**GIẢI BÀI TOÁN 8-PUZZLE TRONG TRƯỜNG HỢP BẢNG BỊ CHE**

**Nguyễn Thanh Hoàng, Nguyễn Thành Công**

*Trường đại học Sư phạm Kỹ thuật TP.HCM*

**TÓM TẮT**

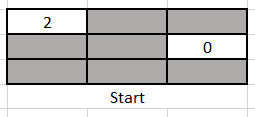
Bài toán 8-puzzle bị che là một phiên bản cải tiến của 8-puzzle truyền thống – một trò chơi quen thuộc trong trí tuệ nhân tạo. Trong quá trình học tập giải thuật A\* nhằm giải quyết 8-puzzle truyền thống, chúng em đã quyết định tìm ra cách giải 8puzzle bị che bằng phương pháp lật ô chữ kết hợp với giải thuật A\*. Để giải quyết vấn đề trên chúng em đã nhập trạng thái ban đầu và trạng thái đích của puzzle và thực hiện phương pháp lật ô chữ và giải thuật A\* để giải, từ đó thống kê được kết quả (Số bước đi, số trường hợp tìm được lời giải). Từ việc thống kê và tính kết quả trung bình ta sẽ biết được các trường hợp mà trò chơi có thể giải được và thời gian để thực hiện lời giải đó. Việc xây dựng một phương pháp giải 8-puzzle bị che nhằm giúp việc giải quyết bài toán trở nên dễ dàng và ứng dụng thuật toán vào các trò chơi tự động trở nên dễ dàng hơn.Xây dựng bài toán ứng dụng cho các trò chơi trí tuệ, áp dụng trí tuệ nhân tạo vào nhu cầu giải trí của con người

-Từ khóa : 8-puzzle bị che; lật ô chữ; giải thuật A\*; Trí tuệ nhân tạo;trạng thái bắt đầu ; trò chơi

**A. Bài toán 8-puzzle bị che**

**1.GIẢI THÍCH BÀI TOÁN**

Trong bài toán 8-Puzzle bị che ta có trạng thái ban đầu và trạng thái đích là các số từ 0 đến 8 nằm trong 1 bảng vuông 3x3 (mỗi số nằm một ô và trạng thái ban đầu phải khác trạng thái đích) nhưng so với phiên bản truyền thống thì ta chỉ có thể nhìn thấy được nhiều nhất 2 ô giá trị ( ô số 0 và một ô trống ngẫu nhiên để có thể nhìn thấy giá trị của ô đó). Để giải quyết bài toán, ta phải di chuyển ô số 0 (số 0 là ô trống) trong trạng thái ban đầu để tìm được các ô còn ẩn và sau đó giải 8-puzzle để tiến tới trạng thái đích. Hàm chi phí sử dụng trong các thuật toán để giải quyết bài toán này là số ô nằm sai vị trí trong trạng thái ban đầu so với trạng thái đích.

*Hình 1. Mô tả trạng thái đầu và trạng thái kết thúc 8-puzzle bị che*

**2.CÁCH GIẢI QUYẾT BÀI TOÁN**

Đầu tiên, ta truyền vào một trạng thái ban đầu. Sau đó, ta xác định ô trống và ô được nhìn thấy. Tiếp theo, ta sử dụng các bước đi hợp lệ của ô trống đó. Đầu tiên, di chuyển ô trống đến ô được nhìn thấy. Sau đó, dùng phương pháp lật ô chữ để tìm các số còn ẩn. Cuối cùng, áp dụng các thuật toán A\* để giải quyết bài toán.

**B.THUẬT TOÁN**

**1.Phương pháp lật ô chữ**

**1.1 Định nghĩa**

Phương pháp lật ô chữ là phương pháp được dùng để tìm các ô còn ẩn, bằng cách đưa số 0 về ô trống và thực hiện các thao tác di chuyển xoay vòng để tìm được các ô còn ẩn

**1.2 Đưa ô trống về vị trí nhìn thấy**

**Function** move0(Start)

k = location of open box;

hang = (k - 1) / 3;

cot = (k - 1) % 3;

