

Programación Entera y Combinatorial: Descomposición y Heurísticas

IN770

Universidad de Chile, DII

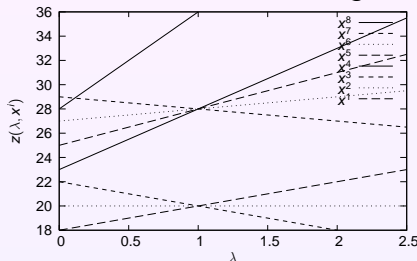
7 de marzo de 2011

Contenidos

- 1 Descomposición para MIP
- 2 Heurísticas en Optimización Combinatorial:
- 3 Algunos ejemplos en detalle

Un Ejemplo

- Consideremos el siguiente Ejemplo:



- $\max\{7x_1 + 2x_2 : -x_1 + 2x_2 \leq 4, x \in Q\}.$
- $Q = \{x \in \mathbb{Z}_+^2 : 5x_1 + x_2 \leq 20, -2x_1 - 2x_2 \leq -7, x_1 \geq 2, x_2 \leq 4\}.$

- $Q = \{x^i : i = 1, \dots, 8\} = \{(2, 2), (2, 3), (2, 4), (3, 1), (3, 2), (3, 3), (3, 4), (4, 0)\}.$
- $z_{LR}(\lambda) = \max\{7x_1 + 2x_2 + \lambda(4 + x_1 - 2x_2) : x \in Q\}.$

Relajación Lagrangeana

- Consideramos $IP(Q) : z_{IP} = \max\{cx : x \in S\}$ donde $S = \{x : A^1x \leq b^1, x \in Q\}$.
 - Podemos interpretar λ como *penalizaciones* en la función objetivo por violar $A^1x \leq b^1$, de hecho, podemos tomar λ tal que aseguremos $A^1x \leq b^1$.
 - $LR(\lambda)$ es una relajación (en dominio y función objetivo) de $IP(Q)$ (i.e. $z_{IP} \leq z_{LR}(\lambda)$).
 - Definimos $LD : z_{LD} = \min\{z_{LR}(\lambda) : \lambda \in \mathbb{R}_+^{m_1}\}$, note que $z_{IP} \leq z_{LD}$, LD es el dual lagrangeano de $IP(Q)$.
 - Claramente $z_{LR}(\lambda) = \max\{z(\lambda, x) : x \in \text{conv.hull}(Q)\}$.
 - Pero también, si $\text{conv.hull}(Q) = \text{conv.hull}\{x^i\}_{i=1}^m$, entonces $z_{LR}(\lambda) = \max\{z(\lambda, x^i) : i \in \{1, \dots, m\}\}$.
 - Como $z(\lambda, x^i) = cx^i + \lambda(b^1 - A^1x^i)$ es lineal afín, entonces $z_{LR}(\lambda)$ es una función convexa lineal por tramos.

Interpretando z_{LD}

$$z_{LD} = \max\{cx : A^1x \leq b^1, x \in \text{conv.hull}(Q)\}$$

$$z_{LR}(\lambda) = \max\{(c - \lambda A^1)x + \lambda b^1 : x \in Q\}$$

$$= \max\{cx + \lambda(b^1 - A^1x) : x \in \text{conv.hull}(Q)\}$$

$$\Rightarrow z_{LD} = \min\{\max\{cx + \lambda(b^1 - A^1x) : x \in \text{conv.hull}(Q)\} : \lambda \geq 0\}$$

$$Q \neq \emptyset \Rightarrow = \min \left\{ \begin{array}{l} \max\{cx^k + \lambda(b^1 - A^1x^k) : k \in K\} : \\ (c - \lambda A^1)r^j \leq 0, j \in J, \lambda \geq 0 \end{array} \right\}$$

$$= \min_{\lambda \geq 0, \eta} \left\{ \eta : \begin{array}{ll} \eta + \lambda(A^1x^k - b^1) \geq cx^k & k \in K \\ \lambda A^1r^j \geq cr^j & j \in J \end{array} \right\}$$

$$\max \quad c \left(\sum (\alpha^k x^k : k \in K) + \sum (\beta^j r^j : j \in J) \right)$$

$$\begin{array}{ll} \text{dual } \Rightarrow & = \quad \text{s.t.} \quad \sum (\alpha^k : k \in K) = 1 \\ & \quad A^1 \left(\sum (\alpha^k x^k : k \in K) + \sum (\beta^j r^j : j \in J) \right) \leq b^1 \\ & \quad \alpha^k, \beta^j \geq 0 \forall j \in J, k \in K \end{array}$$

