

Presentación

ESTADO DE AVANCE

Problema: 2DSPP

Técnica asignada: Algoritmo Evolutivo (AE)

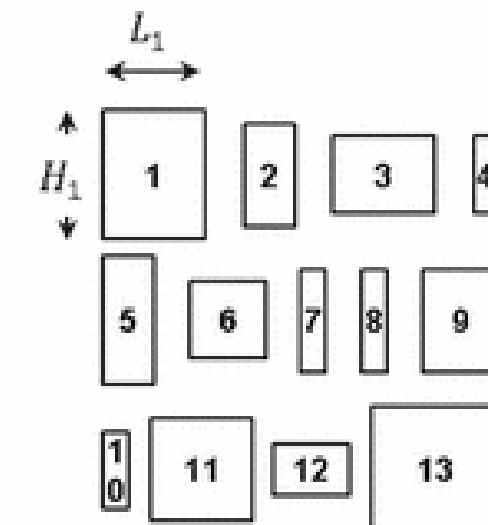
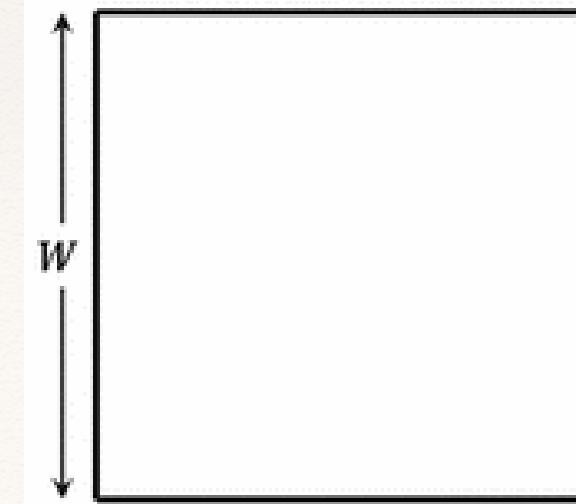


Geraldine Cornejo

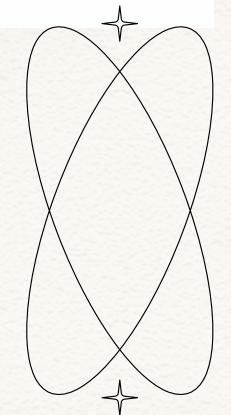
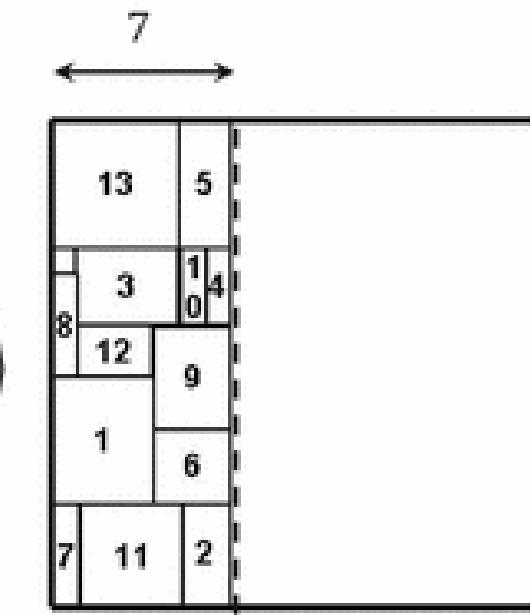
Introducción

- El Problema: Mi proyecto aborda el 2DSPP. Dado un conjunto de rectángulos y una banda de ancho fijo W , el objetivo es encontrar un empaquetamiento que minimice la altura total H .
- La complejidad: Como se detalla en mi informe, este problema es NP-difícil. Esto significa que los métodos exactos no son viables para instancias grandes, y por eso se justifica el uso de una metaheurística como un Algoritmo Evolutivo.

