

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



BÀI TẬP LỚN NHẬP MÔN TRÍ TUỆ NHÂN TẠO

Đề tài:

**THUẬT TOÁN A*
ỨNG DỤNG TRONG BÀI TOÁN GHÉP TRANH**

Sinh viên thực hiện : **Lê Đình Cường**
(Các thành viên khác) :
Mã số sinh viên : **20080370**
Nhóm: 3

HÀ NỘI 04-2013

MỤC LỤC

Lời nói đầu	3
I- BÀI TOÁN GHÉP TRANH.....	4
II- THUẬT TOÁN A*.....	5
1- Giới thiệu thuật toán.....	5
2- Mô tả thuật toán	7
3- Cài đặt thuật toán.....	7
III- CÀI ĐẶT BÀI TOÁN.....	9
1. Trạng thái xuất phát.....	9
2. Cài đặt A*	9
3. Hàm ước lượng heuristic	12
3.1 Các hàm ước lượng heuristic.....	12
3.2 Ví dụ so sánh 3 hàm heuristic.....	14
III- KẾT QUẢ.....	15
1- Giao diện.....	15
2- So sánh.....	16
3- Nhận xét.....	21
IV- KẾT LUẬN.....	22
Tài liệu tham khảo	23
PHIẾU GIAO NHIỆM VỤ BÀI TẬP LỚN	Error! Bookmark not defined.

Lời nói đầu

Đây là tài liệu dùng để biểu diễn cơ bản thiết kế và giải quyết bài toán “Trò chơi ghép tranh” sử dụng thuật toán A^* do tôi thiết kế và lập trình. Tài liệu này giúp ta có cái nhìn toàn vẹn về các chức năng của phần mềm cũng như ứng dụng thuật toán A^* để giải quyết bài toán này. Do thời gian có hạn nên đề án không thể tối ưu được toàn bộ không gian trạng thái bài toán. Tuy nhiên, nhóm sẽ nghiên cứu hoàn thiện trong thời gian sớm nhất.

Nhóm thực hiện đề tài nhằm mục đích xây dựng một hệ thống giải quyết một bài toán thực tế dựa trên chiến lược tìm kiếm heuristic và xây dựng một trò chơi ứng dụng giải trí. Trong quá trình thực hiện đề tài không tránh khỏi những sai sót, nhóm tôi mong sẽ nhận được sự góp ý và đánh giá của thầy.

Xin chân thành cảm ơn !

I- BÀI TOÁN GHÉP TRANH

Game ghép tranh(N-Puzzle) là một trò chơi khá hay và trí tuệ, nó được biết đến với nhiều phiên bản và tên gọi khác nhau như: 8-puzzle, 15-puzzle, Gem puzzle, Boss puzzle. Bài toán N-puzzle là vấn đề cổ điển cho mô hình thuật toán liên quan đến trí tuệ nhân tạo. Bài toán đặt ra là phải tìm đường đi từ trạng thái hiện tại tới trạng thái đích. Và cho tới nay vẫn chưa có thuật toán tối ưu để giải bài toán này.

Phần mềm N-Puzzle là một chương trình xây dựng trò chơi và giải quyết bài toán này. Phần mềm được viết trên nền Java, sử dụng giao diện đồ họa để mô phỏng trò chơi và thuật toán A* để tìm đường đi. Người dùng có thể sử dụng chuột/bàn phím chơi với các kích thước khác nhau và với hình ảnh khác nhau hoặc có thể sử dụng chức năng tìm lời giải nhờ thuật toán A*.

Yêu cầu xây dựng bảng ô vuông n hàng, n cột. Bảng gồm 1 ô trống và n^2-1 ô chứa các số trong phạm vi $[1, n^2-1]$. Xuất phát từ một cách xếp bất kỳ, di chuyển ô trống lên trên, xuống dưới, sang phải, sang trái để đưa các ô về trạng thái đích. Sử dụng chuột hay phím chức năng để di chuyển ô trống. Chương trình có chức năng tự động chơi ở bất kỳ trạng thái nào đó. Mỗi trạng thái của bảng số là một hoán vị của n^2 phần tử. Ở đây ta có thể mở rộng bằng việc thêm hình ảnh vào game hoặc gắn số vào hình ảnh để gợi ý cho người chơi. Ở trạng thái ban đầu, các ô được sắp xếp ngẫu nhiên, và nhiệm vụ của người chơi là tìm được cách đưa chúng về trạng thái đích(ô đầu trống, các ô khác theo thứ tự tăng dần từ trái qua phải, từ trên xuống dưới). Để đơn giản trong cách tiếp cận bài toán, ta giả định chỉ các ô trống di chuyển trong bảng là di chuyển đến các vị trí khác nhau. Như vậy tại một trạng thái bất kỳ có tối đa 4 cách di chuyển đến trạng thái khác(trái, phải, lên, xuống).

