Tecnológico Nacional de México, Campus Mexicali

Ingeniería Sistemas Computacionales

IA - Inteligencia Artificial

Tema:

“Tarea - Colorear Mapa”

Estudiante:

Mac Callum Merino Fatima Berenice

No. De control:

C21490774

Profesor:

Oscar Rubén Batista

Mexicali, B.C., 22 de octubre de 2025.

Tabla de contenido

Tabla de imágenes

Implementación de Coloreo de Regiones en un Mapa mediante CSP

Introducción

El problema de colorear un mapa es un clásico de la Inteligencia Artificial y la teoría de grafos. Consiste en asignar colores a regiones adyacentes de un mapa de tal manera que no haya dos regiones contiguas con el mismo color. Este desafío forma parte de los problemas de Satisfacción de Restricciones (CSP, por sus siglas en inglés) y se aplica en planificación, diseño de horarios y resolución de conflictos espaciales.

En este trabajo se implementa un programa en Python que detecta automáticamente regiones en un mapa, asigna nombres a estas regiones basándose en su disposición espacial y aplica un algoritmo de CSP para colorearlas respetando restricciones de adyacencia. La solución final se visualiza gráficamente sobre la imagen original.

Objetivo

El objetivo del trabajo es desarrollar un programa en Python capaz de:

- Detectar automáticamente regiones de un mapa usando procesamiento de imágenes.
- Asignar nombres a las regiones según su disposición espacial.
- Resolver el problema de coloreo usando un enfoque de CSP con técnicas de AC-3 y backtracking.
- Visualizar el resultado coloreado de manera gráfica, indicando claramente cada región y su color.

Fundamento teórico

El problema de coloreo de mapas puede formalizarse como un CSP donde:

- Cada región es una variable.
- Cada variable tiene un dominio limitado de colores posibles.
- Las restricciones dictan que dos regiones adyacentes no pueden compartir el mismo color.

Un CSP puede resolverse mediante técnicas como:

- AC-3 (Arc Consistency 3): Propaga restricciones para reducir dominios y detectar inconsistencias tempranas.
- Backtracking con heurísticas: Permite buscar soluciones de manera sistemática, utilizando estrategias como MRV (Minimum Remaining Values) y LCV (Least Constraining Value) para optimizar la búsqueda.

Este enfoque garantiza que se encuentre una asignación de colores válida si existe, incluso con un número mínimo de colores.

Motivación

Este ejercicio se realizó para aprender cómo aplicar Inteligencia Artificial y programación a un problema práctico: colorear un mapa de manera que las regiones vecinas no tengan el mismo color. Me permitió combinar conceptos de procesamiento de imágenes, algoritmos de búsqueda y resolución de problemas, y ver cómo la teoría se puede aplicar en la práctica.

Este trabajo es importante para mi formación porque me ayuda a entender cómo modelar problemas, usar herramientas de Python como OpenCV y Matplotlib, y aplicar estrategias de búsqueda y heurísticas para obtener soluciones válidas. Además, me da experiencia en proyectos donde hay que analizar, automatizar y visualizar resultados, habilidades útiles para la carrera de Ingeniería en Sistemas.

Descripción de actividades

1. Detección de regiones

Se utilizó OpenCV para cargar el mapa y convertirlo a escala de grises. Posteriormente se aplicó un umbral binario para resaltar las regiones y se encontraron los contornos correspondientes. Las regiones fueron filtradas por área mínima para eliminar ruido.

2. Cálculo de centroides

Para cada región significativa, se calcularon los centroides mediante momentos de los contornos. Estos centroides permiten ubicar espacialmente las regiones y facilitan la asignación de nombres.

3. Asignación de nombres

Basándose en la posición espacial de los centroides, se asignaron nombres a las regiones siguiendo la disposición geográfica esperada:

Fila superior: Mexicali (arriba-derecha), Tecate, Tijuana, Rosarito.

Otras regiones: Ensenada, San Felipe y San Quintín, asignadas según proximidad y área relativa.

4. Definición de restricciones

Se construyó un grafo de adyacencias basado en la descripción de las regiones, donde cada arista representa una restricción de que las regiones conectadas no deben compartir el mismo color.

5. Resolución mediante CSP

Se aplicó el algoritmo AC-3 para propagar consistencia de arcos y reducir dominios. Posteriormente, se utilizó backtracking con heurísticas MRV y LCV para buscar una asignación de colores válida para todas las regiones.

6. Visualización

Se generó una nueva imagen donde cada región se coloreó según la solución del CSP, usando una paleta definida de colores. Se escribieron los nombres de las regiones centrados en cada área para facilitar la identificación.

