Problema de Transporte Logístico Discreto

Diseño y Análisis de Algoritmos

Richard A. Matos Arderí
Abel Ponce González
Abraham Romero Imbert

Facultad de Matemática y Computación

17 de enero de 2026

Contenidos

Descripción del Problema

Escenario Real:

- **n** paquetes con peso y valor
- **k** vehículos con capacidad máxima
- Distribuir respetando límites de capacidad
- **Objetivo:** Equilibrar la carga

Ejemplo Simple:

- 4 paquetes: 2kg, 3kg, 4kg, 1kg
- 2 camiones: 6kg uno y 5kg otro
- Desafío: distribución equitativa

[ESPACIO PARA ILUSTRACIÓN: Camiones y paquetes]

Entrada del Problema:

- Ítems: Un ítem $i \in I$ se caracteriza por un par (w_i, v_i), $I = \{1, 2, \ldots, n\}$
- Pesos: $w_i > 0$ para cada ítem i
- Valores: $v_i \geq 0$ para cada ítem i
- k contenedores con capacidades $C_1, \ldots, C_k$

Variables de Decisión:

$$x_{ij} = \begin{cases} 1 & \text{si ítem } i \text{ va a contenedor } j \\ 0 & \text{en otro caso} \end{cases}$$

Objetivo: Minimizar: $\max_j V_j - \min_j V_j$
donde $V_j = \sum_{i=1}^{n} v_i \cdot x_{ij}$
Sujeto a: $\forall j \in \{1, \ldots, k\} : \sum_{i:\sigma(i)=j} w_i \leq C_j$

Formulación como Programa Lineal Entero

Planteo ILP:

$$
\begin{aligned}
\text{minimizar} \quad & z^+ - z^- \\
\text{s.a.:} \quad & \sum_{j=1}^{k} x_{ij} = 1, \quad \forall i \\
& \sum_{i=1}^{n} w_i \cdot x_{ij} \leq C_j, \quad \forall j \\
& \sum_{i=1}^{n} v_i \cdot x_{ij} \leq z^+, \quad \forall j \\
& \sum_{i=1}^{n} v_i \cdot x_{ij} \geq z^-, \quad \forall j \\
& x_{ij} \in \{0, 1\}
\end{aligned}
$$

Propiedad: En optimalidad: $z^+ = \text{máx}_j\, V_j$ y $z^- = \text{mín}_j\, V_j$

Problema de Decisión vs Optimización

Problema de Decisión (BALANCED-BIN-PACKING-DEC):

> *Dados n ítems con pesos y valores, k bins con capacidades $C_1, \ldots, C_k$, y un umbral B, ¿existe una asignación factible tal que la diferencia máxima de valores entre bins sea $\leq B$?*

Problema de Optimización (BALANCED-BIN-PACKING-OPT):

> *Dados n ítems con pesos y valores, k bins con capacidades $C_1, \ldots, C_k$, encontrar una asignación factible que minimice la diferencia máxima de valores entre bins.*

Proposición: Si el problema de decisión BALANCED-BIN-PACKING-DEC está en NP, entonces el problema de optimización BALANCED-BIN-PACKING-OPT está en NPO (problemas de optimización NP).

Paso 1: BALANCED-BIN-PACKING-DEC $\in$ NP

Un certificado para una instancia con respuesta 'sí' es una asignación $\sigma : I \rightarrow \{1, \ldots, k\}$. La verificación requiere:

1. Verificar que cada ítem está asignado: $O(n)$
2. Calcular peso total de cada bin: $O(n)$
3. Verificar restricciones de capacidad: $O(k)$
4. Calcular valor total de cada bin: $O(n)$
5. Verificar que $\max_j V_j - \min_j V_j \leq B$: $O(k)$

Total: $O(n + k)$, por lo tanto el certificado es verificable en tiempo polinomial. $\square$

Cadena de Reducciones

$$\text{PARTITION} \leq_p \text{3-PARTITION} \leq_p \text{BALANCED-BIN-PACKING}$$

Implicación: El problema es **NP-Hard**

Resultado: El problema es NP-Completo

No existe algoritmo polinomial conocido (asumiendo P $\neq$ NP)

Se requiere trade-off: **optimalidad vs velocidad**

Métodos Implementados (por tipo)

Exactos (garantizan óptimo):

- Fuerza Bruta
- Branch & Bound
- Programación Dinámica

Greedy / Aproximados (muy rápidos):

- FFD (First Fit Decreasing)
- BFD (Best Fit Decreasing)
- WFD (Worst Fit Decreasing)
- LDF (Largest Difference First)
- LPT Balanced (Longest Processing Time)
- KK (Karmarkar-Karp) para particionamiento
- Round Robin balanceado

Metaheurísticas (búsqueda inteligente):

- Simulated Annealing
- Algoritmos Genéticos
- Búsqueda Tabú

Nota: Greedy prioriza velocidad; metaheurísticas balancean calidad/tiempo; exactos garantizan optimalidad en instancias pequeñas.