6		1
7	2	5
3	8	4

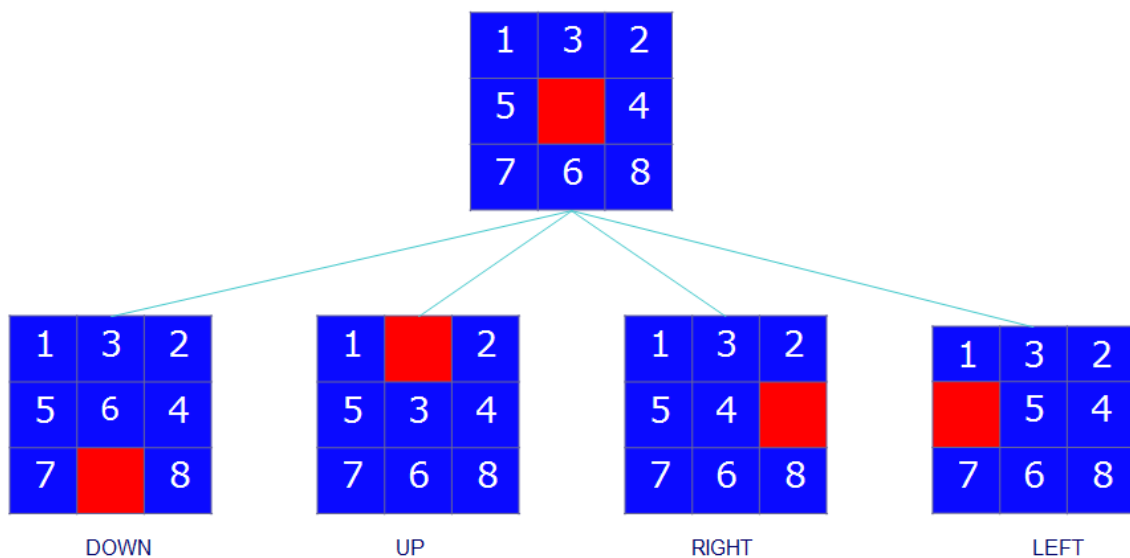
Init State

	1	2
3	4	5
6	7	8

Goal State

1.1.1: Trạng thái bắt đầu và đích

Bước di chuyển của ô trống:



1.1.2: Bước di chuyển của ô trống

II- THUẬT TOÁN A*

1- Giới thiệu thuật toán

Thuật toán A* được mô tả lần đầu tiên năm 1986 bởi Peter Hart, Nils Nilsson và Bertram Raphael. Trong báo cáo của họ, thuật toán được gọi là thuật

toán A, khi sử dụng thuật toán này với một hàm đánh giá heuristic thích hợp sẽ thu được hoạt động tối ưu, do đó mà có tên là A^* .

Trong khoa học máy tính, A^* là một thuật toán tìm kiếm trong đồ thị. Thuật toán này tìm một đường đi từ nút khởi đầu tới một nút đích cho trước (hoặc tới một nút thỏa mãn điều kiện đích). Thuật toán này sử dụng một đánh giá heuristic để xếp loại từng nút theo ước lượng về tuyến đường tốt nhất đi qua nút đó. Thuật toán này duyệt các nút theo thứ tự của đánh giá heuristic này. Do đó, thuật toán A^* là một ví dụ của tìm kiếm theo lựa chọn tốt nhất (best-first search).

Xét bài toán tìm đường – bài toán mà A^* thường được dùng để giải. A^* xây dựng tăng dần tất cả các tuyến đường từ điểm xuất phát cho tới khi nó tìm thấy một đường đi chạm tới đích. Tuy nhiên, cũng như tất cả các thuật toán tìm kiếm có thông tin nó chỉ xây dựng các tuyến đường có vẻ dẫn về đích.

Để biết những tuyến đường nào có khả năng sẽ dẫn tới đích, A^* sử dụng một hàm đánh giá heuristic về khoảng cách từ điểm bất kỳ cho tới đích. Trong trường hợp tìm đường đi, đánh giá này có thể là khoảng cách đường chim bay - một đánh giá xấp xỉ thường dùng cho khoảng cách của đường giao thông.

Điểm khác biệt của A^* đối với tìm kiếm theo lựa chọn tốt nhất là nó còn tính đến khoảng cách đã đi qua. Điều đó làm cho A^* đầy đủ và tối ưu, nghĩa là A^* sẽ luôn tìm thấy đường đi ngắn nhất nếu tồn tại một đường đi như thế. A^* không đảm bảo sẽ chạy nhanh hơn các thuật toán tìm kiếm đơn giản hơn. Trong một môi trường dạng mê cung, cách duy nhất để đến đích có thể là trước hết phải đi về phía xa đích và cuối cùng mới quay trở lại. Trong trường hợp đó, việc thử các nút theo thứ tự “gần đích hơn thì được thử trước” có thể gây tốn thời gian.

2- Mô tả thuật toán

Giả sử n là một trạng thái đạt tới (có đường đi từ trạng thái ban đầu n_0 tới n). Ta xác định hàm đánh giá: $f(n) = g(n) + h(n)$

- $g(n)$ là chi phí từ nút gốc n_0 tới nút hiện tại n
- $h(n)$ chi phí ước lượng từ nút hiện tại n tới đích
- $f(n)$ chi phí tổng thể ước lượng của đường đi qua nút hiện tại n đến đích

Một ước lượng heuristic $h(n)$ được xem là chấp nhận được nếu với mọi nút n : $0 \leq h(n) \leq h^*(n)$

Trong đó $h^*(n)$ là chi phí thật (thực tế) để đi từ nút n đến đích.

3- Cài đặt thuật toán

OPEN(FRIDGE): tập chứa các trạng thái đã được sinh ra nhưng chưa được xét đến. OPEN là một hàng đợi ưu tiên mà trong đó phần tử có độ ưu tiên cao nhất là phần tử tốt nhất.

CLOSE: tập chứa các trạng thái đã được xét đến. Chúng ta cần lưu trữ những trạng thái này trong bộ nhớ để phòng trường hợp khi có một trạng thái mới được tạo ra lại trùng với một trạng thái mà ta đã xét đến trước đó.

Khi xét đến một trạng thái n_i trong OPEN bên cạnh việc lưu trữ 3 giá trị cơ bản g , h , f để phản ánh độ tốt của trạng thái đó, A^* còn lưu trữ thêm hai thông số sau:

- Trạng thái cha của trạng thái n_i (ký hiệu $\text{Cha}(n_i)$): cho biết trạng thái dẫn đến trạng thái n_i .
- Danh sách các trạng thái tiếp theo của n_i : danh sách này lưu trữ các trạng thái kế tiếp n_k của n_i sao cho chi phí đến n_k thông qua n_i từ trạng thái ban đầu là thấp nhất. Thực chất danh sách này có thể được tính từ thuộc tính Cha của các trạng thái đã được lưu trữ. Tuy nhiên việc tính toán này có thể mất nhiều thời gian (khi tập OPEN, CLOSE được mở rộng) nên người ta thường lưu trữ ra một danh sách riêng.

