Báo cáo chương trình

Giải bài toán sudoku sử dụng heuristic

Ngày 15 tháng 07 năm 2025

Họ và tên : Trương Công Hoan

MSSV: 2302700335

Học phần : Introduction to Artificial Intelligence

Giảng viên : Nguyễn Quản Bá Hồng

Mục lục:

1. Mục tiêu:

Bài tập nhằm xây dựng một ứng dụng giải và chơi Sudoku với các kích thước 9x9 và 16x16, hỗ trợ chế độ Sudoku X (kiểm tra thêm hai đường chéo chính). Ứng dụng sử dụng thuật toán backtracking kết hợp heuristic Minimum Remaining Values (MRV) để giải Sudoku, đồng thời cung cấp giao diện đồ họa

thân thiện sử dụng thư viện tkinter của Python. Các tính năng bao gồm tạo đề tự động, kiểm tra lời giải, cung cấp gợi ý, hiển thị lời giải, lưu/tải đề, và lưu kết quả.

2. Mô tả thuật toán:

2.1. Bài toán Sudoku

Sudoku là một bài toán thỏa mãn ràng buộc (Constraint Satisfaction Problem - CSP) trên lưới NxN (N=9 hoặc N=16), với các ràng buộc:

- Mỗi hàng, cột, và vùng box_size x box_size (3x3 hoặc 4x4) chỉ chứa các giá trị từ 1 đến N mà không trùng lặp.
- Với chế độ Sudoku X (N=9), hai đường chéo chính và phụ cũng không được chứa giá trị trùng lặp.
- Mục tiêu: Điền các ô trống sao cho thỏa mãn tất cả ràng buộc.

2.2. Thuật toán giải

Chương trình sử dụng các thuật toán sau:

1. **Backtracking**:
    - Thử từng giá trị hợp lệ cho ô trống, giải đệ quy, và quay lui nếu thất bại.
    - Được sử dụng trong hàm solve_sudoku để giải bảng, generate_full_solution để tạo bảng giải đầy đủ, và count_solutions để kiểm tra tính duy nhất của lời giải.

- Độ phức tạp: O(N^(N^2)) trong trường hợp xấu, nhưng được tối ưu bằng MRV.

2. **Minimum Remaining Values (MRV)**:
    - Chọn ô trống có ít giá trị khả thi nhất để giảm số lần quay lui.
    - Được triển khai trong hàm find_mrv_cell, giúp chọn ô tiếp theo cần điền.
    - Ưu điểm: Giảm không gian tìm kiếm, cải thiện hiệu suất.
3. **Randomization**:
    - Xáo trộn giá trị và vị trí ô cố định để tạo đề và lời giải ngẫu nhiên, tăng tính đa dạng.
    - Được sử dụng trong generate_full_solution (xáo trộn danh sách giá trị) và generate_puzzle (chọn ngẫu nhiên ô cố định).
4. **Constraint Checking**:
    - Hàm is_valid kiểm tra tính hợp lệ của giá trị tại một ô dựa trên ràng buộc hàng, cột, vùng, và đường chéo (nếu Sudoku X).
    - Được sử dụng trong tất cả các hàm giải và kiểm tra.

2.3. Chiến lược tạo đề

- Tạo bảng giải đầy đủ bằng generate_full_solution.
- Chọn ngẫu nhiên các ô cố định dựa trên độ khó (số ô trống: 30/40/50 cho 9x9 và 100/140/180 cho 16x16).
- Kiểm tra đề có duy nhất một lời giải bằng count_solutions.
- Nếu không thành công, thử lại hoặc giảm số ô trống.

2.4. Giao diện đồ họa

- Sử dụng tkinter để tạo giao diện với lưới nhập liệu, menu điều khiển (kích thước, độ khó, Sudoku X), và các nút chức năng (Tạo đề, Kiểm tra, Gợi ý, Lời giải, Hoàn thành, Nhập và giải, Xóa bảng, Lưu/Tải đề, Hướng dẫn).
- Đồng hồ thời gian thực hiển thị thời gian chơi (phút:giây:milligiây).
- Màu sắc và trạng thái ô giúp phân biệt ô cố định (xám), ô người dùng (trắng), ô gợi ý (xanh lá), ô sai (đỏ), và ô lời giải (xanh nhạt).

3. Cấu trúc chương trình

Danh sách hàm chính:

- get_symbols(N): Trả về danh sách ký hiệu (1-9 hoặc 1-9 và A-F).

- is_valid(board, row, col, val, N, box_size, is_sudoku_x): Kiểm tra tính hợp lệ của giá trị tại ô.

- find_mrv_cell(board, N, box_size, is_sudoku_x, symbols): Tìm ô MRV.

- solve_sudoku(board, N, box_size, is_sudoku_x, symbols, trial_count): Giải bảng Sudoku.