Búsqueda Exhaustiva (Fuerza Bruta)

Estrategia: Enumerar y evaluar todas las k^n asignaciones posibles.

Garantías: **Encuentra el óptimo global garantizado**

Límites prácticos (1 segundo):

- $k = 2$: hasta $n = 14$ (16K asignaciones)
- $k = 3$: hasta $n = 11$ (177K asignaciones)
- $k = 4$: hasta $n = 8$ (65K asignaciones)

Complejidad:

- **Tiempo:** $O(k^n \cdot n)$ - Exponencial en ambas dimensiones
- **Espacio:** $O(n + k)$ - Muy eficiente

Aplicabilidad: Validar heurísticas, instancias pequeñas ($n \leq 14$)

Branch and Bound - Búsqueda con Poda

Idea Central: Fuerza Bruta + Cotas Inferiores = Poda Inteligente

Algoritmo:

1. Construir árbol de decisión rama por rama
2. Para cada nodo: calcular **cota inferior** optimista
3. Si cota $\geq$ mejor solución encontrada: **podar rama**
4. Continuar con ramas prometedoras

Mejora respecto Fuerza Bruta:

- Explora **solo subárbol prometedor**
- Cotas ajustadas = mayor poda
- Encontrar óptimo rápido = mejor umbral de poda

Complejidad: $O(k^n)$ peor caso, pero mucho mejor en práctica

Garantías: **Optimalidad + velocidad** en instancias medianas

Programación Dinámica - Construcción Óptima

Estrategia: Construir solución de forma incremental: contenedor por contenedor.

Subestructura Óptima: Si tenemos la mejor asignación a $j-1$ contenedores, podemos encontrar la mejor para j contenedores añadiendo uno nuevo.

Estado DP: $DP[j][mask]$ = mejor configuración de ítems en *mask* usando primeros j contenedores

- j: número de contenedores (1 a k)
- *mask*: subconjunto binario de ítems

Recurrencia: Para cada contenedor j, probar todos los subconjuntos de ítems que le caben

Complejidad:

- **Tiempo:** $O(k^2 \cdot 3^n)$ - Factor 3 por subconjuntos
- **Espacio:** $O(k \cdot 2^n)$ - Tabla completa

Límites: Instancias pequeñas ($n \leq 15$)

Algoritmos Greedy - Primera Parte

Estrategia común: Ordena ítems y asigna cada uno usando una regla local.

First Fit Decreasing (FFD):

- Ordena por peso decreciente
- Asigna al **primer contenedor** con espacio disponible
- Tiempo: $O(n \log n + n \cdot k)$ — Gap: 5-15 %

Best Fit Decreasing (BFD):

- Ordena por valor decreciente
- Asigna al contenedor que **minimiza diferencia** resultante
- Tiempo: $O(n \log n + n \cdot k)$ — Gap: 4-12 %

Worst Fit Decreasing (WFD):

- Ordena por valor decreciente
- Asigna al contenedor con **mayor espacio libre**
- Distribución uniforme — Tiempo: $O(n \log n + n \log k)$ — Gap: 5-15 %

LPT Balanced:

- **Longest Processing Time** para balance
- Asigna al contenedor con **menor valor** actual
- Muy rápido — Tiempo: $O(n \log n + n \log k)$ — Gap: 3-10 %

Algoritmos Greedy y Aproximación - Segunda Parte

Round Robin Greedy:

- Distribución secuencial por **mínimo valor actual**
- Tiempo: $O(n \log n + n \log k)$ — Gap: 4-14 %

Largest Difference First (LDF):

- En cada paso, **evalúa todas combinaciones** (ítem, contenedor)
- Elige la que **minimiza diferencia** resultante
- Más lento pero mejor balance
- Tiempo: $O(n^2 \cdot k)$ — Gap: 2-8 %

Karmarkar-Karp (KK) - Particionamiento:

- Algoritmo clásico de **diferenciación**
- Combina valores: reemplaza dos máximos por su diferencia
- Excelente para 2 contenedores
- Tiempo: $O(n \log n)$ — Gap: 2-6 %

Resumen Greedy:

- **Ventaja:** Muy rápidos, $O(n \log n)$
- **Desventaja:** Gap 2-15 %, sin garantías teóricas
- **Uso:** Baseline rápido, problemas grandes