Thuật toán A*:

function Astar(n_0 , n_{goal})

1. Đặt OPEN chỉ chứa n_0 . Đặt $g(n_0) = h(n_0) = f(n_0) = 0$. Đặt CLOSE là tập rỗng
2. Lặp lại các bước sau cho đến khi gặp điều kiện dừng
 - 2.a Nếu OPEN rỗng: bài toán vô nghiệm, thoát.
 - 2.b Ngược lại, chọn n_i trong OPEN sao cho $f(n_i)$ sao cho $f(n_i)$ min
 - 2.b.1 Lấy n_i ra khỏi OPEN và đưa n_i vào CLOSE
 - 2.b.2 Nếu n_i chính là đích n_{goal} thì thoát và thông báo lời giải là n_i
 - 2.b.3 Nếu n_i không phải là đích. Tạo danh sách tất cả các trạng thái kế tiếp của n_i . Gọi một trạng thái này là n_k . Với mỗi n_k , làm các bước sau:
 - 2.b.3.1 Tính $g(n_k) = g(n_i) + \text{cost}(n_i, n_k)$; $h(n_k)$;
$$f(n_k) = g(n_k) + h(n_k).$$
 - 2.b.3.2 Đặt $\text{Cha}(n_k) = n_i$
 - 2.b.3.3 Nếu n_k chưa xuất hiện trong OPEN và CLOSE thì thêm n_k vào OPEN

III- CÀI ĐẶT BÀI TOÁN

1. Trạng thái xuất phát

Rất dễ thấy mỗi trạng thái của bảng số là một hoán vị của n^2 phần tử (với n là kích thước cạnh), như vậy không gian trạng thái của nó là $n^2!$, 8-puzzle là $9! = 362880$ ($n = 3$) và 15-puzzle là $16! = 20922789888000$ ($n = 4$),.... Khi n tăng lên 1 đơn vị thì không gian trạng thái sẽ tăng lên rất nhanh, điều này khiến cho việc giải quyết với các phiên bản $n > 3$ ít được áp dụng.

Để tạo ra một trạng thái ban đầu của trò chơi ta dùng mảng 1 chiều và sinh ngẫu nhiên một mảng trạng thái là hoán vị của n^2 phần tử trong đoạn $(0; n^2-1)$. Với việc sinh ngẫu nhiên thì sẽ tạo ra những trạng thái không hợp lệ (trạng thái không dẫn tới trạng thái đích của bài toán), thì ta cần phải kiểm tra xem trạng thái có dẫn tới đích được hay không trước khi khởi tạo giao diện. Ngoài ra ta có thể trộn ngẫu nhiên trạng thái đích đến một trạng thái bất kì.

1	2	3	7
4	5	11	10
9	12	14	6
8	13		15

3		2
1	7	4
6	8	5

1.1.3: Trạng thái bắt đầu 15-puzzle và 8-puzzle

2. Cài đặt A*

Việc cài đặt thuật toán A* trong bài toán N-Puzzle cũng giống như phần trên đã nêu:

- **FRINGE** là tập chứa các trạng thái đã được sinh ra nhưng chưa được xét đến.
- **M** là tập các trạng thái tiếp theo của trạng thái n_i

- **KQ** tập trạng thái kết quả, lưu các trạng thái từ trạng thái hiện tại tới đích

Đầu vào: trạng thái hiện tại, trạng thái đích

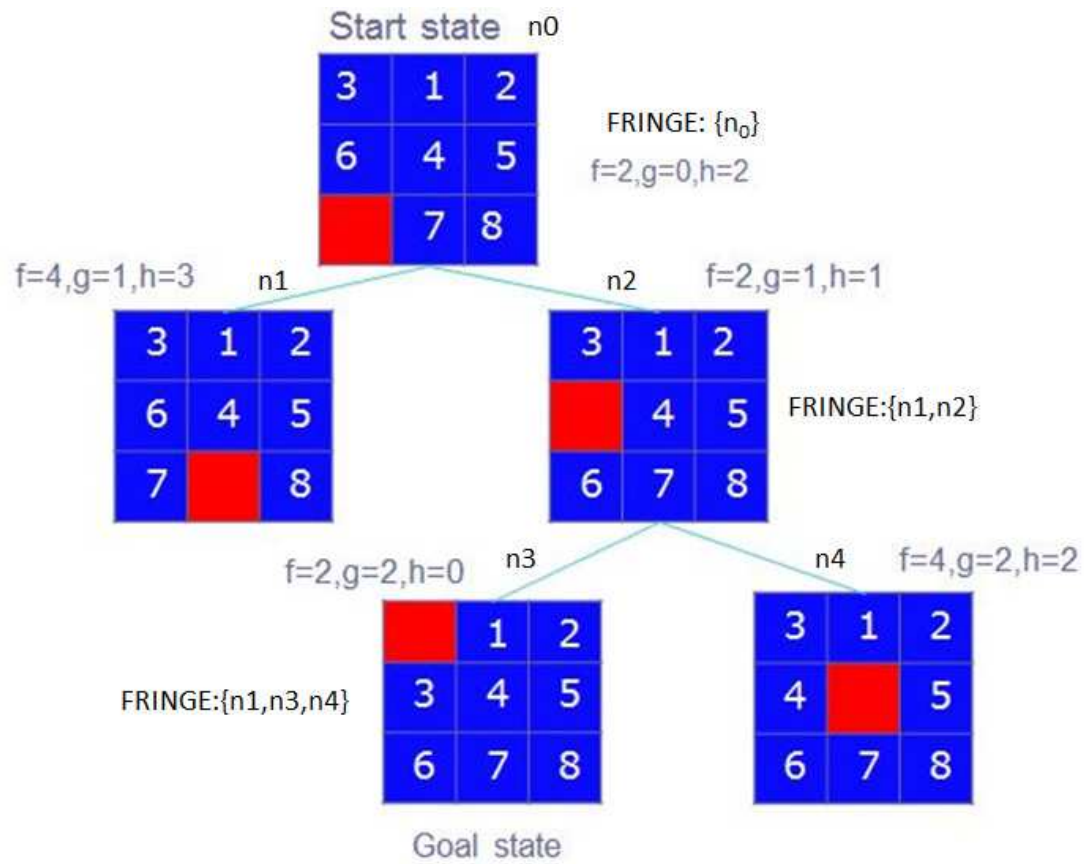
Đầu ra: tập các trạng thái từ trạng thái hiện tại tới trạng thái đích

Điều kiện dừng thuật toán: tìm thấy kết quả hoặc giới hạn thời gian hoặc người dùng cho phép dừng.

