Algoritmo Simulated Annealing

Optimización del Sensor Positioning Problem (SPP)

Autor: Sebastián Andrés Richiardi Pérez

Fecha: 18 de Noviembre 2025

Repositorio: github.com/KroderDev/sensor-positioning-problem

Introducción

El Problema (SPP):

El objetivo es determinar la ubicación óptima de p sensores (o zonas de manejo) en un terreno agrícola representado por una matriz de datos $N \times M$.

Objetivo del Algoritmo:

Minimizar el error cuadrático medio intra-zona, sujeto a restricciones de:

1. **Homogeneidad:** La varianza interna no debe superar un umbral α.
2. **Conexidad:** Las zonas deben ser contiguas.
3. **Forma:** Se establecen formas rectangulares (zonas de manejo operables).

Implementación:

Una metaheurística de **Simulated Annealing (SA)** en C++17, utilizando OpenCV para visualización y JSON para configuración.

Lectura de Archivos & IO

El sistema de Entrada/Salida (`IO.cpp`) maneja la carga de datos y parámetros.

Entradas:

1. **Instancia (`.spp`):** Dimensiones N, M y matriz de valores S (NDVI, pH, etc.).
2. **Configuración (`.json`):** Parámetros del algoritmo (Temperaturas, Iteraciones).
3. **Parámetros de Ejecución (Consola):** Cantidad de zonas p y factor de homogeneidad α.

Lectura de Archivos & IO

Pseudocódigo de Carga (IO.cpp):

```
Estructura ProblemInstance:
    S: Matriz[N][M] de doubles
    p: entero (zonas)
    alpha: double (restricción varianza)

IO::readInstanceFromFile(path):
    Abrir archivo .spp
    Leer N, M y matriz S celda por celda

IO::readParamsFromConsole(inst):
    Pedir p y alpha por stdin y guardarlos en inst

IO::readConfigFromJson(path):
    Construir SAConfig (T0, Tf, max_iterations, iters_per_temp,
    cooling_factor, max_time_seconds, penalty_weight)
```

Ejecución General

El flujo principal (`main.cpp`) orquesta los componentes.

Flujo de Control:

1. Carga de instancia y parámetros de usuario.
2. Carga de configuración SA desde JSON.
3. Generación de **Solución Inicial**.
4. Ejecución del **Simulated Annealing**.
5. Exportación de resultados:
    - Archivo `.out` con matriz de etiquetas.
    - Imagen `.png` (Heatmap con bordes superpuestos usando OpenCV).

Representación de la Solución

La solución se modela discretamente para permitir operaciones rápidas sobre la matríz.

Estructura de Datos (`Solution.hpp`):

```
struct Solution {
    // Matriz de tamaño NxM.
    // Z[i][j] = k, donde k es el ID de la zona (1..p)
    vector<vector<int>> Z;

    // Valor objetivo (suma de errores cuadráticos)
    double errorTotal;
};
```

Ventaja:

