn lại. Giả sử trong t(s) tiếp theo thì những người ở Indicator u đi đến được đoạn p_i.
Đặt path1 $= < p_1, ..., p_{i-1}$, path2 $= < p_i, ..., p_k >$. Trọng số của đoạn đường p sẽ được tính:

$$w = w1 + w2 \tag{3}$$

với

- $w1 = \sum w_j, j < i$, các đoạn w_j được tính với số người trên đoạn là n_1 người, hay là số người đang ở p_1 tại thời điểm hiện tại.
- $w2 = \sum w_j, i <= j <= k$, các đoạn w_j được tính với số người tới được p_i trong t giây tiếp theo.

3 Nội dung thuật toán

Thuật toán nhận đầu vào là một đồ thì gồm đỉnh và các cạnh có thông tin đi kềm, thực hiện nhiệm vụ là chỉ đường đi cho tất cả các đỉnh trong đồ thị. Dưới đây là các đặc tính của lớp Node và Edge.
Node:

- *nextNode*: đỉnh được lựa chọn là hướng đi đến.
- *adjacences*: Danh sách struct gồm có:
  - *node*: đỉnh kề
  - *edge*: cạnh kề
  - *passingWeight*: trọng số của path mà đi qua đỉnh kề
  - *reaching*: đỉnh tới được sau t(s) mà đi qua đỉnh kề
- *weight*: trọng số của đường đi tới root.
- *nComingPeople*: số người sẽ đến sau t(s).
- *tReachedNode*: đỉnh sẽ tới được sau t(s).
- *tComingNodes*: danh sách đỉnh tới được sau t(s).
- *label*: nhãn của đỉnh.

Edge:

- *from*: đỉnh đến của cạnh.
- *to*: đỉnh đi.
- *Length, Width, Trustness*: các thông số.
- *nPeople*: số người trên cạnh.
- *density*: mật độ.
- *weight*: trọng số của cạnh.

Algorithm 1 Algorithm caption

Input: Đồ thị có trọng số ứng với các Indicator và Node
Output: Đường đi ngắn nhất từ root đến mọi đỉnh trong đồ thị
heap: cấu trúc heap min với phần tử là các đỉnh và so sánh dựa trên weight.

```
1:  procedure MAIN ALGORITHM
2:      heap.push(root)
3:      root.label ← true
4:      while heap.size > 0 do
5:          u ← heap.pop()
6:          u.label = true
7:          s ← u.tReachedNode
8:          s.nComingPeople ← s.nComingPeople + E[u,u.next].nPeople
9:          s.tComingNodes ← u
10:         UpdateComingNode(s)
11:         for v in u.adjacences and v.node.label = true do
12:             UpdateComingPeople(v.edge)
13:         for v in u.adjacences and v.node.label = false do
14:             s = v.FindCrossNode(v.node, v.edge, t)
15:             s.nComingPeople ← s.nComingPeople + E[v.node,u].nPeople
16:             w1 ← CalculateWeight(u, s, E[v.node,u].nPeople)
17:             w2 ← CalculateWeight(s, root, s.nComingPeople)
18:             newW ← E[v.node,u].weight + w1 + w2
19:             ad in v.node.adjacences which ad.node = u
20:             ad.edgeWeight ← newW
21:             ad.reachedNode ← s
22:             if newW < v.node.weight then
23:                 v.node.weight ← newW
24:                 v.node.next ← u
25:                 v.node.tReachedNode ← s
26:                 heap.push(v.node)
27:             end if
28:             s.nComingPeople ← s.nComingPeople - E[v.node,u].nPeople
        end for
    end while
```

Algorithm 2 FindCrossNode

Input: *Node* v, *Edge* e, *double* t.
Output: Trả lại đỉnh xa nhất trên đường đi ngắn nhất của u mà v đi tới được trong thời gian t.

```
procedure My Procedure
    sumWeight ← v_tb * t
    nPeople ← E[v,u].nPeople
    come ← v, to ← u
    while sumWeight > 0 do
        sumWeight ← sumWeight - GetWeight(E[come,to], nPeople)
        come ← to
        to ← to.next
        if come = root then break
    end while
return come
```

Algorithm 3 CalculateWeight

Input: *Node* u, *Node* s, int nPeople
Output: Trả lại trọng số của quãng đường đi từ u tới s với nPeople người đi.

```
procedure My Procedure
    weight ← 0
    spanW1 ← u.spanW1 - s.spanW1
    spanW2 ← u.spanW2 - s.spanW2
    weight ← spanW1 + nPeople*spanW2
return weight
```

Algorithm 4 UpdateComingNode

Input: *Node* s
Output: Cập nhật lại trọng số của các Node mà sẽ tới được s trong thời gian t những vẫn chưa được gán nhãn.

```
procedure My Procedure
    for u in s.tComingNodes and u.label = false do
        v in u.adjacences which v.tReachedNode = s
        w1 ← CalculateWeight(v.node, s, E[u,v].nPeople)
        w2 ← CalculateWeight(s, root, s.nComingPeople)
        u.adjacentEdgeWeights[v] ← E[u,v.node].weight + w1 + w2
        GetNextNode(u)
        heap.push(u)
    end for
procedure GetNextNode(u)                          ▷ Đặt lại đỉnh kề tốt nhất cho u
    for v in u.adjacences and v.node.label = true do
        if v.edgeWeight < weight then
            u.weight = v.edgeWeight
            u.next = v.node
            u.tReachedNode = v.reachedNode

        end if
```