Trong bài toán này ta có thể cải tiến bằng việc bỏ tập **CLOSE**. Ta thấy trong bước 2.b.3 ở trên sau khi tìm ra các trạng thái con của trạng thái n_i . Khi đó n_i có tối đa 4 trạng thái con, nhưng trong các trạng thái con của n_i có 1 trạng thái trùng với trạng thái cha của n_i . Vì vậy ta có thể loại bỏ trạng thái này để tránh việc xét lặp. Khi loại bỏ trạng thái này sẽ không tồn tại một trạng thái con nào trùng với một trạng thái trong tập **CLOSE** nữa. Việc loại bỏ này sẽ tránh được việc kiểm tra ở bước 2.b.3.3 gây tốn rất nhiều thời gian.

Ngoài ra, trong game này còn cài đặt thêm thuật toán A^* sâu dần(IDA^*) là một biến thể của thuật toán tìm kiếm A^* . IDA^* giúp loại bỏ hạn chế bộ nhớ của A^* mà ko hy sinh giải pháp tối ưu. Mỗi lần lặp của thuật toán là quá trình tìm kiếm theo chiều sâu, $f(n) = g(n) + h(n)$, tạo nút mới. Khi một nút được tạo ra có chi phí vượt quá một ngưỡng cutoff thì nút đó sẽ bị cắt giảm, quá trình tìm kiếm backtracks trước khi tiếp tục. Các ngưỡng chi phí được khởi tạo ước lượng heuristic của trạng thái ban đầu, và trong mỗi lần lặp kế tiếp làm tăng tổng chi phí của các nút có chi phí thấp đã được cắt tĩa trước đó. Thuật toán chấm dứt khi trạng thái đích có tổng chi phí không vượt quá ngưỡng hiện tại.

Minh họa A*

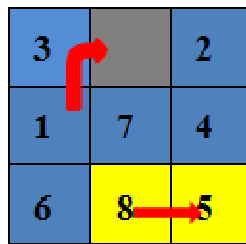


1.1.4: Minh họa A*

3. Hàm ước lượng heuristic

3.1 Các hàm ước lượng heuristic

3.1.1 heuristic1 = tổng khoảng cách dịch chuyển ($\leftarrow, \rightarrow, \uparrow, \downarrow$) ngắn nhất để dịch chuyển các ô sai về vị trí đúng của nó (khoảng cách Manhattan)



3		2
1	7	4
6	8	5

1.1.5: Minh họa

Giả sử: $+ row_d, col_d$ là tọa độ (dòng và cột) của ô ở vị trí đúng

$+ row_s, col_s$ là tọa độ (dòng và cột) của ô ở vị trí sai

$+ xd = |col_s - col_d|$

$+ yd = |row_s - row_d|$

$\Rightarrow d = xd + yd$ là khoảng cách ngắn nhất để di chuyển 1 ô về đúng vị trí

$\Rightarrow h1 = \sum d$

Trong bảng số 3x3 trên, để di chuyển ô số 8 về vị trí đúng cần 1 lần, để di chuyển ô số 1 về vị trí đúng cần 2 lần (qua 2 ô khác). Để thu được kết quả này ta làm phép tính đơn giản: lấy tổng khoảng cách của các dòng và cột giữa hai vị trí (vị trí hiện tại và vị trí đúng)

- Lấy tọa độ của ô số 1 ở vị trí hiện tại ta có: $row_s = 1, col_s = 0$
- Lấy tọa độ ô số 1 ở vị trí đúng: $row_d = 1/3 = 0; col_s = 1\%3 = 1$
- Vậy khoảng cách ngắn nhất để di chuyển ô số 1 về vị trí đúng:

$$d_1 = |row_s - row_d| + |col_s - col_d| = |1 - 0| + |0 - 1| = 2$$

Tương tự, tính tất cả các khoảng cách d của các ô sai còn lại ta được

$$h_1 = 1+0+2+1+1+0+1+1 = 7$$

3.1.2 heuristic2 = tổng khoảng cách dịch chuyển ($\leftarrow, \rightarrow, \uparrow, \downarrow$) ngắn nhất để dịch chuyển các ô sai về vị trí đúng của nó cộng thêm chỉ số phạt cặp ô hàng xóm với nhau đang nằm ngược vị trí của nhau.

	1	2
3	4	5
6	7	8

1.1.6a: Đích

	2	1
6	7	5
3	8	4

1.1.6b: Minh họa

$$\Rightarrow h_2 = \sum d + a$$

Trong đó **a** là chỉ số phạt cặp ô hàng xóm đang nằm ngược vị trí. Cặp (2,1) muốn về đúng vị trí cần dịch chuyển ít nhất 4 bước (không đề ý tới các ô khác), 2 bước đã được tính trong $\sum d$ nên **a** = 2. Vì vậy trong 1.1.6b có 2 cặp hàng xóm nằm ngược vị trí nên ở đây $a = 2+2 = 4$.

