**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**

**ĐỀ TÀI:**

**TÌM HIỂU VÀ CÀI ĐẶT THUẬT TOÁN TÌM KIẾM HILL CLIMBING**

Giáo viên hướng dẫn: **MAI CƯỜNG THỌ**

Sinh viên thực hiện: **BÙI VĂN VIỆT**

**NGUYỄN MINH THẢO**

**TRẦN TIẾN**

Lớp**: 60.CNTT-3**

**Năm học: 2020 - 2021**

**Đề bài: Áp dụng thuật toán Hill-Climbing để giải bài toán 8-Queen**

Phần I: Giới thiệu về bài toán 8-Queen

1. Giới thiệu:

Bài toán tám quân hậu là bài toán đặt tám quân hậu trên bàn cờ vua kích thước 8×8 sao cho không có quân hậu nào có thể “ăn” được quân hậu khác, hay nói khác đi không quân hậu nào có để di chuyển theo quy tắc cờ vua. Màu của các quân hậu không có ý nghĩa trong bài toán này. Như vậy, lời giải của bài toán là một cách xếp tám quân hậu trên bàn cờ sao cho không có hai quân nào đứng trên cùng hàng, hoặc cùng cột hoặc cùng đường chéo.

1. Đề bài:

**A picture containing text, crossword puzzle

Description automatically generated**Cho một bàn cờ kích thước 8\*8. Chúng ta cần đặt n (n>=4) quân hậu vào bàn cờ sao cho chúng không tấn công được nhau, tức là không có cặp con hậu nào nằm cùng hàng, cùng cột, cùng đường chéo.

**Phần II: Thuật toán Hill-Climbing để giải bài toán 8-Queen:**

Giải thuật leo đồi chính là nền tảng cơ sở của các kỹ thuật tìm kiếm cục bộ. Mặc dù đây là giải thuật đơn giản nhưng nó lại rất mạnh và hiệu quả trong việc giải quyết các bài toán CSP lớn. Thuật ngữ “leo đồi” (hill-climbing) xuất phát từ cơ chế “tu chỉnh lập”: ở mỗi của việc tìm kiếm, chúng ta sẽ chọn một bước chuyển mà nó cải thiện giá trị hàm mục tiêu để thực hiện. Giải thuật leo đồi được mô tả như sau:

**Procedure** Hill\_Climbing\_Search;

**Begin**

Khởi tạo danh sách L chỉ chứa trạng thái đầu;

**Loop do**

**If** L rỗng **then** {thông báo thất bại; **stop**;}

Loại trạng thái u đầu danh sách L;

**If** u là trạng thái kết thúc của **then** {thông báo thành công; **stop**;}

**For** mỗi trạng thái v kề u đặt v vào L sao cho các phần tử được đưa vào đầu danh sách L có đánh giá giảm dần;

End;

Trong giải thuật leo đồi, chỉ những bước chuyển cải thiện được hàm chi phí hoặc không làm cho hàm chi phí thay đổi mới được chọn vì vậy việc tìm kiếm sẽ liên tục bước lên vị trí cao hơn cho đến khi nó gặp điều kiện dừng. Khuyết điểm của giải thuật leo đồi:

Diagram

Description automatically generatedVấn đề lớn nhất của giải thuật leo đồi mắc phải là nó sẽ bị rơi vào vùng tối ưu cục bộ, đó là lúc chúng ta leo lên một đỉnh mà chúng ta không thể tìmđược láng giềng nào tốt hơn được nữa nhưng đỉnh này lại không phải đỉnh cao nhất.

Khi leo đến một đỉnh tối ưu cục bộ, để tìm được lời giải tốt hơn nữa ta có ta có thể lặp lại thuật toán leo đồi với nhiều điểm xuất phát khác nhau được chọn ngẫu nhiên và lưu lại kết quả tốt nhất ở mỗi lần lặp. Nếu số lần lặp đủ lớn thì ta có thể tìm được đỉnh tối ưu toàn cục, tuy nhiên với những bài toán có không gian tìm kiếm khổng lồ (chẳng hạn như bài toán xếp lịch) ta không thể đưa ra số lần lặp đủ lớn để đảm bảo tìm được lời giải tối ưu. Cho nên đây là phương pháp giải quyết không mang lại nhiều hiệu quả, chúng ta sẽ khảo sát các giải thuật cải tiến khác khắc phục được vấn đề này ở phần sau.