**While** (index[0] != k)

{

**while** (hang > hang0)

{

swap(a[hang0+1][cot0], a[hang0][cot0])

swap(index[a[hang0+1][cot0]],index[a[hang0][cot0]])

hang0++

}

**While** (hang < hang0)

{

swap(a[hang0 - 1][cot0], a[hang0][cot0])

swap(index[a[hang0 - 1][cot0]], index[a[hang0][cot0]])

hang0--

}

**While** (cot > cot0)

{

swap(a[hang0][cot0 + 1], a[hang0][cot0])

swap(index[a[hang0][cot0+1]],index[a[hang0][cot0]])

cot0++

}

**while** (cot < cot0)

{

swap(a[hang0][cot0 - 1], a[hang0][cot0]);

swap(index[a[hang0][cot0-1]],index[a[hang0][cot0]]);

cot0--;

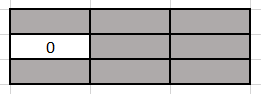
}

**return** start

Hình 2. Thuật toán đưa ô trống về ô được nhìn thấy

Ở đây em sẽ dùng phương pháp so sánh vị trí của ô trống và ô được chọn qua tọa độ. Đầu tiên thuật toán sẽ so sánh vị trí hàng của ô trống và ô được chọn và lựa chọn cách đi lên hoặc đi xuống cho phù hợp. Hành động kết thúc khi số hàng của ô trống bằng số hàng của ô được chọn. Tương tự như với khi di chuyển theo chiều ngang.Đây là bước khởi đầu để giải các ô bị che khác.

**1.3 Lật ô chữ trong trường hợp nhìn thấy ở viền ngoài**

**

*Hình 3*. Vị trí ô nhìn thấy ở viền ngoài

**function** zeroinedge(i, j, b[3][3], k)

hang, cot, s = 7, x, t, p, l

hang = (k - 1) / 3

cot = (k - 1) % 3

**while** (s != 0)

{

x = 1, t = 1, p = 1, l = 1, k = 0;

**while** (x != 0 && t != 0 && l != 0 && p != 0)

{

if (x == 1)

{

**while** (i <= 1)

{

swap(b[i + 1][j], b[i][j])

**if** (isSolve)

**return** solve

i++

}

x--

}

if (p == 1)

{

**while** (j <= 1)

{

swap(b[i][j + 1], b[i][j])

**if** (isSolve)

**return** solve

j++

}

p--

}

if (l == 1)

{

**while** (i >= 1)

{

swap(b[i - 1][j], b[i][j])

**if** (isSolve)

**return** solve

i--

}

l--

}

if (t == 1)

{

**while** (j >= 1)

{

swap(b[i][j - 1], b[i][j])

**if** (isSolve)

**return** solve

j--

}

t--

}

}

s--

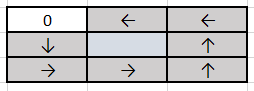
}

**return** solve

Hình 4. Thuật toán lật ô chữ cho trường hợp ô trống nằm viền ngoài

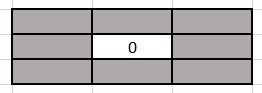
Ở trường hợp này ta di chuyển ô trống đi xung quanh viền đến khi nào tìm được hết các số bị che thì dừng lại, số còn lại sẽ được tìm ra bằng cách loại trừ. Như vậy mỗi khi di chuyển xong một vòng và bắt đầu vòng mới thì một ô bị che sẽ được hiển thị. Ở đây bọn em cho ô trống di chuyển vòng theo viền của puzzle đến khi các ô ở viền được lật hết thì dừng. Cuối cùng chỉ còn ô che ở giữa và ta dùng loại suy để tìm ra nó.

Ô trống ở đây được di chuyển bằng các so sánh vị trí của nó theo chiều ngang , chiều dọc. Mỗi lần di chuyển một đơn vị. Khi di chuyển sẽ có di chuyển theo 4 hướng đi: Trái, phải, xuống, lên.

**

Hình 5. Hướng di chuyển khi ô nằm ở rìa

**1.4 Khi ô được chọn nằm ở trung tâm**

**

*Hình 6.* Ô được chọn nằm ở trung tâm

swap(b[i][j + 1], b[i][j])

j++

swap(b[i + 1][j], b[i][j])

i++

swap(b[i - 1][j], b[i][j])

i--

swap(b[i - 1][j], b[i][j]);

i--

swap(b[i][j - 1], b[i][j]);

j--

swap(b[i + 1][j], b[i][j]);

i++

swap(b[i][j + 1], b[i][j]);

j++

swap(b[i - 1][j], b[i][j]);

i--

swap(b[i][j + 1], b[i][j]);

j++

swap(b[i + 1][j], b[i][j]);

i++

swap(b[i][j - 1], b[i][j]);