3.1.3 heuristic3

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

1.1.7a: Đích

2	6	3	7
4	5	10	11
12	9	14	1
8	13		15

1.1.7b: Minh họa

Xét 1 ô nằm sai vị trí: $d = |row_s - row_d|^2 + |col_s - col_d|^2$

Đặt $d_3 = \sum d$

$$\Rightarrow h_3 = d_3 - [0.15 * d_3] + a$$

3.1.4 heuristic4

Xét 1 ô sai nằm sai vị trí: $d = |row_s - row_d|^2 + |col_s - col_d|^2$

Đặt $d_4 = \sum d$

$$\Rightarrow h_4 = d_4 + a$$

3.2 Ví dụ so sánh 3 hàm heuristic

8	7	1
6		4
3	2	5

Xét trạng thái hình trên

$$+ \text{heuristic1} = 4 + 2 + 1 + 1 + 1 + 1 + 3 + 1 = 14$$

$$+ \text{heuristic2} = \text{heuristic1} + 2 = 16 \quad (a = 2)$$

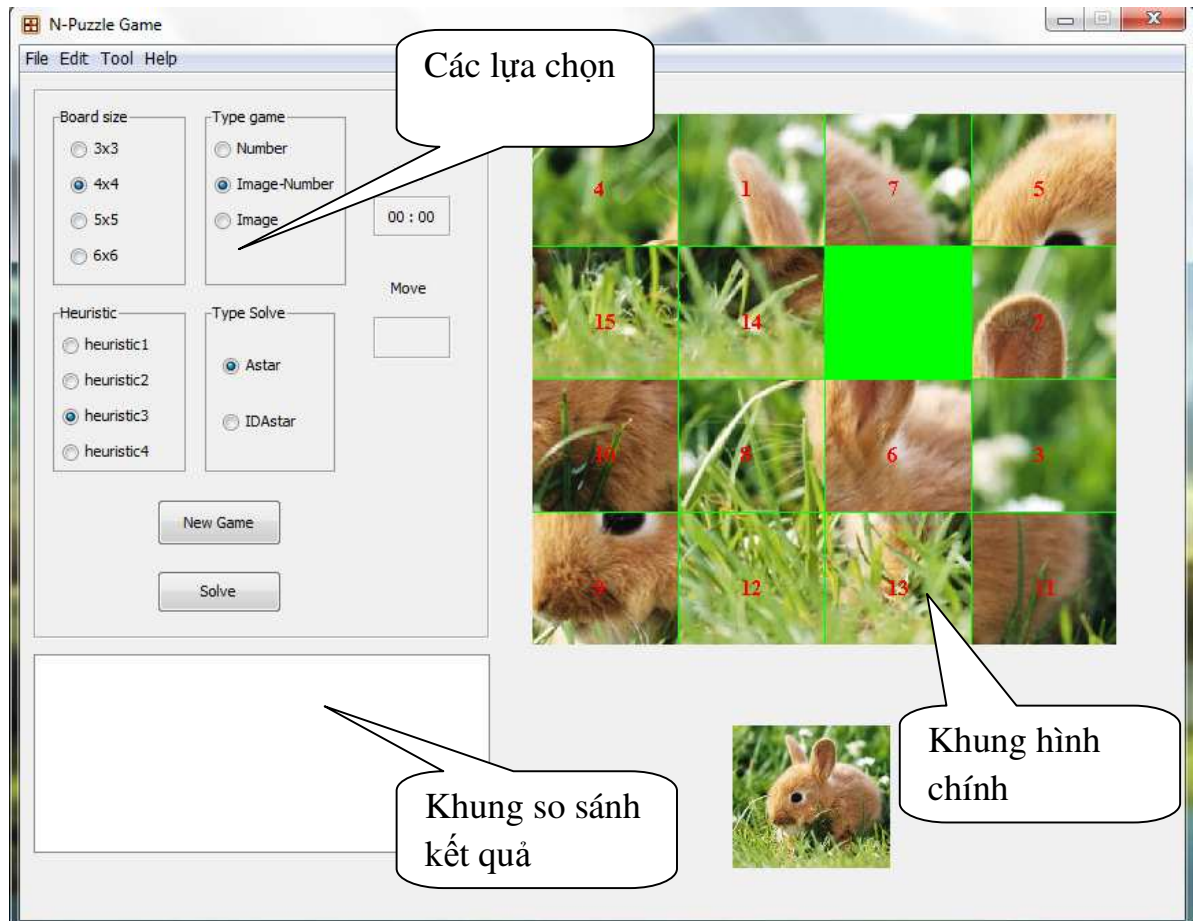
$$+ d_{34} = 8 + 4 + 1 + 1 + 1 + 1 + 5 + 1 = 22$$

$$+ \text{heuristic3} = d_{34} - [0.15 * d_{34}] + 2 = 21$$

$$+ \text{heuristic4} = d_{34} + 2 = 24$$

III- KẾT QUẢ

1- Giao diện



2- So sánh

Với trạng thái bắt đầu là trạng thái ở hình trên:

