BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO CUỐI KỲ

Đề tài: TÌM KIẾM LỜI GIẢI THỎA MÃN RÀNG BUỘC VÀ ỨNG DỤNG TRONG BÀI TOÁN TÔ MÀU ĐỒ THỊ

Giảng viên hướng dẫn : Trần Tiến Đức
Sinh viên thực hiện : Nguyễn Thị Hồng Thơ
MSSV : 22151305
Lớp : 22133B
Khóa : 2022
Mã lớp : ARIN330585_23_2_05

Thành phố Hồ Chí Minh, tháng 05 năm 2024

DANH SÁCH THAM GIA ĐỀ TÀI

HỌC KÌ II, NĂM HỌC: 2023 – 2024

Lớp: ARIN330585_23_2_05

Tên đề tài:

Tìm kiếm lời giải thỏa mãn ràng buộc và ứng dụng trong bài toán tô màu đồ thị

HỌ VÀ TÊN SINH VIÊN	MÃ SỐ SINH VIÊN
Nguyễn Thị Hồng Thơ	22151305

Nhận xét của giảng viên

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Ngày 19 tháng 05 năm 2024

Giảng viên chấm điểm

DANH MỤC HÌNH ẢNH

MỤC LỤC

CHƯƠNG 1: TỔNG QUAN VỀ RÀNG BUỘC

1. Ràng buộc trong các bài toán

Ràng buộc là một quan hệ trên một tập các biến.

Một bài toán thỏa mãn ràng buộc (Constraint Satisfaction Problem – CSP) bao gồm:

1. Một tập hữu hạn các biến X
2. Miền giá trị (một tập hữu hạn các giá trị) cho mỗi biến D
3. Một tập hữu hạn các ràng buộc C

Một lời giải (solution) của bài toán thỏa mãn ràng buộc là một phép gán đầy đủ các giá trị của các biến sao cho thỏa mãn tất cả các ràng buộc.

2. Các dạng ràng buộc

Ràng buộc đơn (unary constraint) chỉ liên quan đến 1 biến. Ví dụ: SA ≠ green

Ràng buộc nhị phân (binary constraint) liên quan đến 2 biến. Ví dụ: SA ≠ WA

Ràng buộc bậc cao (higher-order constraint) liên quan đến nhiều hơn 2 biến. Ví dụ: Các ràng buộc trong bài toán mật mã số học.

Lời giải bài toán ràng buộc có thể được giải bằng kiểm thử (Generate and Test) hoặc quay lui (backtracking). Quay lui là giải thuật tìm kiếm được sử dụng phổ biến nhất trong CSP.

3. Giải thuật quay lui

Bản chất của giải thuật RescusiveBacktracking là phép duyệt theo chiều sâu có thêm bước kiểm tra sự thỏa mãn của các ràng buộc mỗi bước:

- Mỗi lần gán, chỉ làm việc (gán giá trị) cho một biến.
- Tìm kiếm bằng kiểm thử: mỗi lần gán xác định các giá trị cho tất cả các biến.

Phương pháp tìm kiếm quay lui đối với bài toán CSP:

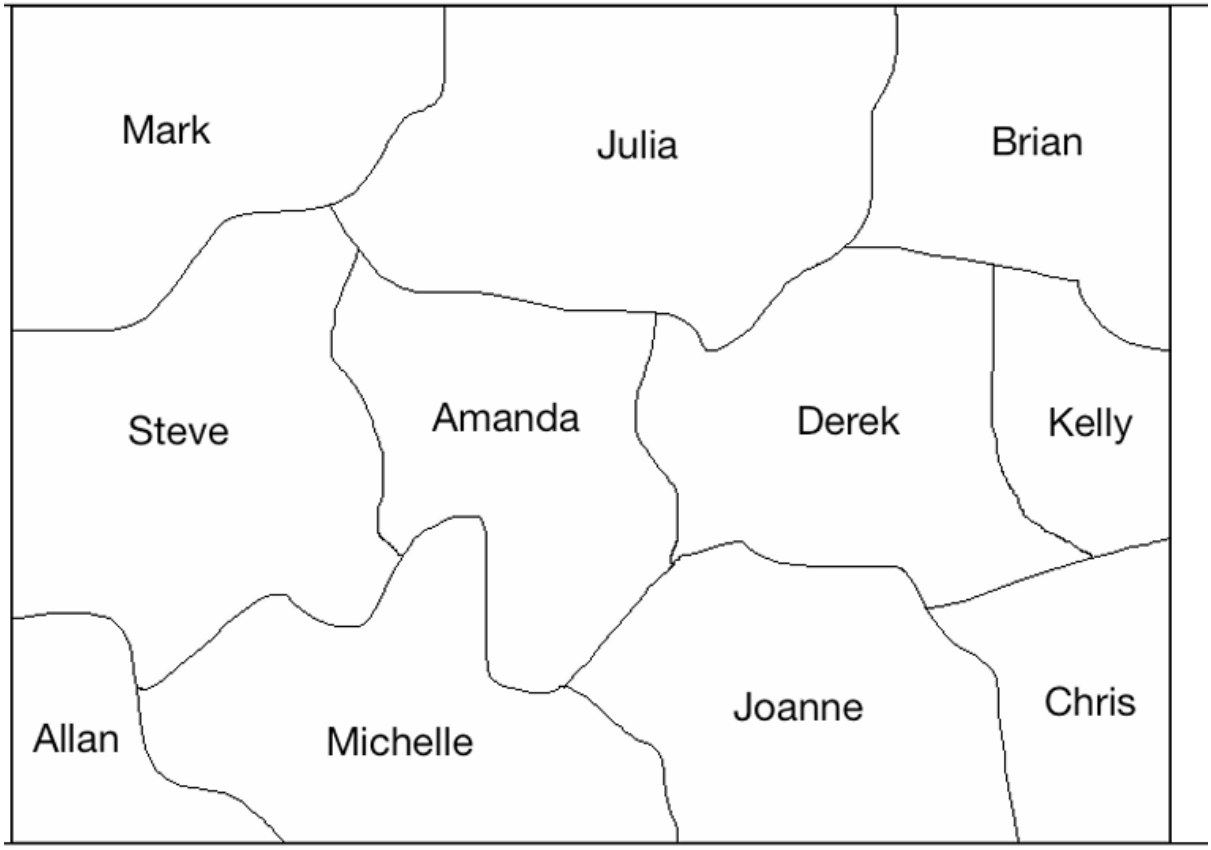
- Gán giá trị lần lượt cho các biến – Việc gán giá trị của biến này chỉ được làm sau khi đã hoàn thành việc gán giá trị của biến khác.
- Sau mỗi phép gán giá trị cho một biến nào đó, kiểm tra các ràng buộc có được thỏa mãn bởi tất cả các biến đã được gán giá trị cho đến thời điểm hiện tại – Quay lui (backtrack) nếu có lỗi (không thỏa mãn các ràng buộc).

Mã giả của giải thuật quay lui:

```
Function Backtracking-Search(problem) returns a solution, or failure
Return RescusiveBacktracking({},problem);
------------------------------------------------------------
Function RescusiveBacktracking(assignment, problem) returns a solution, or failure
if (length(assignment)==n) return assignment ;
 var  < Chọn_biến_chưa_gán(problem, assignment);
for each value in Miền_giá_trị(var,problem)
if KiemTraNhấtQuán(assignment U{var=value}, problem)
          assignment= assignment U{var=value}
          RescusiveBacktracking(assignment, problem);
        assignment= assignment - {var=value}
return failure;
```

CHƯƠNG 2: ỨNG DỤNG CỦA TÌM KIẾM CÓ RÀNG BUỘC VÀ QUAY LUI

1. Bài toán tô màu



Hình 1. Bản đồ bài toán tô màu

Bài toán: Sử dụng bốn màu để tô bản đồ các tỉnh của một nước sao cho các thành phố khác nhau thì có màu khác nhau.