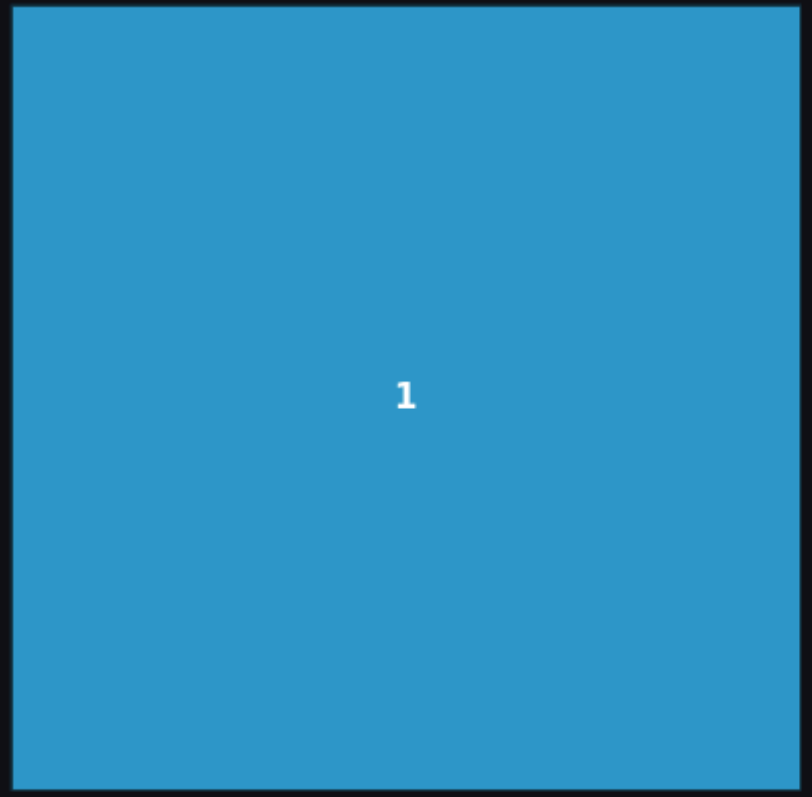
Esta representación permite verificar vecinos y calcular varianzas iterando directamente sobre la matriz, facilitando la implementación de restricciones de adyacencia.

Algoritmo - Solución Inicial

Para comenzar con una solución válida y razonable, se utiliza una heurística constructiva basada en cortes tipo "Guillotina".

Procedimiento (`buildInitialSolution`):

El objetivo es dividir el terreno recursivamente hasta obtener p rectángulos.



Algoritmo - Solución Inicial

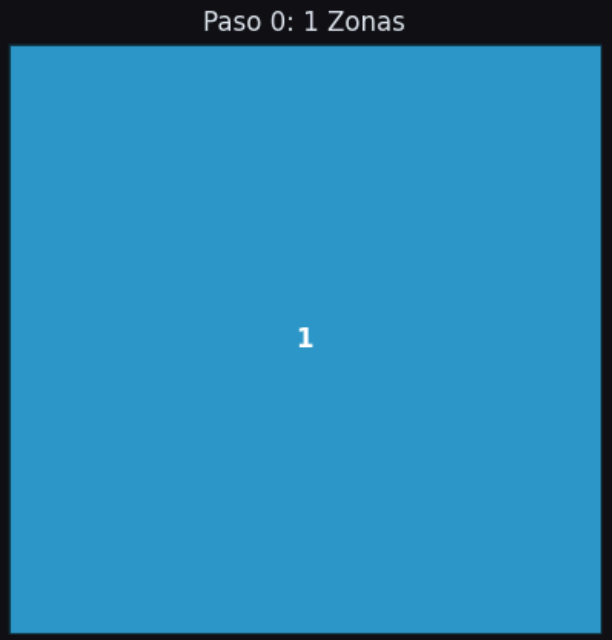
Pseudocódigo:

```
// buildInitialSolution(instance)
Lista rectangulos = { Rectangulo(0, 0, N-1, M-1) }

Mientras tamaño(rectangulos) < p:
    r = rectangulo de mayor área

    Si (r.alto >= 2 O r.ancho >= 2):
        corte = Horizontal/Vertical aleatorio
        r1, r2 = Dividir r en corte
        Reemplazar r con r1 y r2 en la lista
    Sino:
        break // no se puede dividir más

Asignar IDs 1..p sobre la matriz Z según los rectángulos
Calcular Error Inicial con calculateErrorAndVariance()
```



Configuración SA

El comportamiento del recocido se define externamente (`data/config/default.json`) para facilitar el ajuste de parámetros sin recompilar.

Parámetros Clave:

- **Esquema de Enfriamiento:** Geométrico ($T_{k+1} = T_k \cdot \text{cooling\_factor}$).
- **Penalización:** Peso W para violaciones de varianza.