Metaheurísticas

Simulated Annealing, Algoritmos Genéticos, Búsqueda Tabú

Esquema general:

1. Generar solución inicial
2. Mejorar iterativamente con movimientos locales
3. Aceptar movimientos malos ocasionalmente
4. Detener por convergencia o tiempo

Ventajas: Rápidas, buenos resultados
Desventajas: Sin garantías, aleatoriedad, parámetros

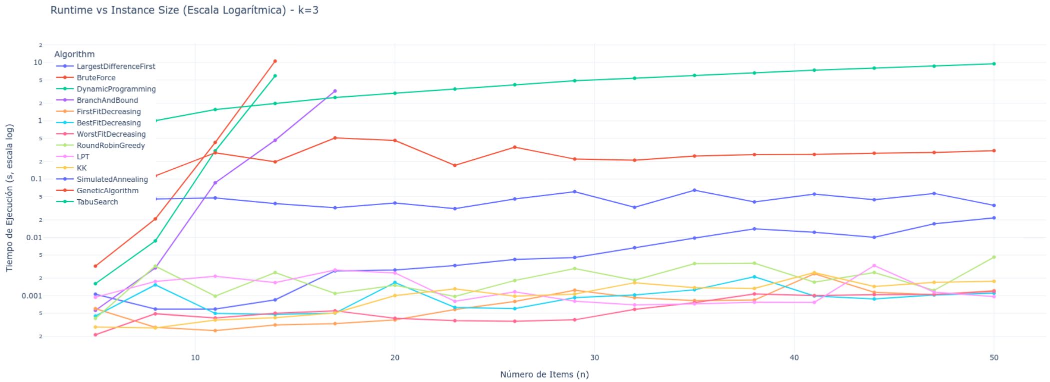
Análisis de Complejidad - Comparativa

Complejidad temporal y espacial de los algoritmos implementados

Algoritmo	Tiempo	Espacio
Fuerza Bruta	$O(k^n \cdot n)$	$O(n + k)$
FFD	$O(n \log n + n \cdot k)$	$O(n + k)$
BFD	$O(n \log n + n \cdot k)$	$O(n + k)$
WFD	$O(n \log n + n \log k)$	$O(n + k)$
LPT Balanced	$O(n \log n + n \log k)$	$O(n + k)$
Round Robin	$O(n \log n + n \log k)$	$O(n + k)$
LDF	$O(n^2 \cdot k)$	$O(n + k)$
KK (Karmarkar-Karp)	$O(n \log n)$	$O(n)$
Prog. Dinámica	$O(k^2 \cdot 3^n)$	$O(k \cdot 2^n)$
Branch & Bound	$O(k^n)$ peor caso	$O(n \cdot k)$
Simulated Annealing	$O(I \cdot n)$	$O(n)$
Genetic Algorithm	$O(G \cdot P \cdot n)$	$O(P \cdot n)$
Tabu Search	$O(I \cdot N)$	$O(n + T)$

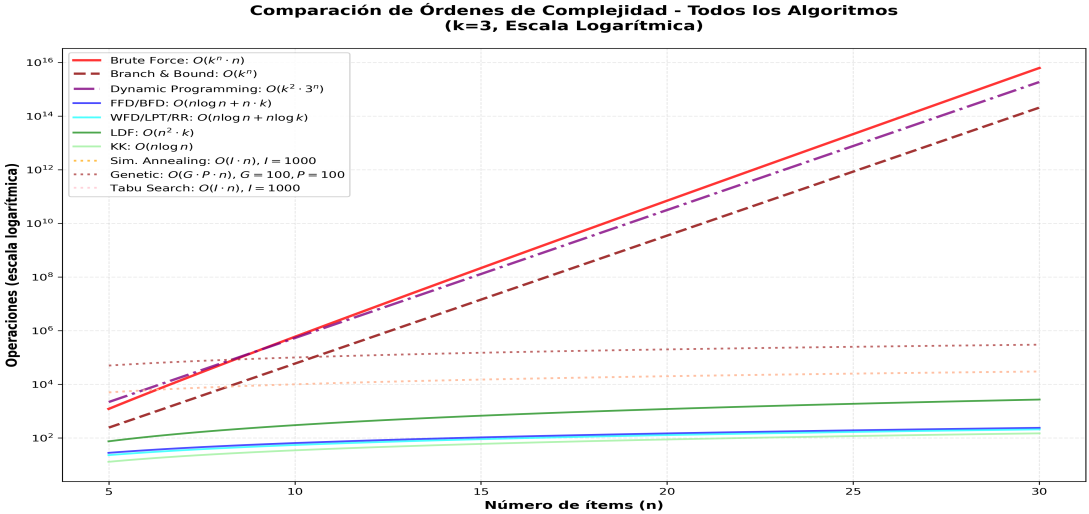
Leyenda: n=ítems, k=contenedores, I=iteraciones, G=generaciones, P=población, N=vecindario, T=lista tabú