Các biến: M, S, ALL, J, AMD, MCL, B, D, J, K, C (viết tắt tên các thành phố)

Các miền giá trị: {green, red, blue, gray}

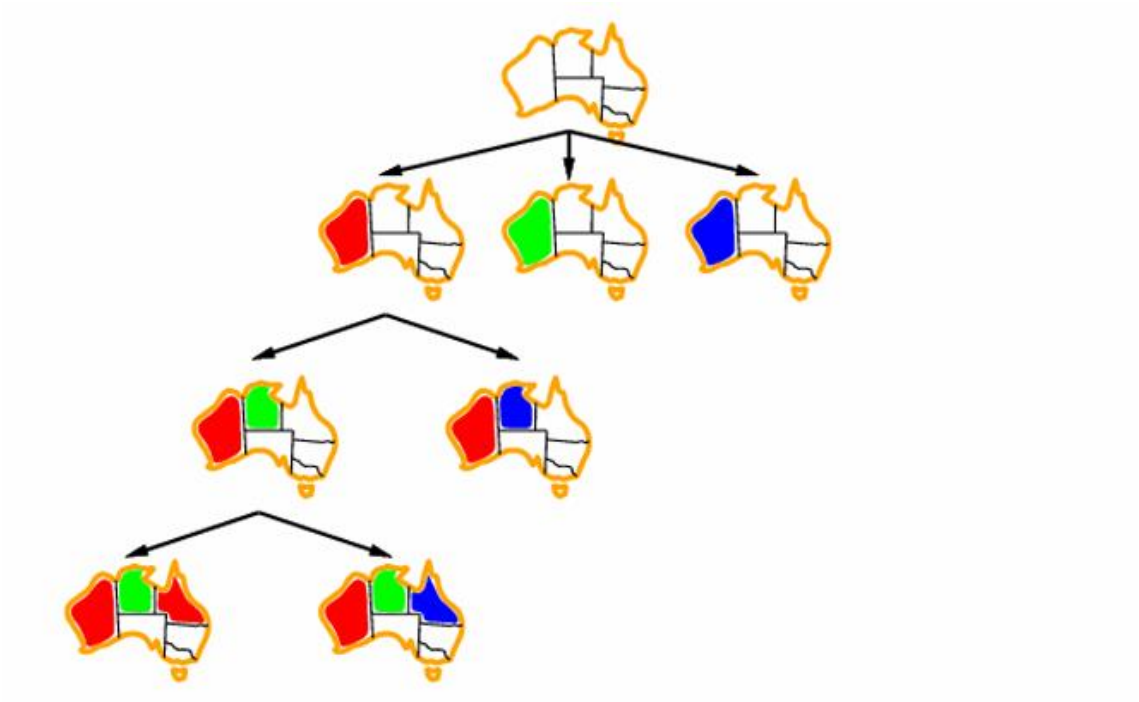
Các ràng buộc: Các vùng liền kề nhau phải có màu khác nhau.

Ví dụ lời giải:

$M \neq S$

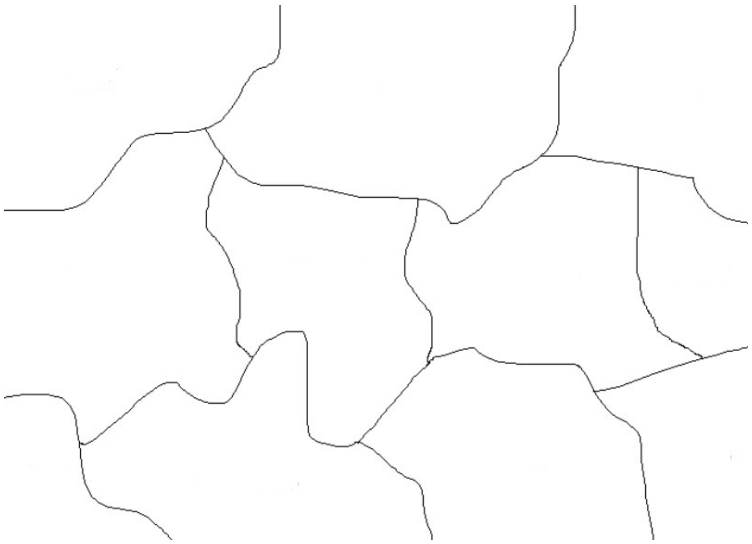
(M, S) = {red, green}, (red, blue), (red, gray),...}

Biến trong bài toán tô màu có thể biểu diễn bằng đồ thị sau:



Hình 2. Đồ thị bài toán tô màu

Tạo một hình ảnh ‘regions.jpg’ có hình dạng tương tự như bản đồ trên:



Hình 3. Bản đồ regions

Sử dụng backtrack từ simpleai.search:

from simpleai.search import CspProblem, backtrack

Code gốc: https://github.com/aimacode/aima-python/blob/master/search.pyl.