Strip packing problem



Optimal solution



Avance: Implementación mínima

Lectura de Instancia:

```
struct Rectangulo {
    int id;
    int w;
    int h;
};

struct Instancia {
    int W;
    int n;
    vector<Rectangulo> piezas;
};
```

```
Instancia leerInstancia(const string& nombreArchivo) {
    ifstream archivo(nombreArchivo);
    if (!archivo.is_open()) {
        throw runtime_error("Error: No se pudo abrir el archivo " + nombreArchivo);
    }
    Instancia inst;
    archivo >> inst.n;
    archivo >> inst.W;
    for (int i = 0; i < inst.n; ++i) {
        Rectangulo p;
        archivo >> p.id >> p.w >> p.h;
        inst.piezas.push_back(p);
    }
    archivo.close();
    return inst;
}
```

Avance: Implementación mínima

Generación de solución inicial:

```
struct Solucion {
    vector<int> orden_piezas;
    vector<bool> rotaciones;
};

Solucion generarSolucionInicial(const Instancia& inst) {
    Solucion sol;
    sol.orden_piezas.resize(inst.n);
    sol.rotaciones.resize(inst.n);
    iota(sol.orden_piezas.begin(), sol.orden_piezas.end(), 0);
    random_device rd;
    mt19937 g(rd());
    shuffle(sol.orden_piezas.begin(), sol.orden_piezas.end(), g);
    for (int i = 0; i < inst.n; ++i) {
        sol.rotaciones[i] = (rand() % 2 == 0);
    }
    return sol;
}
```

Avance: Implementación mínima

Cálculo de función evaluación:

```
double calcularAlturaTotal(const Solucion& sol, const Instancia& inst) {
    double altura_total_H = 0.0;
    double altura_estante_actual = 0.0;
    double ancho_estante_actual = 0.0;
    for (int i = 0; i < inst.n; ++i) {
        int indicePieza = sol.orden_piezas[i];
        bool rotar = sol.rotaciones[indicePieza];
        const Rectangulo& piezaOriginal = inst.piezas[indicePieza];
        int w_actual = rotar ? piezaOriginal.h : piezaOriginal.w;
        int h_actual = rotar ? piezaOriginal.w : piezaOriginal.h;
        if (ancho_estante_actual + w_actual <= inst.W) {
            ancho_estante_actual += w_actual;
            altura_estante_actual = max(altura_estante_actual, (double)h_actual);
        } else {
            altura_total_H += altura_estante_actual;
            ancho_estante_actual = w_actual;
            altura_estante_actual = h_actual;
        }
    }
    altura_total_H += altura_estante_actual;
    return altura_total_H;
}
```

Avance: Implementación mínima

Programa principal:

```
int main() {
    try {
        // 1. LECTURA DE INSTANCIA
        Instancia instancia = leerInstancia("./CasosDePrueba/BENG10.txt");
        cout << "--- 1. Lectura de Instancia ---" << endl;
        cout << "Instancia 'instancia.txt' leida con exito." << endl;
        cout << "Ancho del Strip (W): " << instancia.W << endl;
        cout << "Número de piezas (n): " << instancia.n << endl;
        cout << "-----" << endl;

        // 2. GENERACIÓN DE SOLUCIÓN INICIAL
        Solucion solInicial = generarSolucionInicial(instancia);
        cout << "--- 2. Solucion Inicial Generada ---" << endl;
        cout << "Orden (índices): ";
        for(int idx : solInicial.orden_piezas) cout << idx << " ";
        cout << endl;
        cout << "Rotaciones (1=SI): ";
        for(bool r : solInicial.rotaciones) cout << r << " ";
        cout << endl;
        cout << "-----" << endl;

        // 3. CÁLCULO DE FUNCIÓN DE EVALUACIÓN
        double alturaObtenida = calcularAlturaTotal(solInicial, instancia);
        cout << "--- 3. Funcion de Evaluacion ---" << endl;
        cout << "La altura (H) de la solucion inicial es: " << alturaObtenida << endl;
        cout << "-----" << endl;

    } catch (const exception& e) {
        cerr << e.what() << endl;
        return 1;
    }
    return 0;
}
```

Diseño Algoritmo Evolutivo



REPRESENTACIÓN

El cromosoma (struct Solucion) tiene dos componentes clave que evolucionan en conjunto:

1. Vector de Orden (Permutación): Un vector<int> que define el orden en que se intentarán colocar las N piezas.
2. Vector de Rotación (Binario): Un vector<bool> que define si la pieza en esa posición está rotada 90° (true) o no (false).