- is_initial_board_valid(board, N, box_size, is_sudoku_x): Kiểm tra tính hợp lệ của bảng ban đầu.

- generate_full_solution(N, box_size, symbols, is_sudoku_x): Tạo bảng giải đầy đủ.

- count_solutions(board, N, box_size, is_sudoku_x, symbols, limit): Đếm số lời giải.

- generate_puzzle(N, box_size, symbols, is_sudoku_x, empty_cells): Tạo đề Sudoku.

- Lớp SudokuGUI: Quản lý giao diện và tương tác người dùng.

Biến chính:

board: Ma trận NxN lưu trạng thái bảng Sudoku.

trial_count: Đếm số lần thử trong thuật toán backtracking.

self.entries: Danh sách ô nhập liệu (tk.Entry) trong GUI.

self.solution: Lưu bảng lời giải.

self.empty_cells: Đếm số ô trống còn lại.

4. Giải thích mã nguồn Python

Dưới đây là giải thích một số đoạn mã quan trọng từ chương trình:

4.1. Hàm find_mrv_cell

```
def find_mrv_cell(board, N, box_size, is_sudoku_x, symbols):
    min_options = N + 1
    best_cell = None
    for row in range(N):
        for col in range(N):
            if board[row][col] == '0':
                options = [val for val in symbols if is_valid(board, row, col, val, N, box_size, is_sudoku_x)]
```

```
            if not options:
                return None
            if len(options) < min_options:
                min_options = len(options)
                best_cell = (row, col, options)
                if min_options == 1:
                    return best_cell
    return best_cell
```

Mô tả: Tìm ô trống có ít giá trị khả thi nhất (MRV).

Chi tiết:

- Duyệt qua tất cả ô trống (board[row][col] == '0').
- Tính danh sách options (giá trị hợp lệ) bằng is_valid.
- Cập nhật best_cell nếu số options nhỏ hơn min_options.
- Thoát sớm nếu tìm thấy ô với chỉ 1 giá trị khả thi.
- Trả về None nếu có ô trống không có giá trị hợp lệ (bảng không giải được).

4.2. Hàm solve_sudoku

```
def solve_sudoku(board, N, box_size, is_sudoku_x, symbols, trial_count):
    cell = find_mrv_cell(board, N, box_size, is_sudoku_x, symbols)
```

```
    if cell is None:
        return all(all(cell != '0' for cell in row) for row in board)
    row, col, options = cell
    for val in options:
        trial_count[0] += 1
        board[row][col] = val
        if solve_sudoku(board, N, box_size, is_sudoku_x, symbols, trial_count):
            return True
        board[row][col] = '0'
    return False
```

Mô tả: Giải bảng Sudoku bằng backtracking với MRV.

Chi tiết:

- Tìm ô MRV bằng find_mrv_cell.
- Nếu không còn ô trống, kiểm tra bảng đã đầy chưa; nếu đúng, trả về True.
- Thử từng giá trị trong options, tăng trial_count, và giải đệ quy.
- Nếu thất bại, đặt lại ô về '0' và thử giá trị khác.
- Trả về False nếu không tìm được lời giải.

4.3. Hàm generate_puzzle

```
def generate_puzzle(N, box_size, symbols, is_sudoku_x, empty_cells,
max_attempts=20):
    for attempt in range(max_attempts):
        solution = generate_full_solution(N, box_size, symbols, is_sudoku_x)
        if not solution:
            continue
        puzzle = [['0' for _ in range(N)] for _ in range(N)]
        positions = [(r, c) for r in range(N) for c in range(N)]
        random.shuffle(positions)
        fixed_positions = positions[:N * N - empty_cells]
        for r, c in fixed_positions:
            puzzle[r][c] = solution[r][c]
        temp_board = copy.deepcopy(puzzle)
        trial_count[0] = 0
        if count_solutions(temp_board, N, box_size, is_sudoku_x, symbols) == 1 and
solve_sudoku(temp_board, N, box_size, is_sudoku_x, symbols, trial_count):
```

```
            return puzzle, solution
        if attempt == max_attempts - 1 and empty_cells > N * N // 4:
            return generate_puzzle(N, box_size, symbols, is_sudoku_x, empty_cells - 5,
max_attempts)
    raise ValueError("Không thể tạo đề hợp lệ sau nhiều lần thử.")
```

Mô tả: Tạo đề Sudoku với số ô trống xác định, đảm bảo có duy nhất một lời giải.

Chi tiết:

- Tạo bảng giải đầy đủ bằng generate_full_solution.
- Chọn ngẫu nhiên các ô cố định, số ô trống bằng empty_cells.
- Kiểm tra đề có duy nhất một lời giải bằng count_solutions và giải được bằng solve_sudoku.
- Nếu thất bại sau max_attempts, giảm số ô trống và thử lại.
- Ném lỗi nếu không tạo được đề hợp lệ.