2. Code giao diện GUI:

#----

from simpleai.search import CspProblem, backtrack

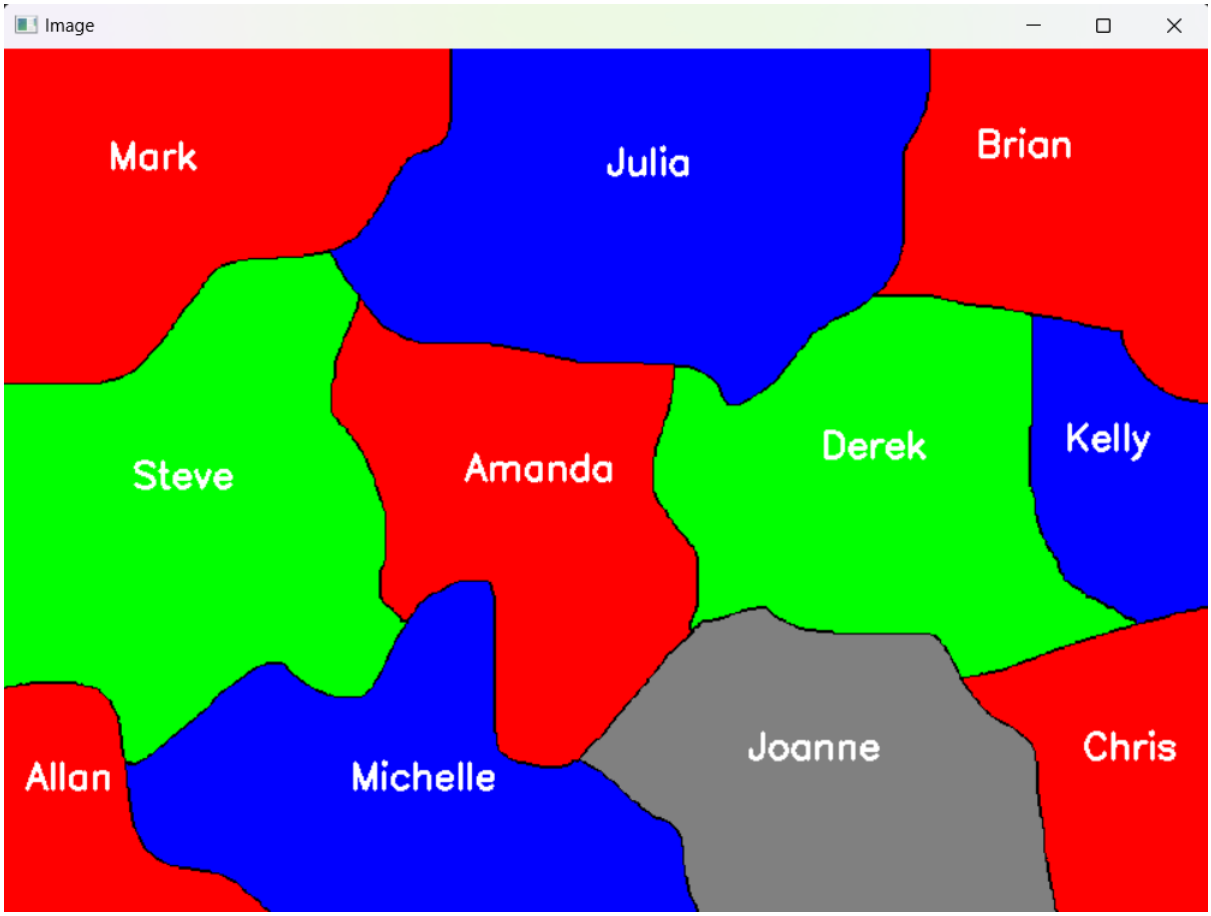
import cv2

```
import numpy as np
image = cv2.imread('regions.jpg', cv2.IMREAD_GRAYSCALE)
M, N = image.shape
val, image = cv2.threshold(image, 128, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)
image_s = image.copy()
image_s = cv2.cvtColor(image_s, cv2.COLOR_GRAY2BGR)
points_dict = {
    "Mark": (100, 88),
    "Julia": (430, 92),
    "Brian": (680, 80),
    "Steve": (120, 300),
    "Amanda": (357, 295),
    "Derek": (580, 280),
    "Kelly": (735, 275),
    "Allan": (45, 500),
    "Michelle": (280, 500),
    "Joanne": (540, 480),
    "Chris": (750, 480)
}
mau_red = (0, 0, 255)
mau_green = (0, 255, 0)
mau_blue = (255, 0, 0)
mau_gray = (128, 128, 128)

def constraint_func(names, values):
    return values[0] != values[1]

if __name__=='__main__':
    # Define the possible colors
    colors = dict((name, ['red', 'green', 'blue', 'gray']) for name in points_dict.keys())
```

```
# Define the constraints
constraints = [
    (('Mark', 'Julia'), constraint_func),
    (('Mark', 'Steve'), constraint_func),
    (('Julia', 'Steve'), constraint_func),
    (('Julia', 'Amanda'), constraint_func),
    (('Julia', 'Derek'), constraint_func),
    (('Julia', 'Brian'), constraint_func),
    (('Steve', 'Amanda'), constraint_func),
    (('Steve', 'Allan'), constraint_func),
    (('Steve', 'Michelle'), constraint_func),
    (('Amanda', 'Michelle'), constraint_func),
    (('Amanda', 'Joanne'), constraint_func),
    (('Amanda', 'Derek'), constraint_func),
    (('Brian', 'Derek'), constraint_func),
    (('Brian', 'Kelly'), constraint_func),
    (('Joanne', 'Michelle'), constraint_func),
    (('Joanne', 'Amanda'), constraint_func),
    (('Joanne', 'Derek'), constraint_func),
    (('Joanne', 'Kelly'), constraint_func),
    (('Derek', 'Kelly'), constraint_func),
