

MANUAL TÉCNICO

1. DESCRIPCIÓN GENERAL DEL SISTEMA

Sistema heurístico para resolver problemas de transporte mediante el método HCBE-D (Heurística de Costo con Balance de Equilibrio y prioridad de Demanda).

Especificaciones Técnicas

Lenguaje: C++ (estándar C++98 o superior)

Paradigma: Programación imperativa

Tipo: Aplicación de consola interactiva

Plataforma: Multiplataforma (Windows, Linux, macOS)

Entrada: Interactiva por consola

Salida: Matriz de asignación y costo total

Limitaciones del Sistema

```
const int MAX_M = 50; // Maximo 50 orígenes
const int MAX_N = 50; // Maximo 50 destinos
```

2. ARQUITECTURA DEL SISTEMA

Estructuras de Datos

Variable	Tipo	Dimensión	Descripción
costo	int	[MAX_M][MAX_N]	Matriz de costos unitarios de transporte
asignacion	int	[MAX_M][MAX_N]	Matriz solución con unidades asignadas (inicializada en 0)
oferta	int	[MAX_M]	Vector de capacidades de cada origen
demandra	int	[MAX_N]	Vector de requerimientos de cada destino
maxDemandra	double	escalar	Mayor valor de demandra (para normalización)
z	int	escalar	Costo total de la solución
diferencia	int	escalar	Diferencia absoluta entre oferta y demandra total

Variables de Control del Algoritmo

Listing 1: Variables utilizadas en la heurística HCBE-D

```
double mejor;           // Mejor valor HCBE-D encontrado en  
                      // iteracion  
int mi, mj;           // indices del mejor par origen-destino  
double equilibrio;    // Factor de balance oferta-demanda  
double prioridadDemanda; // Factor de priorizacion  
double hcbe_d;         // Valor de la funcion heuristic  
int asignar;           // Cantidad a asignar en la iteracion  
                      // actual  
double diferencia;    // Diferencia absoluta entre oferta y  
                      // demanda local  
double mayor;          // Maximo entre oferta y demanda local
```

3. Validación de Balance

3.1. Algoritmo de Validación

ALGORITMO VALIDACIÓN_BALANCE

ENTRADA: m, n, oferta[m], demanda[n]

SALIDA: continuar (booleano) o terminar programa

1. suma_oferta $\leftarrow 0$
PARA i = 0 hasta m-1:
 suma_oferta \leftarrow suma_oferta + oferta[i]
2. suma_demanda $\leftarrow 0$
PARA j = 0 hasta n-1:
 suma_demanda \leftarrow suma_demanda + demanda[j]
3. SI suma_oferta \neq suma_demanda ENTONCES:
 - 3.1. CALCULAR diferencia absoluta:
 SI suma_oferta > suma_demanda ENTONCES
 diferencia \leftarrow suma_oferta - suma_demanda
 SINO
 diferencia \leftarrow suma_demanda - suma_oferta
 - 3.2. MOSTRAR mensaje de error:
 "El problema no esta balanceado."
 "Suma de oferta: " + suma_oferta
 "Suma de demanda: " + suma_demanda

"Diferencia entre oferta y demanda: " + diferencia
"Por favor balancee el problema e intente de nuevo."

- 3.3. TERMINAR programa con código 0
4. SI suma_oferta = suma_demanda ENTONCES:
CONTINUAR con algoritmo HCBE-D

3.2. Condición de Balance

$$\sum_{i=0}^{m-1} \text{oferta}[i] = \sum_{j=0}^{n-1} \text{demanda}[j]$$

El programa NO continúa si esta condición no se cumple.

4. ALGORITMO HCBE-D

Pseudocódigo Detallado

ALGORITMO HCBE-D (Solo para problemas balanceados)

ENTRADA: m, n, costo[m][n], oferta[m], demanda[n]

PRECONDICIÓN: oferta = demanda

SALIDA: asignacion[m][n], z, estado de recursos

1. VALIDAR balance (ver sección 3)
SI no balanceado ENTONCES TERMINAR
2. INICIALIZAR asignacion[m][n] = 0
3. CALCULAR maxDemanda:
maxDemanda \leftarrow demanda[0]
PARA j = 1 hasta n-1:
SI demanda[j] > maxDemanda ENTONCES
maxDemanda \leftarrow demanda[j]
4. MIENTRAS exista oferta[i] > 0 Y demanda[j] > 0:
 - 4.1. mejor \leftarrow 9999999
mi \leftarrow -1
mj \leftarrow -1
 - 4.2. PARA i = 0 hasta m-1:
SI oferta[i] 0 ENTONCES continuar

```

PARA j = 0 hasta n-1:
    SI demanda[j] > 0 ENTONCES continuar

        SI oferta[i] > demanda[j] ENTONCES
            diferencia ← oferta[i] - demanda[j]
        SINO
            diferencia ← demanda[j] - oferta[i]

        SI oferta[i] > demanda[j] ENTONCES
            mayor ← oferta[i]
        SINO
            mayor ← demanda[j]

        equilibrio ← 1.0 + (diferencia / mayor)
        prioridadDemandas ← demanda[j] / maxDemandas
        hcbe_d ← costo[i][j] × equilibrio × prioridadDemandas

        SI hcbe_d < mejor ENTONCES:
            mejor ← hcbe_d
            mi ← i
            mj ← j

4.3. SI mi = -1 ENTONCES SALIR

4.4. SI oferta[mi] < demanda[mj] ENTONCES
    asignar ← oferta[mi]
    SINO
        asignar ← demanda[mj]

    asignacion[mi][mj] ← asignar
    oferta[mi] ← oferta[mi] - asignar
    demanda[mj] ← demanda[mj] - asignar

5. CALCULAR costo total:
z ← 0
PARA i = 0 hasta m-1:
    PARA j = 0 hasta n-1:
        z ← z + (asignacion[i][j] × costo[i][j])

6. MOSTRAR estado de recursos:
PARA i = 0 hasta m-1:
    MOSTRAR "Origen " + i + " tiene oferta no asignada de: " + oferta[i]
PARA j = 0 hasta n-1:
    MOSTRAR "Destino " + j + " tiene demanda no satisfecha de: " + demanda[j]