j--

swap(b[i - 1][j], b[i][j]);

i --

**return** b

**Function** zeroinmidle( a, flag, index)

swap(b[i + 1][j], b[i][j])

swap(b[i + 1][j], b[i][j])

swap(b[i - 1][j], b[i][j])

swap(b[i - 1][j], b[i][j])

swap(b[i][j - 1], b[i][j])

swap(b[i][j - 1], b[i][j])

swap(b[i][j + 1], b[i][j])

swap(b[i][j + 1], b[i][j])

swap(b[i + 1][j], b[i][j])

i++

swap(b[i][j + 1], b[i][j])

j++

swap(b[i - 1][j], b[i][j])

i--

swap(b[i][j - 1], b[i][j])

j--

swap(b[i + 1][j], b[i][j])

i++

swap(b[i][j - 1], b[i][j])

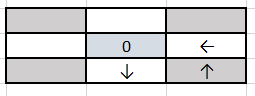
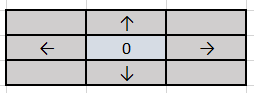
j--

swap(b[i - 1][j], b[i][j])

i--

*Hình 7. Thuật toán lật ô chữ cho trường hợp ô trống nằm ở trung tâm*

Ở trường hợp này ta có 2 phần di chuyển. Phần đầu tiên ta sẽ tìm bốn ô ở trục ngang và dọc của ô trung tâm bằng cách đổi chỗ của nó với ô trống, khi đó ta tìm được 4 ô bị che. Tiếp đến, ô trống sẽ đi theo một vòng tròn nhỏ (từ 4 tới 6 ô) để tìm các ô ở góc. Bắt đầu từ trung tâm, ô trống sẽ được di chuyển theo các hướng chỉ định để tìm ra các ô bị che. Kết thúc thuật toán ta sẽ được một puzzle hoàn toàn không bị che.

**

Hình 8. Hướng di chuyển khi ô nằm ở rìa

**2.Giải thuật tìm kiếm A\***

**2.1 Giải thuật A\***

**function** A\*(Start,goal)

Ftier = FIFO\_initial()

Explorer= FIFO\_initial()

FIFO\_add(Ftier, Start)

Ftier->nodeCount++

**while**( Ftier->nodeCount >0 && isnotgoal)

{

Node\* Curr = FindMinValue(Ftier)

FIFO\_add(Explorer, createNode(Curr))

MoveaStar(Curr)

CurrChild = Curr -> nextNode

**while** (CurrChild != NULL)

{

**if**(Goal)

{

**return** CurrChild

}

}

**if**(!checkExist(Ftier,currChild) && !checkExist(Explorer, currChild))

{

Node\* Temp = createNode(CurrChild)

Temp->nextNode = NULL

FIFO\_add(Ftier, Temp)

}

CurrChild = CurrChild->nextNode

}

**Return** failure

Hình 9. Giải thuật A\*

Hình 8 Là giải thuật A \* được dùng để giải bài toán 8-puzzle sau khi đã tìm được các giá trị của từng ô

Đầu tiên ta đặt trạng thái ban đầu là trạng thái của 8-puzzle sau khi đã tìm ra giá trị của từng ô trên puzzle, và trạng thái đích là trạng thái mà ta đã đưa ra ở đầu bài toán.

Ta sẽ thực hiện quá trình di chuyển ô trống cho đến khi tìm được trạng thái đích theo quy tắc di chuyển:

Từ trạng thái ban đầu, ta sẽ cho sinh ra các nút con CurrChild tượng trưng cho các bước mà ô trống có thể di chuyển được, các Currchild đều có chi di phí di chuyển, dựa vào các các chi phí vừa tìm được ta sẽ chọn ra nút con có chi phí nhỏ nhất và không lặp lại so với trạng thái đã tồn tại khi di chuyển để lấy làm bước đi tiếp theo. Việc di chuyển sẽ được diễn ra như vậy cho đến khi không còn nút con nào và trạng trái di chuyển đạt đến trạng thái đích.