Nhận xét: hiệu quả của giải thuật leo đồi phụ thuộc rất nhiều vào “bề mặt” của không gian tìm kiếm bài toán. Nếu bài toán chỉ có vài đỉnh tối ưu cục bộ thì giải thuật sẽ không tìm ra lời giải tối ưu rất nhanh, tuy nhiên trong trường hợp bề mặt không gian tìm kiếm lại quá lồi lõm (rất nhiều đỉnh tối ưu cục bộ), giải thuật sẽ bị luẩn quẩn trong vùng tối ưu cục bộ và có thể không tìm ra được lời giải tối ưu của bài toán.

1. Lời giải cho thuật toán (áp dụng Hill Climbing):

Ý tưởng thuật toán: Xếp tất cả N con Hậu lên bàn cờ (mỗi con hậu chiếm

1 ô cờ). Mỗi cách xếp như vậy được gọi là 1 State (trạng thái).Ta cần một trạng thái ban đầu, có thể là ngẫu nhiên. Trạng thái đích là trạng thái mà các quân hậu không thể tấn công nhau. Từ trạng thái ban đầu, ta thực hiện thay đổi vị trí quân cờ để tạo ra trạng thái tiếp theo. Hàm chi phí tính bằng số quân hậu có thể bị tấn công. Ta sẽ chọn trạng thái nào có giá trị hàm nhỏ nhất. Và để cải thiện hàm chi phí, chúng ta sẽ chọn quân hậu nào có khả năng bị tấn công nhiều nhất dịch chuyển tạo ra trạng thái mới.

B1: Khởi tạo bàn cờ ngẫu nhiên các vị trí của quân hậu. Ta gọi đây là trạng thái ban đầu.

B2: Chọn quân cờ có khả năng bị tấn công nhiều nhất. Thực hiện dịch chuyển các nước đi phù hợp để tạo ra các trạng thái con.

B3: Từ các trạng thái con này chọn ra trạng thái có hàm chi phí nhỏ nhất. Chọn nó là thái đầu tiên và lưu chúng vào danh sách trạng thái được chọn.

B4 Lặp lại B2 cho đến khi không còn quân cờ nào trong tình trạng bị tấn công.

1. Mã giả:

***//khởi tạo hàm cung cấp điểm bắt đầu cho thuật toán***

void configureRandomly(int board[][N],

int\* state)

{

***//lặp lại các chỉ số cột***

for (int i = 0; i < N; i++) {

***// lấy chỉ mục hàng ngẫu nhiên***

state[i] = rand() % N;

***//Đặt quân hậu vào vị trí vừa lấy được trong bàn cờ***

board[state[i]][i] = 1;

}

}

***// hàm tính toán giá trị mục tiêu của trạng thái(quân hậu tấn công nhau)***

int calculateObjective(int board[][N],

int\* state)

{

***//với mỗi quân hậu trong cột, ta kiểm tra các quân hậu khác có nằm trong hàng của quân hậu này***

***//nếu tìm thấy thì tăng biến số tấn công***

***// số lượng các quân hậu tấn công nhau***

***//ban đầu = 0***

int attacking = 0;

***// tạo biến để định nghĩa hàng và cột***

int row, col;

for (int i = 0; i < N; i++) {

***// tại mỗi cột 'i', quân hậu được đặt ở hàng state[i]***

***// bên trái cùng 1 hàng(hàng không đổi và cột giảm)***

row = state[i], col = i - 1;

while (col >= 0 && board[row][col] != 1) {

col--;

}

if (col >= 0 && board[row][col] == 1) {

attacking++;

}

***// bên phải cùng 1 hàng(hàng không đổi và cột tăng lên)***

row = state[i], col = i + 1;

while (col < N && board[row][col] != 1) {

col++;

}

if (col < N && board[row][col] == 1) {

attacking++;

}

***// Theo đường chéo sang trái lên(hàng và cột giảm dần)***

row = state[i] - 1, col = i - 1;

while (col >= 0 && row >= 0 && board[row][col] != 1) {

col--;

row--;

}

if (col >= 0 && row >= 0 && board[row][col] == 1) {

attacking++;

}

***// Theo đường chéo sang phải xuống(hàng và cột tăng đồng thời)***

row = state[i] + 1, col = i + 1;

while (col < N && row < N && board[row][col] != 1) {

col++;

row++;

}

if (col < N && row < N && board[row][col] == 1) {

attacking++;

}

***// Theo đường chéo từ trái xuống(cột giảm và hàng tăng)***

row = state[i] + 1, col = i - 1;

while (col >= 0 && row < N && board[row][col] != 1) {

col--;

row++;