$$= \max\{cx : A^1x \leq b^1, x \in \text{conv.hull}(Q)\}.$$

Interpretando z_{LD}

Calculado z_{LD}

Usando

$$\min_{\lambda \geq 0, \eta} \left\{ \eta : \begin{array}{ll} \eta + \lambda(A^1 x^k - b^1) \geq cx^k & k \in K \\ \lambda A^1 r^j \geq cr^j & j \in J \end{array} \right\}$$

o

$$\begin{array}{ll} \text{máx} & c \left(\sum(\alpha^k x^k : k \in K) + \sum(\beta^j r^j : j \in J) \right) \\ \text{s.t.} & \sum(\alpha^k : k \in K) = 1 \\ & A^1 \left(\sum(\alpha^k x^k : k \in K) + \sum(\beta^j r^j : j \in J) \right) \leq b^1 \\ & \alpha^k, \beta^j \geq 0 \forall j \in J, k \in K \end{array}$$

podemos calcular (o acotar) z_{LD} por generación de columnas/restricciones en los problemas respectivos.

Interpretando z_{LD}

Theorem

$z_{LD} = z_{IP}$ para todo c si y solo si
 $\text{conv.hull}(Q \cap \{x \in \mathbb{R}_+^n : A^1 x \leq b^1\}) = \text{conv.hull}(Q) \cap \{x \in \mathbb{R}_+^n : A^1 x \leq b^1\}.$

Theorem (Propiedad de integralidad)

$z_{LD} = z_{LP}$ para todo c si y solo si los puntos extremos de $\{x \in \mathbb{R}_+^n : A^2 x \leq b^2\}$ son enteros.

En General:

Dado que $\text{conv.hull}(S) \subseteq \text{conv.hull}(Q) \cap \{x \in \mathbb{R}_+^n : A^1 x \leq b^1\} \subseteq \{x \in \mathbb{R}_+^n : Ax \leq b\}$ se tiene que $z_{IP} \leq z_{LD} \leq z_{LP}.$

Problema de Flujo con restricción de Capital

- Consideramos

$$\begin{aligned}
 &\text{máx} && \sum_{i,j \in N} c_{ij} x_{ij} \\
 &\text{s.t.} && \sum_{i \in N} x_{ij} = 1 \quad \forall j \in N \quad (1) \\
 &&& \sum_{j \in N} x_{ij} = 1 \quad \forall i \in N \quad (2) \\
 &&& \sum_{i,j \in N} t_{ij} x_{ij} \leq b \quad (3) \\
 &&& x \in \{0, 1\}^{n^2}
 \end{aligned}$$

- Podemos interpretar el problema como asignar n trabajadores a n trabajos bajo restricción de capital.
- EL problema es \mathcal{NP} -completo

Problema de Flujo con restricción de Capital

- Tenemos múltiples opciones:
 - Relajando (3), entonces $LR_1(\lambda)$ es un problema de asignación (\Rightarrow entero) con objetivo $\lambda b + \sum (x_{ij}(c_{ij} - \lambda t_{ij}) : i, j \in N)$.
 - Relajando (1) y (2), entonces $LR_2(u, v)$ es un knapsack con función objetivo $\sum (u_i + v_i : i \in N) + \sum (x_{ij}(c_{ij} - u_i - v_j) : i, j \in N)$.
 - Relajando (1), entonces $LR_3(u)$ es un knapsack con restricciones GUB y objetivo $\sum (u_i : i \in N) + \sum (x_{ij}(c_{ij} - u_i) : i, j \in N)$.
 - Relajando (1) y (3) sólo nos quedan restricciones GUB (\Rightarrow entero), entonces $LR_4(u, \lambda)$ tiene objetivo $\lambda b + \sum (u_i : i \in N) + \sum (x_{ij}(c_{ij} - u_i - \lambda t_{ij}) : i, j \in N)$.
 - Es fácil ver que $z_{IP} \leq z_{LD}^3 \leq z_{LD}^2 \leq z_{LD}^1 = z_{LD}^4 = z_{LP}$.