Esta representación garantiza que cada pieza se usa exactamente una vez, manejando una restricción clave por diseño.

Diseño Algoritmo Evolutivo

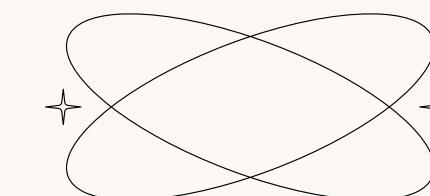
EVALUACIÓN Y SELECCIÓN

Función de evaluación

- El "Cálculo de $f(e)$ " (mi función calcularAlturaTotal) actúa como un Decodificador.
- Toma el cromosoma (orden + rotación).
- "Decodifica" la solución empaquetando las piezas usando la heurística Bottom-Left-Fill (BLF) (o Shelf).
Esto garantiza que no hay solapamiento.
- El valor que devuelve es la Altura H final, que es lo que buscamos minimizar.

Operador de Selección

- Usaré la Selección por Ruleta, como vimos en la materia.
- Dado que mi $f(e)$ minimiza H y la ruleta maximiza, el fitness de un individuo será $1 / H$ o $H_{\max} - H$.



Diseño Algoritmo Evolutivo

OPERADORES DE EVOLUCIÓN

Cruzamiento

- Vector de Orden (Permutación): Se usará Order Crossover (OX). Es ideal para permutaciones porque preserva el orden relativo de los padres.
- Vector de Rotación (Binario): Se usará Cruce Uniforme. Cada bit de rotación se hereda aleatoriamente del Padre 1 o del Padre 2.

Mutación

- Vector de Orden (Permutación): Se usará Swap Mutation. Se eligen dos índices al azar del vector y se intercambian sus piezas (ej. $[... \text{R1} ... \text{R5} ...] \rightarrow [... \text{R5} ... \text{R1} ...]$).
- Vector de Rotación (Binario): Se usará Bit Flip. Se elige un bit de rotación al azar y se invierte (ej. true \rightarrow false).

Elitismo

- Se preservará al mejor individuo de la generación N directamente en la generación $N+1$ para asegurar que la mejor solución encontrada no se pierda.

Problemas y Dudas

Avances del Informe:

- No comprendí la retroalimentación en la sección modelo matemático.
- La corrección principal sobre "Referencias?" fue investigada y corresponde a las citas de (Iori, Martello y Vigo, 2020) y (Cid-García y Ríos-Solís, 2020), que ya están identificadas.

Problemas y Dudas:

1. Duda de Diseño (Representación): Mi diseño "ad-hoc" (Permutación + heurística BLF) es una representación indirecta. ¿Es este el enfoque correcto?
2. Duda de Diseño (Alternativa): La alternativa sería una representación directa (evolucionar las coordenadas x, y del Modelo Continuo de mi informe), pero esto parece mucho más complejo de manejar en el cruzamiento y la mutación. ¿Qué enfoque me recomiendan?
3. Duda de Diseño (Restricciones): Mi $f(e)$ actual garantiza la factibilidad (no-solapamiento) gracias a la heurística. ¿Es esto bueno? ¿O es mejor permitir soluciones infactibles (con solapamiento) y penalizarlas con un costo altísimo en la $f(e)$?

Referencias

- [1] Juan Cid-García and Yasmin Ríos-Solís. A review of the two-dimensional strip packing problem. *Applied Sciences*, 10(3):1–22, 2020.
- [2] Holger Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44(2):145–159, 1990.
- [3] Manuel Iori, Silvano Martello, and Daniele Vigo. An updated overview of algorithms for the two-dimensional bin and strip packing problems. *TOP*, 28(3):339–377, 2020.
- [4] Silvano Martello and Daniele Vigo. Exact solution of the two-dimensional finite bin packing problem. *Management Science*, 44(3):388–399, 1998.
- [5] José Fernando Oliveira, María Antonia Carraville, and Cláudia Ribeiro. A survey of heuristics for the two-dimensional rectangular strip packing problem. *European Journal of Operational Research*, 255(3):757–773, 2016.
- [6] Gerhard Wäscher, Heike Haubner, and Heike Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3):1109–1130, 2007.
- [7] Guntram Wäscher, Heiko Haubner, and Hans Christian Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3):1109–1130, 2007.