]
# Define the colors
colors_dict = {
    'red': mau_red,
    'green': mau_blue,
    'blue': mau_green,
    'gray': mau_gray
}
problem = CspProblem(points_dict.keys(), colors, constraints)
output = backtrack(problem)
```

```
    print('\nColor mapping:\n')
    mask = np.zeros((M+2, N+2),np.uint8)
    for k, v in output.items():
        print(k, '==>', v)
        # Define the colors
        color = colors_dict[v]
        point = points_dict[k]
        cv2.floodFill(image_s, mask, point, color)
    font = cv2.FONT_HERSHEY_SIMPLEX
    font_scale = 0.8
    font_thickness = 2
    for k, v in points_dict.items():
        (text_width, text_height), _ = cv2.getTextSize(k, font, font_scale, font_thickness)
        text_offset_x = v[0] - text_width // 2
        text_offset_y = v[1] - text_height // 2
        cv2.putText(image_s, k, (text_offset_x, text_offset_y), font, font_scale, (255, 255,
255), font_thickness)
    cv2.imshow('Image', image_s)
    cv2.waitKey()
```

Kết quả giao diện:



Hình 4. Giao diện bài toán tô màu

Kết quả in ra màn hình:

```
Color mapping:

Mark ==> red
Julia ==> green
Brian ==> red
Steve ==> blue
Amanda ==> red
Derek ==> blue
Kelly ==> green
Allan ==> red
Michelle ==> green
Joanne ==> gray
Chris ==> red
```

Hình 5. Kết quả in ra màn hình bài toán tô màu

TÀI LIỆU THAM KHẢO

1. Alberto Artasanchez Prateek Joshi, Artificial Intelligence with Python, Packt Publishing Ltd, 2nd Edition, 2020.

2. Stuart J. Russell and Peter Norvig, Artificial Intelligence: A Modern Approach, 4th Edition, Pearson Education ©, 2021.

3. Từ Minh Phương, Giáo trình Nhập môn trí tuệ nhân tạo, Học Viện Công nghệ Bưu chính Viễn thông, 2014.

4. George F. Luger, William A. Stubblefield – Albuquerque – Artificial Intelligence – Wesley Publishing Company, Inc – 1997.

5. Phạm Văn Hải, Lê Thanh Hương, Nhập môn trí tuệ nhân tạo, Trường Đại học Bách Khoa Hà Nội.