Cost Splitting Dual

Consideramos (RIP)

$$z_{IP} = \max cx^1$$

$$A^1 x^1 \leq b^1$$

$$A^2 x^2 \leq b^2$$

$$x^1 - x^2 = 0$$

$$x^1, x^2 \in \mathbb{Z}_+^n$$

La relajación (CSD):

Considerando $x^1 - x^2 = 0$ la restricción a relajar, obtenemos:

$$z_{CSD} = \min_{c^1 + c^2 = c} \max_{x^1, x^2} c^1 x^1 + c^2 x^2$$

$$A^1 x^1 \leq b^1$$

$$A^2 x^2 \leq b^2$$

$$x^1, x^2 \in \mathbb{Z}_+^n$$

Cost Splitting Dual

Calidad de la relajación:

$$z_{CSD} = \max \left\{ cx : x \in \begin{array}{l} \text{conv.hull}\{x \in \mathbb{Z}_+^n : A^1x \leq b^1\} \cap \\ \text{conv.hull}\{x \in \mathbb{Z}_+^n : A^2x \leq b^2\} \end{array} \right\}$$

- Esta técnica se llama separación de costos.
- Es especialmente útil cuando:
 - $\text{conv.hull}\{x \in \mathbb{Z}_+^n : A^1x \leq b^1\} \subsetneq \text{conv.hull}\{x \in \mathbb{R}_+^n : A^1x \leq b^1\}$; en estos casos $z_{CSD} < z_{LD}$ para alguna función objetivo.
 - Los conjuntos de restricciones $A^i x \leq b^i$ son *simples*, y la dificultad viene dada por su interacción.
 - En el ejemplo anterior, podemos tomar $A^1x \leq b^1$ como las restricciones (1) y (3), y $A^2x \leq b^2$ como las restricciones (2) y (3), y obtenemos $z_{CSD} < z_{LD}^3$.

Surrugate Dual:

- Consideramos

$$IP(Q) : z_{IP} = \max\{cx : A^1x \leq b^1, x \in Q\}.$$

- Dado $\lambda \in \mathbb{R}_+^n$, definimos
$$SD(\lambda) : z_{SD}(\lambda) = \max\{cx : \lambda A^1x \leq \lambda b^1, x \in Q\}$$
- La idea es reducir las restricciones difíciles a una sola restricción.
- Definimos el *surrogate dual* de $IP(Q)$ como
$$z_{SD} = \min_{\lambda \geq 0} z_{SD}(\lambda).$$
- Notemos que $LR(\lambda)$ es una relajación de $SD(\lambda)$, y
$$z_{SD} \leq z_{LD}.$$
- SD puede ser usado en la práctica en forma computacional, sin embargo, no tiene las propiedades teóricas del dual lagrangeano.

Reformulación de Bender

- Hemos visto como lidiar con restricciones difíciles, ahora veremos la noción de manejar *variables difíciles*.

- $$\begin{aligned} & \text{máx } cx + hy \\ & \text{Consideramos: } (MIP) : z_{MIP} = \begin{aligned} & Ax + Gy \leq b \\ & x \in X \subseteq \mathbb{Z}_+^n, y \in \mathbb{R}_+^p \end{aligned} \end{aligned}$$
- Podemos ver las variables enteras x como *difíciles*, pues transforman el problema de LP a MIP, o las variables y pues transforman un problema entero puro a uno mixto.
 - Consideremos que x ha sido fijado, entonces $LP(x) : z_{LP}(x) = \text{máx}\{hy : Gy \leq b - Ax, y \in \mathbb{R}_+^p\}.$
 - Su dual es $\text{mín}\{u(b - Ax) : uG \geq h, u \in \mathbb{R}_+^m\}.$

Reformulación de Bender

- Caracterizando $z_{LP}(x)$:
 - Consideremos $\{u^k : k \in K\}$ los puntos extremos de $Q = \{u \in \mathbb{R}_+^m : uG \geq h\}$, y $\{v^j : j \in J\}$ los rayos extremos de $\{u \in \mathbb{R}_+^m : uG \geq 0\}$.
 - Note que si $Q \neq \emptyset$, entonces $\{v^j : j \in J\}$ son los rayos extremos de Q .
 - entonces $z_{LP}(x) = \min_{k \in K} u^k(b - Ax)$ si $v^j(b - Ax) \geq 0, \forall j \in J$, y $z_{LP}(x) = -\infty$ en cualquier otro caso.
 - Entonces, si $Q \neq \emptyset$ podemos reformular (MIP) como:

$$z = \max_x \left\{ \begin{array}{l} cx + \min_{k \in K} u^k(b - Ax) \\ \text{s.t.} \quad v^j(b - Ax) \geq 0 \\ x \in X \end{array} \right\}$$

Reformulación de Bender

- Reformulando (MIP):
 - El problema anterior puede re-escribirse como (MIP'):

$$z = \max_{\eta, x} \left\{ \begin{array}{ll} \eta & \\ \text{s.t.} & \eta \leq cx + u^k(b - Ax) \quad \forall k \in K \\ & v^j(b - Ax) \geq 0 \quad \forall j \in J \\ & x \in X, \eta \in \mathbb{R} \end{array} \right\}$$

- Típicamente, (MIP') tiene una enorme cantidad de restricciones.
- Una relajación común es considerar un sub-conjunto de puntos y rayos extremos de Q .
- Podemos hacer *generación de restricciones* en forma dinámica.