Comparación de Tiempos



Instancias aleatorias con $n \in [5, 50]$ e $k = 3$
Eje Y: tiempo (segundos, escala logarítmica)
Eje X: número de ítems

Órdenes de Complejidad

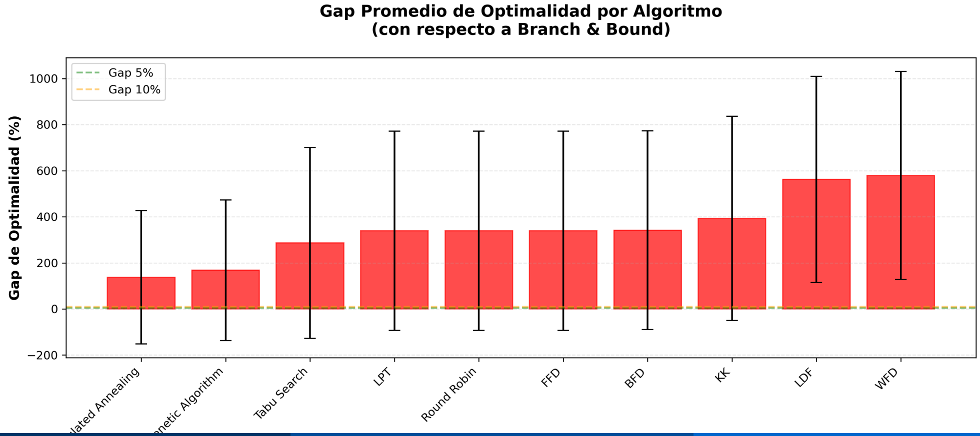


Comparación teórica de órdenes de complejidad con $k = 3$ contenedores

Calidad de Soluciones

Gap de Optimalidad:

$$\text{Gap}(\%) = \frac{\text{Heurístico} - \text{Óptimo}}{\text{Óptimo}} \times 100$$



Resumen Comparativo

Algoritmo	Óptimo	Velocidad	Rango	Aplicación
Métodos Exactos (Garantizan optimalidad)				
Fuerza Bruta	Sí	Muy lenta	$n < 15$	Validación, instancias mínimas
Branch & Bound	Sí	Media	$n < 25$	Instancias medianas, mejor relación
Prog. Dinámica	Sí	Media	$n < 20$	Instancias pequeñas con subproblemas
Métodos Aproximados/Greedy (Rápidos, sin garantía)				
FFD (First Fit Dec.)	No	Muy rápida	Todos	Baseline rápido, gap 5-15 %
BFD (Best Fit Dec.)	No	Muy rápida	Todos	Equilibrio simple, gap 4-12 %
WFD (Worst Fit Dec.)	No	Muy rápida	Todos	Distribución balanceada, gap 5-15 %
LDF (Largest Diff. First)	No	Rápida	Todos	Balance óptimo, gap 2-8 %
LPT Balanced	No	Muy rápida	Todos	Greedy mejorado, gap 3-10 %
KK (Karmarkar-Karp)	No	Muy rápida	Todos	Particionamiento, gap 2-6 %
Round Robin	No	Muy rápida	Todos	Distribución uniforme, gap 4-14 %
Metaheurísticas (Búsqueda inteligente, mejor calidad)				
Simulated Annealing	No	Rápida	Todos	Escape de mínimos locales, gap 0.5-2 %
Algoritmos Genéticos	No	Rápida	Todos	Problemas complejos, gap 0.5-3 %
Búsqueda Tabú	No	Rápida	Todos	Mejor calidad promedio, gap 0.2-1.5 %

Conclusiones

1. **NP-Hard:** No existe algoritmo polinomial (bajo P $\neq$ NP)
2. **Trade-off:** Optimalidad vs velocidad

Elección depende del contexto:

- Instancias pequeñas: B&B o DP
- Problemas reales: Metaheurísticas
- Baseline rápido: Greedy (FFD, LPT, KK, LDF)

3. **Herramientas:** 14 algoritmos, dashboard interactivo, benchmarking
4. **Aplicaciones:** Logística, distribución de carga, computación distribuida

Agradecimientos

Gracias por su atención

Preguntas y Discusión

Código disponible en GitHub