❖ Thuật toán A*

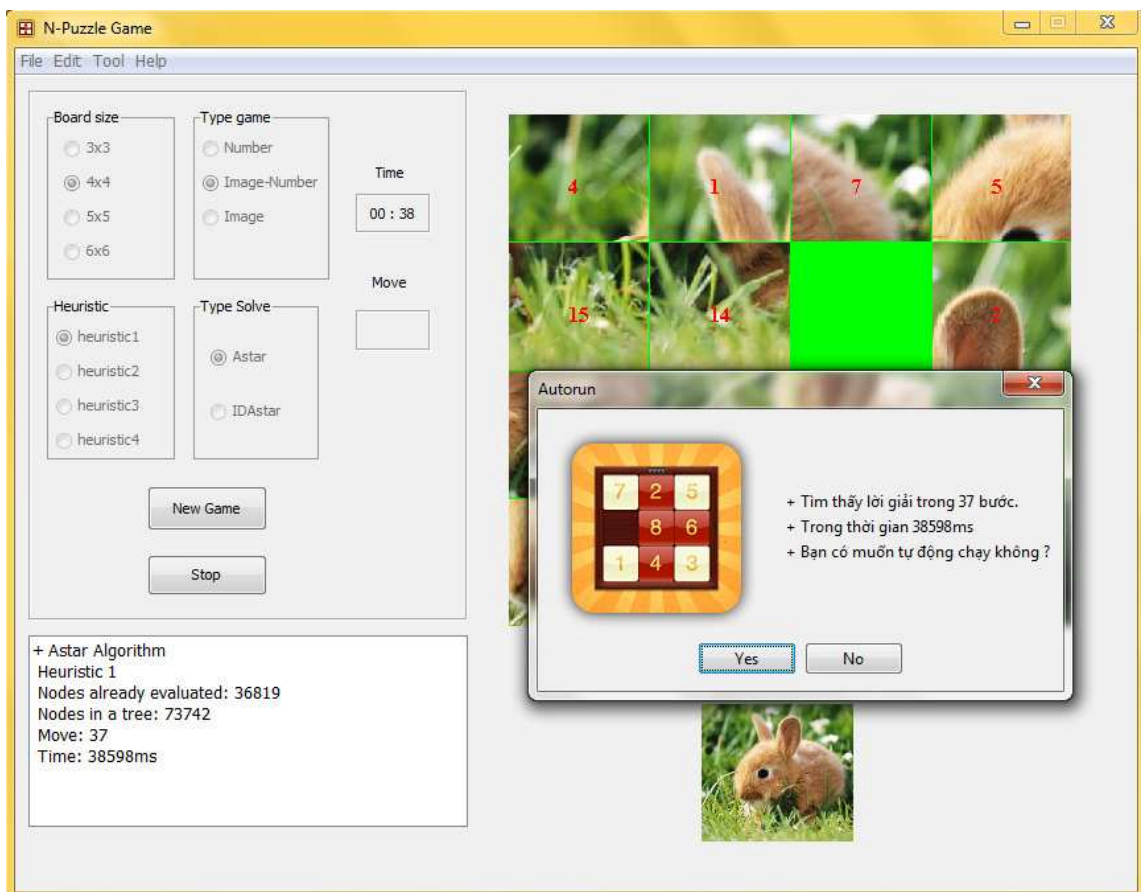
+ heuristic 1:

Số bước thực hiện: 37

Số nút đã xét: 36819

Tổng số nút trên cây: 73742

Thời gian giải quyết: 38598ms



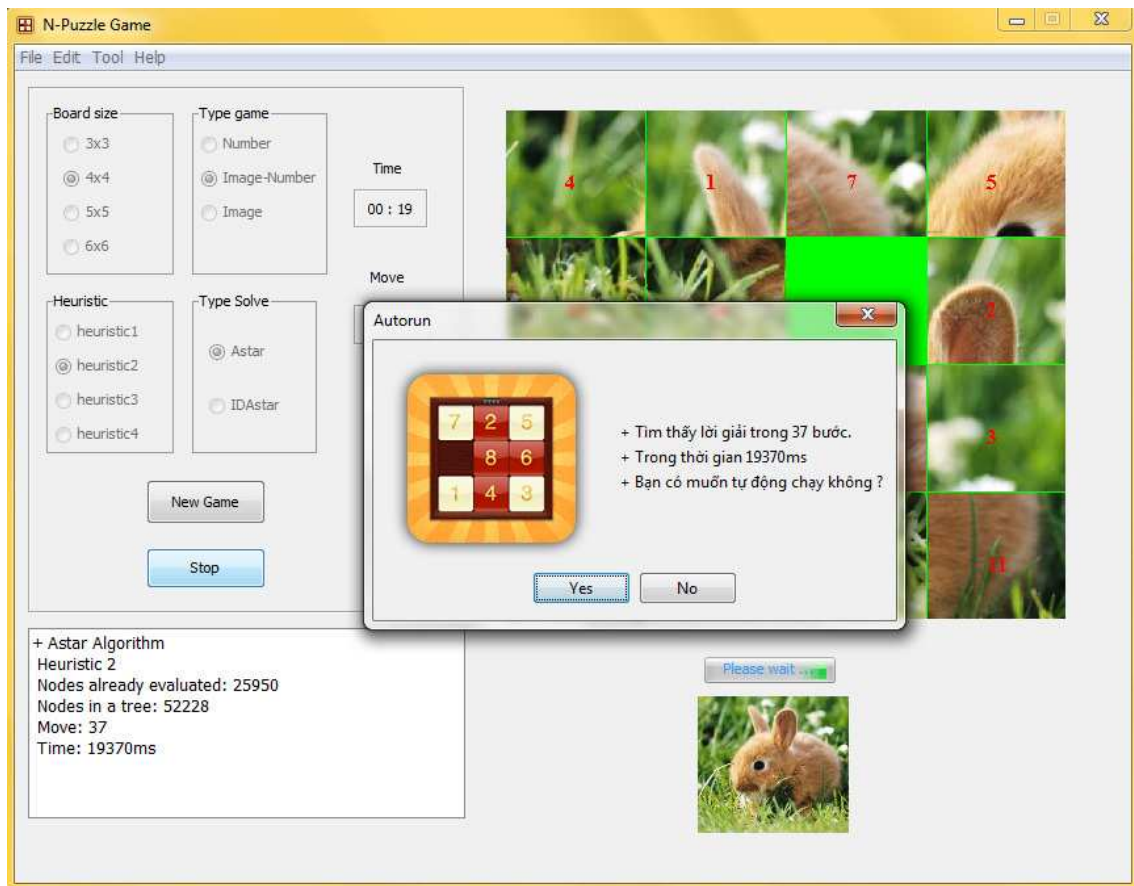
+ heuristic2

Số bước thực hiện: 37

Số nút đã xét: 25950

Tổng số nút trên cây: 52228

Thời gian giải quyết: 19370ms



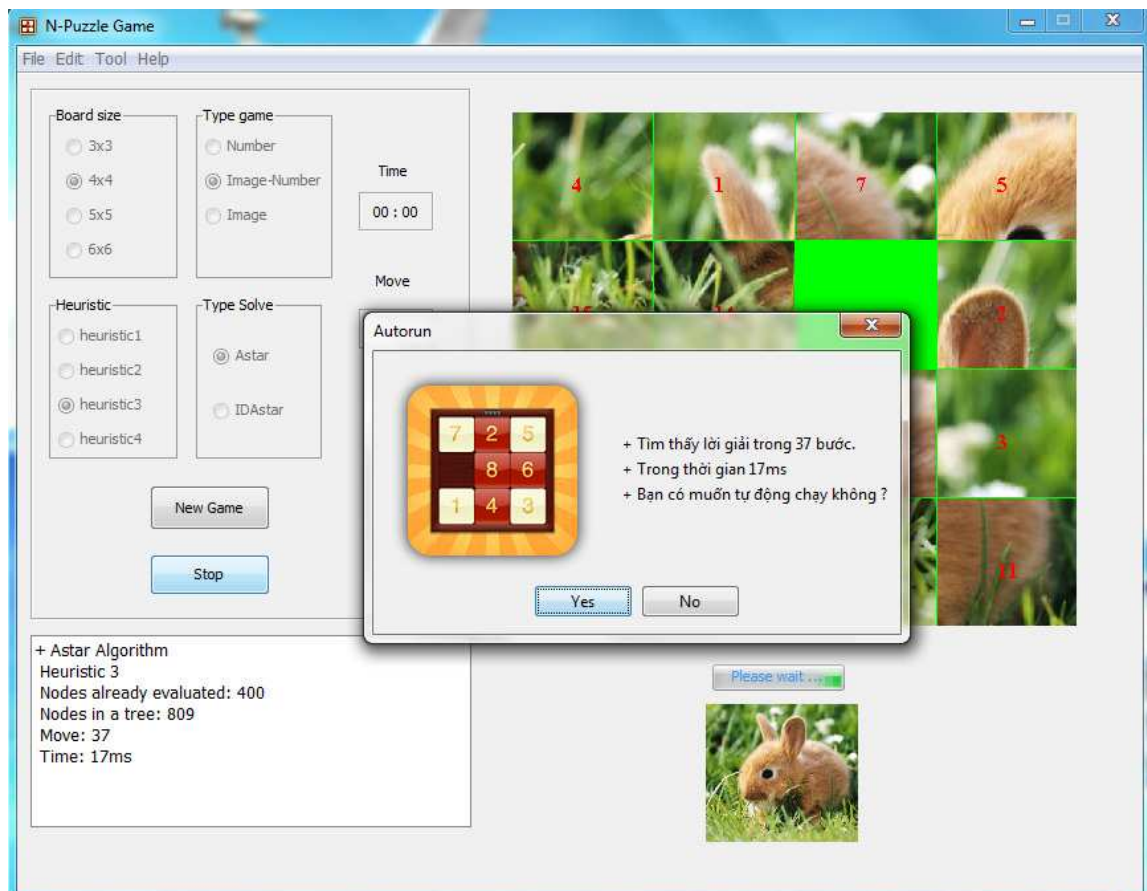
+ heuristic3:

Số bước thực hiện: 37

Số nút đã xét: 400

Tổng số nút trên cây: 809

Thời gian giải quyết: 17ms



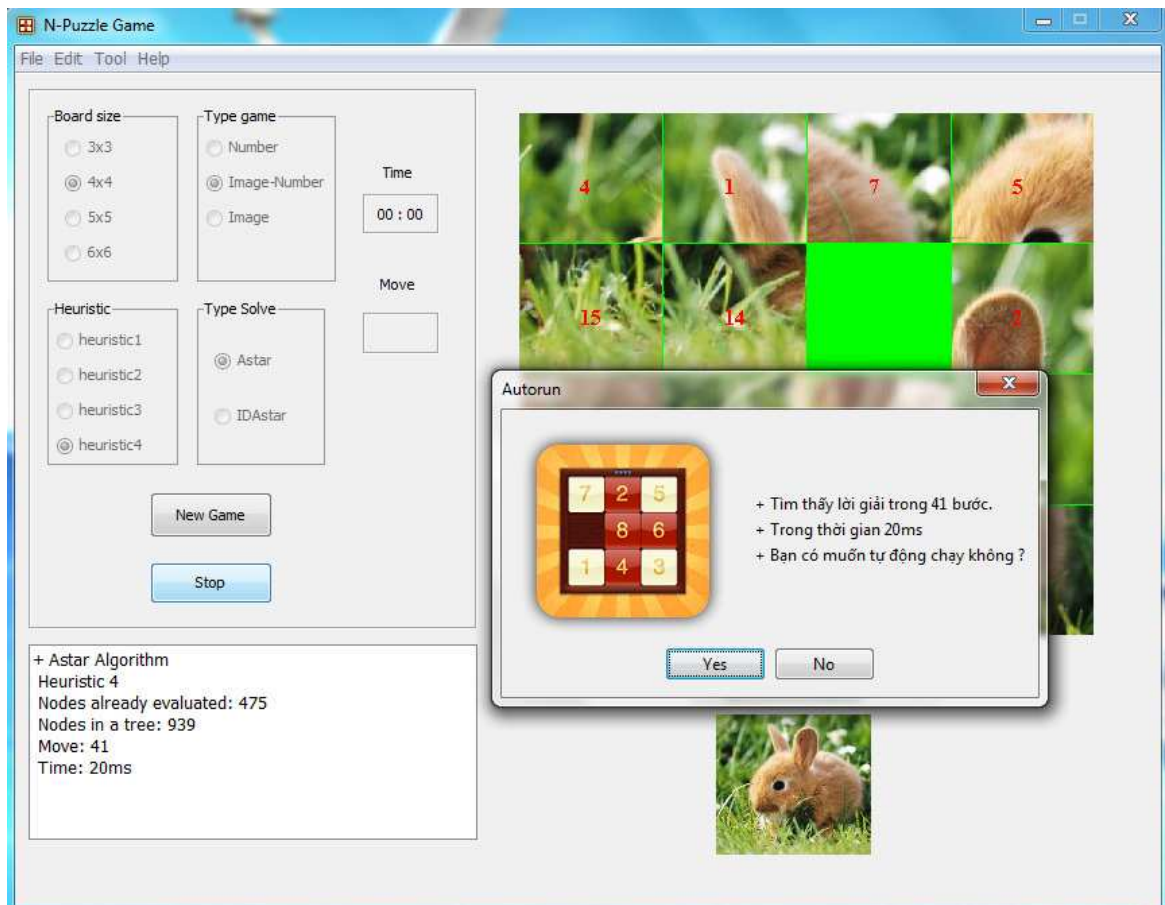
+ heuristic4

Số bước thực hiện: 41

Số nút đã xét: 475

Tổng số nút trên cây: 939

Thời gian giải quyết: 20ms



❖ Thuật toán IDA*

+ heuristic1

Số bước thực hiện: 37

Số nút đã xét: 77849

Tổng số nút trên cây: 156896

Thời gian giải quyết: 9760ms

+ heuristic2

Số bước thực hiện: 37

Số nút đã xét: 48304

Tổng số nút trên cây: 97311

Thời gian giải quyết: 4081ms

+ heuristic3

Số bước thực hiện: 37

Số nút đã xét: 404

Tổng số nút trên cây: 811

Thời gian giải quyết: 4ms

+ heuristic4

Số bước thực hiện: 43

Số nút đã xét: 4834

Tổng số nút trên cây: 9622

Thời gian giải quyết: 41ms

3- Nhận xét

- Ta thấy $heuristic2 = heuristic1 + a$, nên $h1(n) \leq h2(n) \leq h^*(n)$. Hàm $h2(n)$ giúp số nút duyệt ít hơn và thời gian duyệt nhanh hơn $h1(n)$. Vì vậy $h2(n)$ hiệu quả hơn $h1(n)$. Hai hàm $heuristic1$ và $heuristic2$ tỏ ra kém hiệu quả khi kích thước trạng thái của bài toán tăng lên, dẫn đến mất nhiều thời gian và tốn bộ nhớ. Nguyên nhân là do ước lượng chi phí $heuristic1$ và $heuristic2$ quá nhỏ so với chi phí thực tế $h^*(n)$.

- Hàm $heuristic3$ là một ước lượng chi phí khá tối ưu, không gian trạng thái và thời gian duyệt giảm đi đáng kể so với hai hàm ước lượng trên. Nguyên nhân là do hàm ước lượng chi phí $heuristic3 \approx h^*(n)$ (gần với chi phí thực tế).

- Hàm $heuristic4$ cũng khá hiệu quả về mặt bộ nhớ và thời gian khi duyệt, nhưng $heuristic4$ không đưa ra đường đi tối ưu vì $heuristic4$ ước lượng chi phí lớn hơn chi phí thực tế. Có thể nói, trong 4 hàm ước lượng $heuristic3$ làm hàm hiệu quả nhất.

- Tính tối ưu của thuật toán A^* phụ thuộc nhiều vào hàm ước lượng $h(n)$ và chỉ phù hợp với không gian trạng thái nhỏ. Nếu không gian các trạng thái là hữu hạn và có giải pháp tránh việc xét lặp lại các trạng thái thì giải thuật A^* là hoàn chỉnh (tìm được lời giải) - nhưng không đảm bảo tính tối ưu. Nếu không có giải pháp để tránh việc xét lặp thì A^* không hoàn chỉnh (không đảm bảo tìm được lời giải). Nếu không gian trạng thái là vô hạn thì giải thuật A^* là không hoàn chỉnh (không đảm bảo tìm được lời giải).

- IDA^* hiệu quả hơn A^* về mặt bộ nhớ và thời gian. Nói chung IDA^* nhanh hơn A^* , nhưng IDA^* cũng không đảm bảo tính tối ưu.

- Trong bài toán này ta có thể cân nhắc giữa việc tìm đường đi tối ưu nhưng sẽ mất nhiều thời gian với việc tìm đường đi không tối ưu với thời gian nhanh hơn.

- Hàm $heuristic3$ tuy khá hiệu quả nhưng không mở rộng được với mọi không gian trạng thái vì khi A^* xét các nút thì phải tính toán $heuristic$ gây tốn nhiều thời gian. Nhóm đang nghiên cứu phương pháp tính trước các hàm $heuristic$ nhưng do thời gian có hạn nên chưa thể hoàn thành được.

IV- KẾT LUẬN

Thông qua việc tìm hiểu và nghiên cứu đề tài này giúp chúng tôi có cái nhìn toàn diện hơn trong việc ứng dụng trí tuệ nhân tạo vào giải quyết vấn đề thực tế. Đây là bài toán cổ điển trong trí tuệ nhân tạo cho các thuật toán mô hình hóa liên quan đến tìm kiếm có tri thức bổ sung. Đề tài đã được nhiều người nghiên cứu giải quyết, nhưng cho đến nay vẫn chưa có cách giải quyết tối ưu cho tất cả không gian trạng thái trò chơi vì kích thước tăng không gian trạng thái sẽ tăng lên rất nhanh. Hy vọng những nghiên cứu đánh giá của chúng tôi sẽ góp phần bổ sung thêm một hướng giải quyết cho bài toán. Do thời gian có hạn nên đề tài không tránh khỏi những sai sót, mong thầy góp ý, đánh giá giúp chúng tôi hoàn thiện đề tài.

Tài liệu tham khảo

- Bộ thư viện chuẩn của Sun Microsystems
<http://www.oracle.com/us/technologies/java/index.html>
- Bài giảng Nhập môn trí tuệ nhân tạo – Nguyễn Nhật Quang
- http://en.wikipedia.org/wiki/A*_search_algorithm
- http://en.wikipedia.org/wiki/Fifteen_puzzle
- <http://yinyangit.wordpress.com/2010/12/16/algorithm-phan-tich-va-gi%E1%BA%A3i-bai-toan-n-puzzle/>

Hà Nội, ngày 15 tháng 04 năm 2013

Tác giả BTL

Lê Đình Cường