Descomposición de Dantzig-Wolfe

- El enfoque anterior descompone algunas variables en forma *dual*, veremos ahora que también podemos aplicar estas ideas a un enfoque *primal*.
 - Consideramos

$$(MIP) : \quad z_{MIP} = \begin{array}{ll} \text{máx} & cx \\ \text{s.t.} & A^1 x \leq b^1 \\ & A^2 x \leq b^2 \\ & x_i \in \mathbb{Z}, \forall i \in I \subseteq N \end{array}$$

- Considerando $Q = \{x : A^2 x \leq b^2, x_i \in \mathbb{Z}, \forall i \in I \subseteq N\} \neq \emptyset$, y su descomposición en puntos extremos $\{v^i : i \in V\}$ y rayos extremos $\{r^i : i \in R\}$, entonces podemos re-escribir (MIP) como sigue:

Descomposición de Dantzig-Wolfe

- Descomponiendo (MIP):
 - Con esta descomposición obtenemos:

$$\begin{aligned}
 &\text{máx} \quad \sum_{i \in V} cv^i \lambda_i + \sum_{i \in R} cr^i \eta_i \\
 &\text{s.t.} \quad \sum_{i \in V} Av^i \lambda_i + \sum_{i \in R} Ar^i \eta_i \leq b^1 \\
 &(\text{MIP}') : \quad \sum_{i \in V} \lambda_i = 1 \\
 &\quad \lambda, \eta \geq 0 \\
 &\quad \left(\sum_{i \in V} v^i \lambda_i + \sum_{i \in R} r^i \eta_i \right)_j \in \mathbb{Z}, \forall j \in I \subseteq N
 \end{aligned}$$

- Nótese que ahora la maximización es en términos de λ, η .
- Número de variables crece (potencialmente) en forma exponencial.

Descomposición de Dantzig-Wolfe

- ¿Qué ganamos con esta descomposición?
 - En general $z_{LP'} \leq z_{LP}$, ¿por que?
 - $z_{LP'} = \max\{cx : A^1x \leq b^1, x \in \text{conv.hull}(Q)\}$ y
 $z_{LP} = \max\{cx : A^i x \leq b^i, i = 1, 2\}.$
 - Podemos usar como relajación el resolver (MIP') sobre un sub-conjunto $V' \subseteq V, R' \subseteq R$, y obtener soluciones heurísticas.
 - ¿Podemos dar una cota de la calidad de tales soluciones?
 - La respuesta corta: Si, pero ¿cómo?
 - Por argumentos de costos reducidos.

Generación de Columnas

- Por simplicidad supondremos que Q es un politopo.
 - Con eso (MIP') puede escribirse como:

$$\begin{aligned}
 MIP'(V) : \quad & \text{máx} \quad \sum_{i \in V} cv^i \lambda_i \\
 & \text{s.t.} \quad \sum_{i \in V} Av^i \lambda_i \leq b^1 \\
 & \quad \sum_{i \in V} \lambda_i = 1, \lambda \geq 0 \\
 & \quad \left(\sum_{i \in V} v^i \lambda_i \right)_j \in \mathbb{Z}, \forall j \in I \subseteq N
 \end{aligned}$$

- Definimos $LP(V)$ como su relajación lineal.
- Consideremos una solución óptima a $LP(V')$ para $V' \subseteq V$, sea π una solución dual óptima, $N \subseteq V'$ el conjunto de variables no básicas.

Generación de Columnas

- Reescribiendo $LP(V')$.
 - Con lo anterior, nos queda:

$$\begin{aligned} \text{máx} \quad & \sum_{i \in N} (cv^i - \pi_o - \pi_A Av^i) \lambda_i \\ LP(V') : \quad & \text{s.t.} \quad \sum_{i \in V'} Av^i \lambda_i \leq b^1 \quad (\pi_A) \\ & \sum_{i \in V'} \lambda_i = 1, \lambda \geq 0 \quad (\pi_o) \end{aligned}$$

- Optimalidad implica que $(cv^i - \pi_o - \pi_A Av^i) \leq 0 \forall i \in N$.
- ¿Qué implica que $(cv^i - \pi_o - \pi_A Av^i) \leq 0$ para $i \in V \setminus V'$?
 - $z_{LP}(V') = z_{LP}(V)$.
- ¿Qué implica que $(cv^i - \pi_o - \pi_A Av^i) \leq \delta$ para $i \in V \setminus V'$?
 - $z_{LP}(V') + \max\{0, \delta\} \leq z_{LP}(V)$.