Ejemplo de Configuración:

```
{
  "T0": 1000.0,              // Temp. Inicial
  "Tf": 0.001,               // Temp. Final
  "max_iterations": 100000,// Limite iteraciones
  "iters_per_temp": 100,    // Iteraciones por nivel de temp
  "cooling_factor": 0.95,   // Velocidad de enfriamiento
  "max_time_seconds": 10.0,// Tiempo maximo por corrida
  "penalty_weight": 1000.0 // Costo por violar varianza
}
```

Función de Evaluación (Energía)

El algoritmo permite explorar soluciones infactibles (que violan la varianza máxima) mediante una función de costo penalizada.

Función de Energía:

$$E(Z) = \text{ErrorTotal}(Z) + W \cdot \text{Penalización}(Z)$$

Cálculo de Penalización (`calculateVariancePenalty`):

```
VarianzaTotalGlobal = calculateTotalVariance(S)
Limite = alpha * VarianzaTotalGlobal
Penalizacion = 0

Para cada zona k en 1..p (según calculateErrorAndVariance):
    Si counts[k] == 0:
        ...
```

Función de Evaluación (Energía)

Cálculo de Penalización (`calculateVariancePenalty`):

```
VarianzaTotalGlobal = calculateTotalVariance(S)
Limite = alpha * VarianzaTotalGlobal
Penalizacion = 0

Para cada zona k en 1..p (según calculateErrorAndVariance):
    Si counts[k] == 0:
        Penalizacion += Limite   // zona vacía
    Si varianza(k) > Limite:
        Penalizacion += (varianza(k) - Limite)

Retornar Penalizacion (calculateVariancePenalty)
```

Nota: Esto guía al algoritmo suavemente de regreso a la región factible si se sale.

Generación de Vecindario

El operador de movimiento es crítico para mantener la forma de las zonas. No se mueven píxeles aleatorios, sino **límites de zonas**.

Generación de Vecino (`generateNeighbor`):

1. Seleccionar una zona k aleatoria.
2. Calcular su *Bounding Box* (límites actuales).
3. Elegir una dirección (Arriba, Abajo, Izquierda, Derecha) y una acción (Expandir o Contraer).
4. **Acción:** Mover la frontera de la zona en la dirección elegida, "robando" o "cediendo" celdas a las zonas adyacentes.
5. Validar conexidad con `isPartitionConnected`; si falla, se descarta el vecino.
6. Reparar forma con `makeRectsIfNonOverlapping` (solo si los bounding boxes no se solapan).

Generación de Vecindario

Reparación de Forma (`makeRectsIfNonOverlapping`): Tras el movimiento, se intenta forzar que la zona resultante sea un rectángulo perfecto (su Bounding Box) si esto no genera solapamientos. Esto mantiene las zonas limpias y operables.

Algoritmo - Simulated Annealing

```
Solucion actual = buildInitialSolution(instancia)
VarGlobal = calculateTotalVariance(instancia)

Si !isSolutionValid(actual.Z, VarGlobal):
    Repetir hasta 1000 veces:
        vecino = generateNeighbor(actual)
        Si isSolutionValid(vecino, VarGlobal):
            actual = vecino; break

Calcular energia actual = error + W*penalty
mejor = actual
T = T0

Mientras T > Tf Y iteraciones < max Y tiempo < max_time_seconds:
    Para i = 0 hasta iters_per_temp:
        vecino = generateNeighbor(actual)
        Si NO isPartitionConnected(vecino): Continuar
        Si !makeRectsIfNonOverlapping(vecino): Continuar

        Energia vecino = error + W*penalty (calculateErrorAndVariance + calculateVariancePenalty)
        Delta = Energia vecino - Energia actual
        Si (Delta < 0) O (Random(0,1) < exp(-Delta / T)):
            actual = vecino
            Energia actual = Energia vecino
        Si Energia actual < Energia mejor:
            mejor = actual

    T = T * cooling_factor
```

Resultados: Definición de la Instancia

Para validar el algoritmo, utilizamos una matriz sintética de 5×5 con gradientes de valor claros.