Resultados

Número de regiones: 7

```python
# Filtrar por área y calcular centroides
regiones = []
centros = []
for c in contornos:
    area = cv2.contourArea(c)
    if area >= MIN_AREA:
        M = cv2.moments(c)
        if M["m00"] == 0:
            continue
        cx = int(M["m10"] / M["m00"])
        cy = int(M["m01"] / M["m00"])
        regiones.append(c)
        centros.append((cx, cy, area))

if len(regiones) < TARGET_N:
    print(f"Advertencia: se detectaron {len(regiones)} regiones (esperado ~{TARGET_N}). Ajusta MIN_AREA.")
# Si hay más regiones que municipios, toma las más grandes
if len(regiones) > TARGET_N:
    # ordena por área descendente
    idxs = sorted(range(len(centros)), key=lambda i: centros[i][2], reverse=True)[:TARGET_N]
    regiones = [regiones[i] for i in idxs]
    centros = [centros[i] for i in idxs]

# Recalcular centros para las regiones seleccionadas
centros = []
for c in regiones:
    M = cv2.moments(c)
    cx = int(M["m10"] / M["m00"])
    cy = int(M["m01"] / M["m00"])
    area = cv2.contourArea(c)
    centros.append((cx, cy, area))

print(f"Regiones a usar: {len(regiones)}")
```

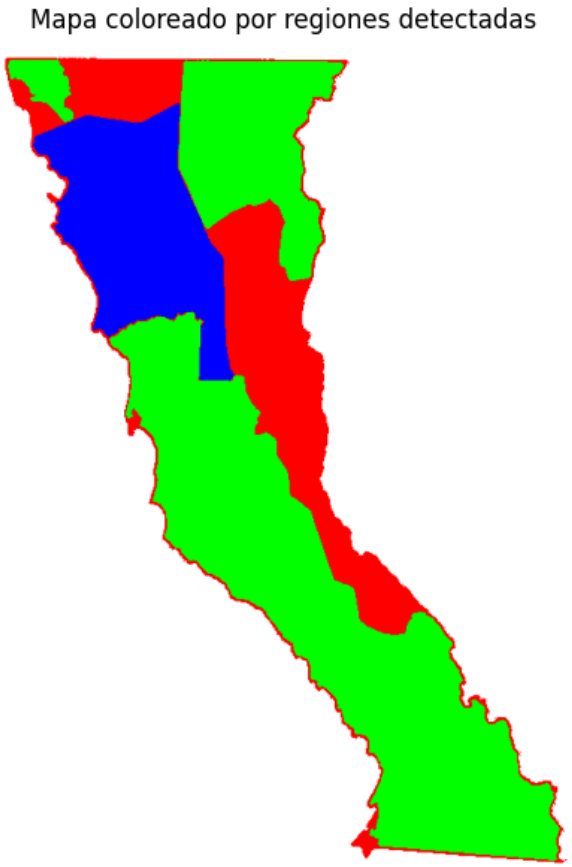
Método utilizado: CSP con AC-3 y backtracking

Colores utilizados: Rojo, Verde y Azul

```python
CONSTRAINTS = [
    ("Mexicali", "Tecate"),
    ("Mexicali", "San Felipe"),
    ("Tecate", "Tijuana"),
    ("Tecate", "Ensenada"),
    ("Tijuana", "Rosarito"),
    ("Tijuana", "Ensenada"),
    ("Rosarito", "Ensenada"),
    ("Ensenada", "San Quintin"),
    ("Ensenada", "San Felipe"),
    ("San Felipe", "San Quintin")
]
```

Representación gráfica: Cada región del mapa está coloreada según la solución encontrada, respetando todas las restricciones de adyacencia.

```python
# Guardar resultado y mostrar
cv2.imwrite("resultado.png", cv2.cvtColor(salida, cv2.COLOR_RGB2BGR))
plt.figure(figsize=(6,10))
plt.imshow(salida)
plt.axis("off")
plt.title("Mapa coloreado por regiones detectadas")
plt.show()
```



Conclusión

Este ejercicio permitió comprender cómo los problemas de coloreo de mapas pueden formalizarse como CSP y cómo algoritmos como AC-3 y backtracking con heurísticas eficaces permiten encontrar soluciones válidas. Se comprobó que es posible integrar procesamiento de imágenes y técnicas de Inteligencia Artificial para abordar problemas de planificación y asignación de recursos espaciales.

El trabajo refuerza conceptos de representación espacial, detección de regiones mediante contornos, propagación de restricciones y búsqueda heurística. Además, la visualización gráfica facilita la interpretación de los resultados, mostrando claramente cómo se cumple el objetivo de colorear un mapa respetando las restricciones de adyacencia.

Bibliografía

(N.d.). Edu.Mx. Retrieved October 18, 2025, from https://moodle.itmexicali.edu.mx/pluginfile.php/272064/mod_resource/content/1/3%20-%20Optimizacion%20-%20Intro%20IA%20Python%20-CS50.pdf [1]