Generación de Columnas

- Mejorando $z_{LP}(V')$:
 - Supongamos $V' \subsetneq V$, y una solución dual óptima (π_o, π_A) a $LP(V')$.
 - Consideremos \bar{x} una solución de costo positivo a $-\pi_o + \max\{(c - \pi_A A)x : x \in Q\}$ y \bar{w} una cota dual del mismo problema.
 - $z_{LP}(V') + \bar{w} \leq z_{LP}(V)$.
 - Si ampliamos V' a $V' \cup \{\bar{x}\}$, obtenemos una mejor aproximación de $z_{LP}(V)$.
 - Cualquier solución factible a $MIP'(V')$ es una solución factible de MIP' .
 - Este procedimiento es usado en la práctica en una serie de situaciones.

Generación de Columnas

Cutting-stock

Consideramos el problema de satisfacer demanda de distintos items que se fabrican de un solo componente.

Ruteo de Vehículos

Dada una serie de clientes a visitar, debemos decidir que clientes serán atendidos por los vehículos disponibles en nuestra flota.

¿Por qué heurísticas?

- Cotas inferiores (soluciones factibles) son importantes para que algoritmos como B&B terminen rápidamente.
- Desde el punto de vista práctico, lo que se espera es obtener *buenas* soluciones en un tiempo corto.
- Cómo definimos *buenas*?
 - En trabajos científicos, se necesita una garantía de la calidad de las soluciones.
 - Puede ser que la garantía sea en probabilidad (Algoritmo de Karger).
 - En la práctica, basta con que sea mejor que lo que exista de momento.
 - Ojo, eso INCLUYE el obtener soluciones comparables pero en mucho menos tiempo.

¿Por qué heurísticas?

- En general los problemas de optimización combinatorial son \mathcal{NP} -completos, lo que induce a pensar que obtener soluciones óptimas (en forma consistente) es imposible.
- De ahí el interés de tener esquemas que encuentren soluciones casi óptimas.
- Las heurísticas se caracterizan en dos grandes categorías: Constructivas y de Búsqueda Local.
- La historia de algoritmos de búsqueda local en IP data de fines del 1950, con las primeras heurísticas de cambio de arcos para el TSP (2-Opt).
- Hoy es de uso común en Scheduling, Partición de grafos, diseño de layout, ruteo de vehículos, etc.

Situación Actual

- Muchas de las estrategias se basan en procesos *naturales*, como neural networks, simulated annealing, genetic algorithms, ant colonies, etc. Lo que ha contribuido a su popularidad.
- Algunas de las estrategias modernas han permitido demostrar propiedades teóricas de convergencia.
- Desde el punto de vista práctico, la disponibilidad de poder de cálculo barato, uso de estructura de datos avanzados y otros refinamientos han permitido enfrentar problemas de gran escala (TSP en 1.000.000 de ciudades).
- Simplicidad de las ideas básicas ha contribuido a su uso en general.

Definiciones básicas:

Problema de optimización combinatorial

Es especificado por un conjunto de instancias, con una función objetivo a minimizar (maximizar), donde parte o todas las variables tienen una naturaleza discreta.

Instancia de un problema de OC

Una instancia de un problema particular se especifica como un par (\mathcal{L}, f) , donde \mathcal{L} es el conjunto de soluciones factibles, y $f : \mathcal{L} \rightarrow \mathbb{R}$ es la función de costo. El objetivo es encontrar $i^* \in \mathcal{L}$ tal que $f(i^0) \leq f(i)$, $\forall i \in \mathcal{L}$. $f^* = f(i^*)$ es el valor óptimo para la instancia, y $\mathcal{L}^* = \{i \in \mathcal{L} : f(i) = f^*\}$ es el conjunto de soluciones óptimas.