4.4. Lớp SudokuGUI - Hàm generate_puzzle

```
def generate_puzzle(self):
    N = self.N.get()
    box_size = int(N ** 0.5)
    symbols = get_symbols(N)
```

```
is_x = self.is_sudoku_x.get()
difficulty = self.difficulty.get()
empty_cells = {
    "Dễ": 30 if N == 9 else 100,
    "Trung bình": 40 if N == 9 else 140,
    "Khó": 50 if N == 9 else 180
}[difficulty]
try:
    puzzle, solution = generate_puzzle(N, box_size, symbols, is_x, empty_cells)
    self.solution = solution
    self.start_time = time.perf_counter()
    self.timer_running = True
    self.empty_cells = sum(row.count('0') for row in puzzle)
    for r in range(N):
        for c in range(N):
            entry = self.entries[r][c]
            entry.config(state=tk.NORMAL)
```

```
            entry.delete(0, tk.END)
            if puzzle[r][c] != '0':
                entry.insert(0, puzzle[r][c])
                entry.config(fg="black", bg=self.colors["fixed_cell"],
state=tk.DISABLED)
            else:
                entry.config(fg="blue", bg=self.colors["user_cell"],
state=tk.NORMAL)
    except ValueError as e:
        messagebox.showerror("Lỗi", f"{str(e)} Hãy thử lại hoặc chọn độ khó thấp
hơn.")
        self.clear_board()
```

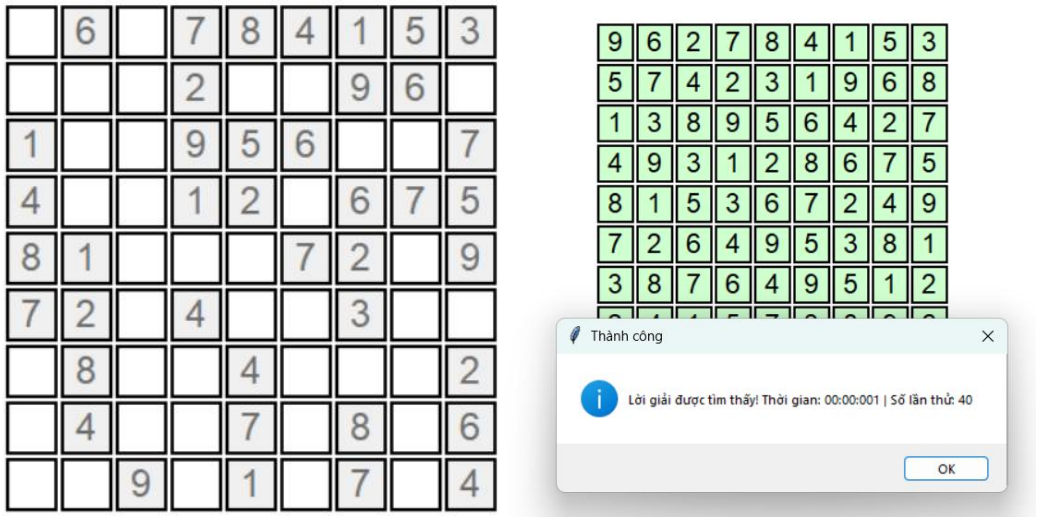
Mô tả: Tạo đề mới và hiển thị trên giao diện.

Chi tiết:

- Lấy tham số từ GUI: N, độ khó, is_sudoku_x.
- Xác định số ô trống dựa trên độ khó.
- Gọi generate_puzzle để tạo đề và lời giải.
- Hiển thị đề trên lưới, ô cố định màu xám và khóa, ô trống màu trắng và chỉnh sửa được.

- Khởi động đồng hồ và cập nhật empty_cells.

5. Kết quả mẫu



6. Đánh giá

- **Ưu điểm**:

  - Giao diện trực quan, dễ sử dụng với các tính năng đầy đủ.
  - Thuật toán backtracking với MRV hiệu quả cho 9x9, chấp nhận được cho 16x16.
  - Hỗ trợ lưu/tải đề và kết quả, giúp tái sử dụng và phân tích.

- **Hạn chế**:

    - Hiệu suất cho 16x16 có thể chậm với độ khó cao.
    - Thiếu kiểm tra thời gian thực khi nhập giá trị.
    - Không hỗ trợ các biến thể Sudoku khác (như Hyper Sudoku).

7.Kết luận :

Bài toán sudoku với thuật toán backtracking kết hợp MRV hiệu quả và giao diện đồ họa thân thiện. Các tính năng như tạo đề, kiểm tra, gợi ý, và lưu/tải đáp ứng tốt nhu cầu của người chơi và người nghiên cứu thuật toán. Trong tương lai, có thể cải tiến hiệu suất và thêm các biến thể Sudoku để tăng tính đa dạng.