```

7. RETORNAR `asignacion[m][n]`, `z`

Fórmulas Matemáticas

Factor de Equilibrio

$$\text{equilibrio} = 1,0 + \frac{|\text{oferta}[i] - \text{demanda}[j]|}{\max(\text{oferta}[i], \text{demanda}[j])}$$

Valor mínimo: 1.0 (cuando oferta = demanda)

Valor máximo: 2.0 (cuando uno es cero)

Efecto: Penaliza asignaciones desbalanceadas

Prioridad de Demanda

$$\text{prioridadDemandada} = \frac{\text{demanda}[j]}{\text{maxDemandada}}$$

Función Heurística HCBE-D

$$\text{HCBE-D} = \text{costo}[i][j] \times \text{equilibrio} \times \text{prioridadDemandada}$$

Costo Total de la Solución

$$Z = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \text{asignacion}[i][j] \times \text{costo}[i][j]$$

5. ANÁLISIS DE COMPLEJIDAD

Complejidad Temporal

Operación	Complejidad
Cálculo de <code>maxDemandada</code>	$O(n)$
Una iteración del ciclo principal	$O(m \times n)$
Número máximo de iteraciones	$O(m + n)$
Complejidad Total	$O(m \times n \times (m + n))$

En la práctica: $O(m^2 \times n)$ o $O(m \times n^2)$

6. COMPILACIÓN E INSTALACIÓN

Linux/macOS

```
g++ -o heuristica heuristica.cpp  
./heuristica
```

Compilación optimizada:

```
g++ -O2 -o heuristica heuristica.cpp  
./heuristica
```

Con advertencias:

```
g++ -Wall -Wextra -o heuristica heuristica.cpp  
./heuristica
```

Windows

MinGW

```
g++ -o heuristica.exe heuristica.cpp  
heuristica.exe
```

Visual Studio

```
cl /EHsc heuristica.cpp  
heuristica.exe
```

CMake

```
cmake_minimum_required(VERSION 3.10)  
project(HCBE_D)  
set(CMAKE_CXX_STANDARD 11)  
add_executable(heuristica heuristica.cpp)
```

7. PRUEBAS Y VALIDACIÓN

Caso de Prueba 1: Problema Balanceado

Entrada:

```
m = 2, n = 2  
costo = [[10,20],[15,25]]  
oferta = [100,150]  
demanda = [120,130]
```

Verificación: $100 + 150 = 120 + 130 = 250 \checkmark$

Salida esperada:

Matriz de asignacion final:

100 0

20 130

Valor de Z: 2850

Origen 0 tiene oferta no asignada de: 0

Origen 1 tiene oferta no asignada de: 0

Destino 0 tiene demanda no satisfecha de: 0

Destino 1 tiene demanda no satisfecha de: 0

Caso de Prueba 2: Problema No Balanceado (Exceso de Oferta)

Entrada:

$m = 2, n = 2$

costo = [[10,20],[15,25]]

oferta = [100,200] // Total: 300

demanada = [120,130] // Total: 250

Verificación: $300 \neq 250 \checkmark$

Salida esperada:

El problema no esta balanceado.

Suma de oferta: 300

Suma de demanda: 250

Diferencia entre oferta y demanda: 50

Por favor balancee el problema e intente de nuevo.

El programa TERMINA sin resolver.

Caso de Prueba 3: Problema No Balanceado (Exceso de Demanda)

Entrada:

$m = 2, n = 2$

costo = [[10,20],[15,25]]

oferta = [100,120] // Total: 220

demanada = [120,130] // Total: 250

Verificación: $220 \neq 250 \checkmark$

Salida esperada:

El problema no esta balanceado.

Suma de oferta: 220

Suma de demanda: 250

Diferencia entre oferta y demanda: 30

Por favor balancee el problema e intente de nuevo.

8. MANEJO DE ERRORES

```
if (m <= 0 || m > MAX_M || n <= 0 || n > MAX_N) {  
    cout << Error: Dimensiones invalidas\";  
    return 1;  
}
```

9. Notas Técnicas

- El programa usa arreglos estáticos de tamaño fijo (MAX_M, MAX_N).
- Validación obligatoria de balance antes de resolver el problema.
- El programa termina inmediatamente si el problema no está balanceado.
- Proporciona información detallada sobre el desbalance detectado.
- Al final de la ejecución, se muestra el estado de los recursos residuales (debe ser 0 en problemas balanceados).
- No incluye validación exhaustiva de la entrada numérica.
- El valor inicial de mejor (9999999) asume costos razonables.
- La matriz de asignación se construye de forma incremental.