Definiciones básicas:

- Típicamente, \mathcal{L} no es dado en forma explícita (i.e. como conjunto de soluciones y valores objetivos), si no que se conoce un algoritmo polinomial que determina si un punto dado x pertenece a \mathcal{L} , y un algoritmo para computar $f(x)$.
- Las soluciones se representan con una serie de *variables de decisión*, restringidas a ciertos rangos.
- Un ejemplo de estas variables esta dado por los problemas de programación entera.
- Usualmente estas variables nos permiten definir *vecindades* de soluciones.

Definiciones básicas:

Vecindades

Dada una instancia (\mathcal{L}, f) de un problema de OC, una función de vecindades es una función $\mathcal{N} : \mathcal{L} \rightarrow 2^{\mathcal{L}}$. La idea básica es que a cada $i \in \mathcal{L}$ $\mathcal{N}(i) \subseteq \mathcal{L}$ define un conjunto de soluciones *cercanas* a i . En general se asume que $i \in \mathcal{N}(i)$.

Óptimo local

Dada una instancia (\mathcal{L}, f) de un problema de OC, y una función de vecindades \mathcal{N} , se dice que i es un óptimo local si $f(i) \leq f(j)$, $\forall j \in \mathcal{N}(i)$. Definimos $\hat{\mathcal{L}}$ como el conjunto de óptimos locales.

Definiciones básicas:

Algoritmo básico de LS:

Require: $t \leftarrow 0, i^t \in \mathcal{L}$

1: **repeat**

2: $t \leftarrow t + 1$.

3: **if** $\exists j \in \mathcal{N}(i^{t-1}) : f(j) < f(i^{t-1})$ **then**

4: $i^t \leftarrow j$.

5: **else**

6: $i^t \leftarrow i^{t-1}$.

7: **until** $f(i^{t-1}) > f(i^t)$

8: **return** $i^t, f(i^t)$

► Extensiones

Vecindades exactas:

Dada una instancia (\mathcal{L}, f) de un problema de OC, y una vecindad \mathcal{N} , \mathcal{N} es exacta si $\hat{\mathcal{L}} = \mathcal{L}^*$ (simplex?).

Definiciones básicas:

First Improvement

Si el paso 4 del algoritmo LS, se toma el primer j que satisface la condición, a esto se le llama LS con first improvement.

Best Improvement

Si el paso 4 del algoritmo LS, se toma el mejor j que satisface la condición, a esto se le llama LS con best improvement.

- Algoritmos de búsqueda local con vecindades exactas son algoritmos de optimización.
- LS puede ser vista como una *caminata* en un grafo dirigido.

Introducción:

- Definir vecindades que generen soluciones de alta calidad es una de las problemáticas centrales en LS.
- Claramente deben depender del problema (y las variables de decisión asociadas) en cuestión (TSP visto como permutación y visto como secuencia de arcos).
- No existen reglas generales de construcción de vecindades.
- Se busca que los grafos inducidos por las vecindades sean fuertemente conexos, i.e. $\forall i, j \in \mathcal{L}$ existe un camino dirigido de i a j .

Vecindades de intercambio

- Representación por secuencias/particiones.
 - Idea: intercambiar elementos en orden o particiones.
 - Se define vecindades de k -intercambio para $i \in \mathcal{L}$, como las soluciones $j \in \mathcal{L}$ que pueden ser obtenidas a partir de i usando k cambios de orden o k cambios de elementos entre las particiones.
 - Sorting: \mathcal{L} permutaciones, $f(\pi) = \sum j\pi(j)$, $\mathcal{N}^k(\pi^0)$ permutaciones obtenidas por intercambiar posiciones de k pares de elementos. Esta vecindad es exacta.
 - Minimum weight partition: dado un grafo $G = (V, E)$, $w : E \rightarrow \mathbb{R}$, objetivo encontrar V_1, V_2 de cardinalidad $|V|/2$ cuyo corte sea de peso mínimo; \mathcal{L} particiones equilibradas, $f(V_1, V_2) = \sum (w_{i,j} : i \in V_1, j \in V_2)$.
 - TSP:

Algunas consideraciones:

- En general necesitamos encontrar una solución inicial, encontrar soluciones vecinas, y evaluar la función objetivo.
 - En algunos casos, encontrar soluciones factibles es \mathcal{NP} -duro, por ejemplo TSP con ventanas de tiempo.
 - Calcular funciones objetivos puede ser difícil, por ejemplo, en el caso de VLSI, donde se quiere encontrar la posición de distintos elementos en una placa de circuitos de forma tal que el largo de las conexiones sea minimizada y el tamaño de la placa también. El problema es que los cables deben seguir un cierto modelo, y para algunos modelos el problema es intratable.