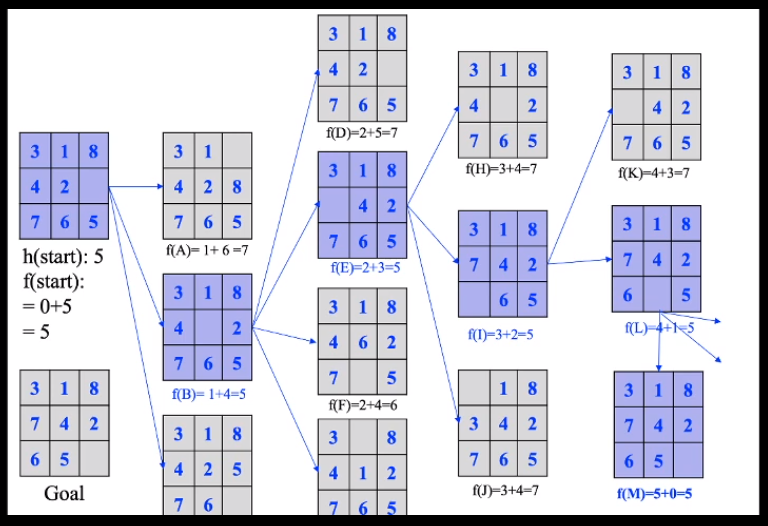
**2.2 Hàm tính chi phí**

**Pathcost = Depth + CountC**

**Depth** : Độ sâu của puzzle, khi di chuyển thì độ sâu sẽ tăng một đơn vị

**CountC** : Số ô khác nhau so với trạng thái đích

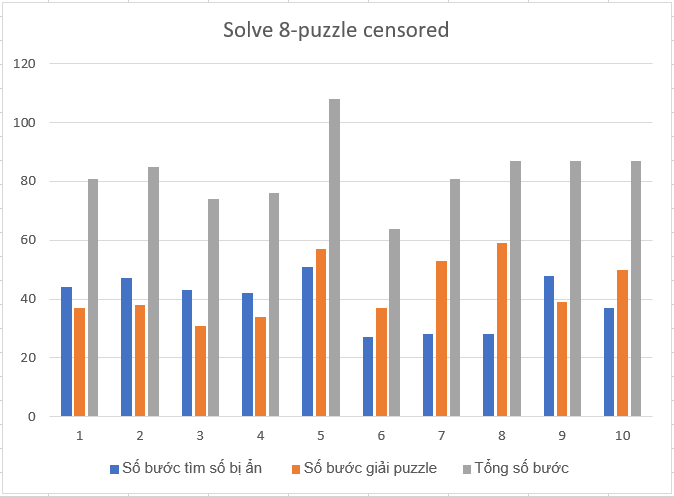
Hình 10. Chi phí di chuyển

**

Hình 11. Di chuyển dựa theo di chí

**3. KẾT QUẢ**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Các trường hợp | Vị trí ô để nhìn thấy | Số bước giải puzzle bị che | Số bước giải puzze | Tổng số bước giải |
| 1 | 1 | 44 | 37 | 81 |
| 2 | 2 | 47 | 38 | 85 |
| 3 | 7 | 43 | 31 | 74 |
| 4 | 4 | 42 | 34 | 76 |
| 5 | 9 | 51 | 57 | 108 |
| 6 | 5 | 27 | 37 | 64 |
| 7 | 5 | 28 | 53 | 81 |
| 8 | 5 | 28 | 59 | 87 |
| 9 | 3 | 48 | 39 | 87 |
| 10 | 6 | 37 | 50 | 87 |
|  | Trung bình cộng | 39.5 | 43.5 | 83 |



*Hình 12.* Biểu đồ kết quả giải 8-puzzle bị che

Hình 12 thể hiện kết quả khi thực thi phương pháp lật ô chữ và giải thuật A\*. Ở đây số bước đi của phương pháp lật ô chữ thấp số bước đi của giải thuật A\*. Trong đó, nếu ta chọn ô trung tâm làm ô nhìn thấy được thì số bước giải sẽ thấp hơn nhiều so với chọn nhìn thấy ở các ô khác.

**4. KẾT LUẬN**

Trong đồ án này chúng em đã dùng phương pháp lật ô và A\* để giải 8-puzzle bị che. Giải thuật đạt độ chính xác tương đối cao với các trạng thái giải được. Kết quả cho thấy đều tìm được các ô bị ẩn và giải được 8-puzzle. Tuy vậy đồ án này vẫn còn một số hạn chế cần khắc phụ ở khâu tìm các ô bị ẩn vì chi phí thực hiện lớn.

**TÀI LIỆU THAM KHẢO**

[1] A\* cho bài toán 8-Puzzle – Tác giả : Sach Le Thanh <https://youtu.be/P_20RYNowyw> (20/06/2020)