}

if (col >= 0 && row < N && board[row][col] == 1) {

attacking++;

}

***// Theo đường chéo sang phải lên(cột tăng và hàng giảm)***

row = state[i] - 1, col = i + 1;

while (col < N && row >= 0 && board[row][col] != 1) {

col++;

row--;

}

if (col < N && row >= 0 && board[row][col] == 1) {

attacking++;

}

}

***// trả về các cặp***

return (int)(attacking / 2);

}

***//hàm nhận trạng thái hiện tại có giá trị nhỏ nhất trong số các trạng thái lân cận***

void getNeighbour(int board[][N], int\* state)

{

***//khai báo và khởi tạo bảng và trạng thái tối ưu(optimal)***

***//với bảng hiện tại và trạng thái là điểm bắt***

int opBoard[N][N];

int opState[N];

copyState(opState, state);

generateBoard(opBoard, opState);

***//khởi tạo giá trị mục tiêu tối ưu***

int opObjective = calculateObjective(opBoard, opState);

***//khai báo và khởi tạo bảng và trạng thái tạm thời cho mục đích tính toán***

int NeighbourBoard[N][N];

int NeighbourState[N];

copyState(NeighbourState, state);

generateBoard(NeighbourBoard, NeighbourState);

***//lặp lại thông qua tất cả các trạng thái lân cận khả thi của bảng***

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

***//điều kiện để bỏ qua trạng thái hiện tại***

if (j != state[i]) {

***//khởi tạo trạng thái lân cận tạm thời với trạng thái lân cận hiện tại***

NeighbourState[i] = j;

NeighbourBoard[NeighbourState[i]][i] = 1;

NeighbourBoard[state[i]][i] = 0;

***//tính giá trị khả thi của trạng thái lân cận***

int temp = calculateObjective(NeighbourBoard, NeighbourState);

***//so sánh các mục tiêu tạm thời và tối ưu lân cận***

***//nếu tạm thời nhỏ hơn tối ưu thì cập nhật cho phù hợp***

if (temp <= opObjective)

{

opObjective = temp;

copyState(opState, NeighbourState);

generateBoard(opBoard, opState);

}

***//cấu hình ban đầu cho lần lặp tiếp theo***

NeighbourBoard[NeighbourState[i]][i] = 0;

NeighbourState[i] = state[i];

NeighbourBoard[state[i]][i] = 1;

}

}

}

***//sao chép bảng và trạng thái tối ưu được tìm thấy đến bảng và trạng thái hiện tại***

copyState(state, opState);

fill(board, 0);

generateBoard(board, state);

}

***//hàm hill climbing***

void hillClimbing(int board[][N], int\* state)

{

***//khai báo và khởi tạo bảng và trạng thái lận cận với bảng và trạng thái hiện tại là điểm bắt đầu***

int neighbourBoard[N][N] = {};

int neighbourState[N];

copyState(neighbourState, state);

generateBoard(neighbourBoard, neighbourState);

do {

***//sao chép bảng và trạng thái lân cận sang bảng và trạng thái hiện tại, từ lân cận trở thành hiện tại sau bước này***

copyState(state, neighbourState);

generateBoard(board, state);

***//lấy trạng thái lân cận tối ưu***

getNeighbour(neighbourBoard, neighbourState);

if (compareStates(state, neighbourState)) {

***//nếu lân cận và hiện tại bằng nhau thì không tồn tại lân cận tối ưu và do đó xuất ra kết quả(ngắt vòng lặp)***

printBoard(board);

break;

}

else if (calculateObjective(board,state) == calculateObjective(neighbourBoard, neighbourState))

{

***//nếu lận cận và hiện tại không bằng nhau nhưng mục tiêu của chúng bằng nhau thì chúng ta đang tiến gần đến mức tối ưu cục bộ***

***//trong mọi trường hợp, nhảy tới một trạng thái lân cận bất kì để thoát khỏi nó***

***//trạng thái lân cận ngẫu nhiên***

neighbourState[rand() % N] = rand() % N;

generateBoard(neighbourBoard, neighbourState);

}

} while (true);

}

1. Kết luận

Bài toán 8 quân hậu là bài toán phổ biến để đánh giá thuật toán. Trong trường hợp này, thuật toán leo đồi vẫn phát huy được tính đơn giản và hiệu quả.

Thuật toán leo đồi được áp dụng trong bài toán 8 quân hậu là phiên bản cơ bản. Chỉ có một bước cải tiến là cải thiện hàm chi phí bằng cách chọn quân cờ có khả năng bị tấn công nhiều nhất.