Ejemplo:

Job Shop Scheduling

Consideramos un conjunto \mathcal{J} de n trabajos, un conjunto \mathcal{M} de m máquinas, y un conjunto de operaciones \mathcal{O} . Cada operación $v \in \mathcal{O}$ tiene un trabajo $J_v \in \mathcal{J}$ al cual pertenece, una máquina $M_v \in \mathcal{M}$ donde debe realizarse y un tiempo de proceso $p_v \in \mathbb{Z}_+$. Además existe un orden \leq_J para las operaciones de cada trabajo. El problema es encontrar un tiempo de comienzo de operación, satisfaciendo las restricciones de precedencia que minimice el tiempo en que la última operación se finaliza, y donde cada máquina puede realizar un trabajo a la vez.

Ejemplo:

Una Vecindad (grafo disjuntivo):

Definimos el grafo $G = (\mathcal{O}, A, E)$ donde

$A = \{(v, w) : v \leq_J w\}$ y $E = \{(v, w) : M_v = M_w\}$.

Asociamos a cada nodo un peso p_v . Toda solución factible puede representarse como un grafo orientado en $\overline{G}(\mathcal{O}, A \cup \overline{E})$ sin ciclos. El costo de la solución es el peso del camino más largo en \overline{G} , ¿Cómo computamos? (TS).

k-exchange:

Podemos definir $\mathcal{N}(\overline{G})$ como aquellos grafos aciclicos que pueden obtenerse a partir de \overline{G} cambiando la orientación de k arcos en E .

¿Cómo verificamos factibilidad? (DFS)

Trade-offs clásicos:

- Problema de búsqueda local es existencia de óptimos locales *pobres*.
- Para valores pequeños de k , vecindades de k – *exchange* son fácilmente explorables pero poseen óptimos locales pobres.
- Valores de k más grandes proveen soluciones mejores, pero tienen tiempos de búsqueda del orden n^k , que puede hacer la búsqueda prohibitiva.
- Aun así, queremos ambas cosas.

Vecindades de profundidad variable:

- Vecindades de profundidad variables:
 - Introducida por Lin-Kernigan [1970] para partición de grafos y para el TSP.
 - Ha sido extendida para distintos tipos de problemas.
 - Aplicación a partición de grafo uniforme:
 - Dado un grafo $G = (V, E)$, $w : E \rightarrow \mathbb{R}$, objetivo encontrar V_1, V_2 de cardinalidad $|V|/2$ cuyo corte sea de peso mínimo; \mathcal{L} particiones equilibradas, $f(V_1, V_2) = \sum (w_{i,j} : i \in V_1, j \in V_2)$.
 - La ganancia de intercambiar (a, b) en V_1, V_2 esta dada por

$$g(a, b) = \sum_{\substack{\{a,v\} \in E, \\ v \in V_2 \setminus \{b\}}} w_{\{a,v\}} - \sum_{\substack{\{a,v\} \in E, \\ v \in V_1}} w_{\{a,v\}} + \sum_{\substack{\{b,u\} \in E, \\ u \in V_1 \setminus \{a\}}} w_{\{b,u\}} - \sum_{\substack{\{b,u\} \in E, \\ u \in V_2}} w_{\{b,u\}}$$

Vecindades de profundidad variable:

- Aplicación a partición de grafo uniforme:
- Nóte que $g(a, b)$ puede ser positivo o negativo.
- La idea es hacer secuencias de k pasos de 2-exchange, donde cada paso *tentativo* se escoge según los valores de $g(a, b)$.
- Usualmente los valores de k varían a lo largo del algoritmo.

Vecindades de profundidad variable:

- 1: $\forall v \in V, m[v] \leftarrow 0, k \leftarrow 0, G(k) \leftarrow 0.$
- 2: **for** $k = 0, k \leq n, k++$ **do**
- 3: Sean a_k, b_k maximizando $g(a_k, b_k)$ con
 $m[a_k] = m[b_k] = 0$
- 4: $m[a] \leftarrow m[b] \leftarrow 1$, realizar cambio *tentativo*
 $g(a_k, b_k), G(k) = G(k - 1) + g(a, b).$
- 5: Escoger $k = \operatorname{argmax}\{G(k) : k = 1, \dots, n\}$ ($G(n) = 0$).
- 6: **if** $G(k) > 0$ **then**
- 7: Realizamos cambios $g(a_i, b_i)$ para $i = 1, \dots, k$,
 partición resultante es vecina de la original.
- 8: **else**
- 9: Solución actual no tiene vecino.