Datos de Entrada (Matriz S) →

Observamos tres regiones distinguibles: valores bajos (8-9) arriba-izq, medios (15-20) y altos (30+) abajo-der.

```
 8.2   9.1  15.3  15.8  14.9
 8.5   9.4  15.6  16.0  15.2
20.1  21.3  22.7  30.2  29.6
19.8  20.5  21.9  31.1  28.4
22.2  22.0  23.5  33.1  32.7
```

Resultados: Definición de la Instancia

Configuración de Ejecución:

- **Zonas (p):** 5
- **Alpha (α):** 0.2 (Restricción estricta de homogeneidad)
- **SA Config:** Default (Enfriamiento geométrico)

```
 8.2   9.1  15.3  15.8  14.9
 8.5   9.4  15.6  16.0  15.2
20.1  21.3  22.7  30.2  29.6
19.8  20.5  21.9  31.1  28.4
22.2  22.0  23.5  33.1  32.7
```

Resultados: Solución Inicial (Heurística)

La solución inicial se genera mediante cortes recursivos (tipo guillotina) buscando maximizar el área, sin considerar los valores del suelo.

Métrica:

- **Error Total (SSE):** `640.287`

Análisis Crítico:

El error es alto porque la Zona 1 (columna izquierda) agrupa valores muy dispares: mezcla el 8.2 (fila 1) con el 22.2 (fila 5). La geometría es correcta, pero la varianza interna es inaceptable.

Resultados: Solución Optimizada

Tras la ejecución del Simulated Annealing, el algoritmo reorganizó los límites buscando homogeneidad.

Métrica:

- **Error Total (SSE):** `24.5767`

Análisis de Mejora:

- **Reducción del error:** ~96% respecto a la inicial.
- **Coherencia:** La Zona 1 ahora captura exclusivamente la esquina superior izquierda (valores 8.2 - 9.4), eliminando la contaminación de valores altos.

Resultados: Solución Optimizada

Métrica:

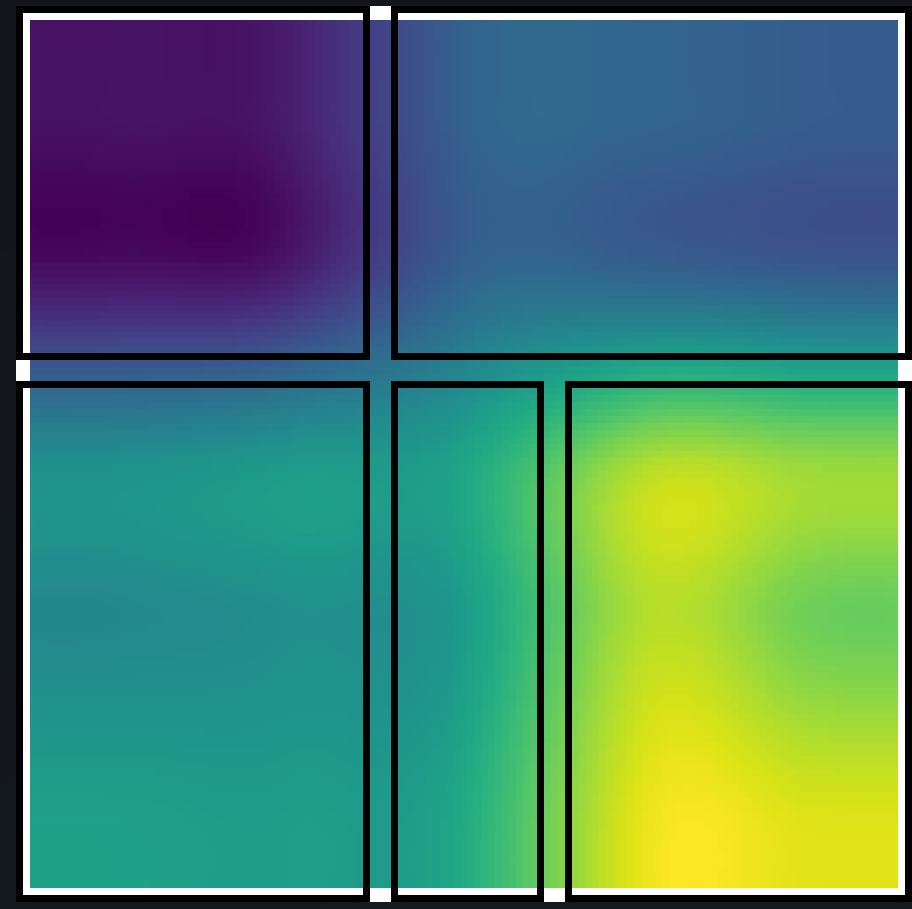
- **Error Total (SSE):** `24.5767`

Análisis de Mejora:

- **Reducción del error:** ~96% respecto a la inicial.
- **Coherencia:** La Zona 1 ahora captura exclusivamente la esquina superior izquierda (valores 8.2 - 9.4), eliminando la contaminación de valores altos.
- **Adaptabilidad:** Las zonas 4 y 5 se ajustaron horizontalmente para segregar el gradiente más fuerte en la parte inferior.

Resultados: Visualización

La segmentación final representada como mapa de calor sobre la matríz.



Interpretación Visual:

- Las líneas representan las fronteras generadas por el algoritmo.
- Se observa una correlación válida entre los colores (valores del suelo) y las regiones delimitadas.
- Se cumple la restricción de **conexidad** y se mantienen formas rectangulares operables, a pesar de la estocasticidad del SA.

Conclusiones y Trabajo Futuro

Proceso de Desarrollo:

1. Diseño de pseudocódigo (foco en estructura de vecindario).
2. Implementación de IO y configuración JSON.
3. Traducción a C++ y refinamiento del movimiento de fronteras para asegurar rectángulos.

Conclusiones:

- El enfoque de penalización permite navegar eficientemente por el espacio de búsqueda sin bloquearse en óptimos locales infactibles.
- La reconstrucción al tras aceptar un movimiento permitió diversificar manteniendo rigida la restricción de la forma.
- La heurística tipo guillotina genera soluciones geométricamente válidas, pero con error alto. El SA reduce drásticamente el error, mostrando que la calidad de la solución inicial influye en cuántas iteraciones y qué T0 se requieren.

Conclusiones y Trabajo Futuro

Posibles Mejoras:

- **Vecindario Avanzado:** Implementar operaciones de *Split & Merge* (dividir una zona y fusionar otras dos) para aumentar aún más la diversificación.
- **Objetivo multi–criterio:** Extender la función de evaluación para balancear homogeneidad, tamaño mínimo/máximo de zonas y regularidad geométrica (por ejemplo, penalizar zonas muy delgadas o muy pequeñas).
- **Algoritmo de Localización:** Implementar en el algoritmo para establecer en el sensor en un punto representativo de la zona, que minimice el error de estimación para los puntos de esa zona.

Referencias

- Huguet, F., Plà-Aragonés, L. M., Albornoz, V. M., & Pohl, M. (2025). **A Genetic Algorithm for Site-Specific Management Zone Delineation**. Mathematics, 13(7), 1064. https://doi.org/10.3390/math13071064
- Torres Herrera, T. B. (2023). **Modelo de Localización de Sensores para la Agricultura de Precisión** (Tesis de Magíster en Ciencias de la Ingeniería). Universidad Técnica Federico Santa María, Chile.
- Cid-García, N. M., Albornoz, V. M., Ortega, R., & Ríos-Solís, Y. A. (2013). **Rectangular shape management zone delineation using integer linear programming**. *Computers and Electronics in Agriculture*, *93*, 1–9. https://doi.org/10.1016/j.compag.2013.01.009