Vecindades de profundidad variable:

- Pasos intermedios pueden empeorar solución.
- Características centrales:
 - Regla de *locking*: \Rightarrow búsqueda en tiempo polinomial.
 - Función de ganancia: Medir cambios en f.o.
- Es sub-conjunto de *n – exchange*!
- Podemos enriquecer la búsqueda usando *back-tracking*.
- Apurar ejecución restringiendo *2 – exchange*.
 - Separar *2 – exchange* como secuencia de *1 – exchange* (Graph Partitioning).
 - Considerar sólo vecinos *más cercanos* (TSP).
 - Considerar arcos en camino máximo (Job-Shop).
 - Test de factibilidad es innecesario.
 - Vecindad es débilmente óptimo conexa.

Otras formas de extender búsqueda local:

- Búsqueda local repetida (*repeated* LS):
 - Repetir búsqueda con múltiples puntos de partida.
 - Mientras tengamos tiempo..... seguir probando!
- Búsqueda local multinivel (*multilevel* LS):
 - Alternar uso de vecindades para evitar óptimos locales.
 - Una alternativa es *perturbar* un óptimo local con un movimiento (*aleatoreo?*) en otra vecindad. Esto se llama *iterated* LS.
 - Cuando la perturbación no es aleatoria, si no que busca algún tipo de mejora local, se le llama *chained* LS (ejemplo 4-bridge Kick para TSP).

Otras formas de extender búsqueda local:

- Problemas de LS:
 - Existencia de *muchos* óptimos locales.
 - ¿Cómo paliamos este problema?
- Simulated annealing (SA)
 - Introducido por Kirkpatrick et. al [1983], Černý [1985].
 - Idea básica es replicar el fenómeno físico del enfriamiento lento de metales.
 - Proceso físico: someter el metal a *baño* térmico.
 - Lentamente bajar la temperatura ambiente.
 - Resultado: configuración física de baja energía.
 - SA es un algoritmo de LS *randomizado*, donde el paso 3 se ejecuta en forma aleatoria.
 - Vecinos con mejor f.o. son siempre aceptados.
 - Vecinos con peor f.o. son aceptados según una probabilidad decreciente en el algoritmo.

► Algoritmo LS

Otras formas de extender búsqueda local:

- Simulated annealing (cont.)
 - Mecanismo de control de probabilidad se llamado *programa de enfriamiento*.
 - En teoría, converge (en probabilidad) a solución óptima, siempre y cuando vecindad sea fuertemente conexa.
 - Tiempo de convergencia es exponencial.
 - En la práctica se usan programas de enfriamiento rápidos.
 - Usualmente, probabilidad de aceptar soluciones está dada por :

$$\mathbb{P}\{\text{aceptar } j\} = \begin{cases} 1 & \text{si } f(j) \leq f(i) \\ e^{\left(\frac{f(i)-f(j)}{c_k}\right)} & \text{si } f(j) > f(i) \end{cases}$$

Otras formas de extender búsqueda local:

- Simulated annealing (cont.)
 - c_k se conoce como temperatura en la iteración k .
 - Generalmente usan temperaturas decrecientes en k .
 - Podemos de-randomizar, usando umbrales de aceptación.
 - No hay resultados de convergencia para versión determinista.
- Tabu Search (TS)
 - Introducido por Glover [1989,1990].
 - En este caso cambiamos el paso 3 por tomar mejor vecino, incluso si es peor que solución actual.
 - Para evitar ciclos, se prohíbe *deshacer* últimos pasos.

Otras formas de extender búsqueda local:

- Tabu Search (cont.)
 - Este mecanismo de control es el llamado *lista tabú*.
 - Vecinos en la lista tabú no participan del paso 3.
 - No existen resultados teóricos de convergencia.
 - Necesita una parametrización especial según problema enfrentado.
- Genetic Algorithms (GA)
 - Basado en Holland [1975] y Goldberg [1989].
 - Idea básica es usar mecanismos generales de *evolución de poblaciones*.
 - Se presume que en una población sobreviven individuos más *adaptados*.
 - Evolución dada por proceso de *recombinación* y *mutación* genética.

Otras formas de extender búsqueda local:

Algoritmo genético:

- 1: **Inicialización:** Construir población original de n soluciones.
- 2: **repeat**
- 3: **Mejora:** Aplicar LS en la población actual.
- 4: **Recombinar:** Generar m hijos a través de los operadores de *recombinación* y *mutación*.
- 5: **Mejora:** Aplicar LS en hijos.
- 6: **Selección:** Reducir la población a su tamaño original, usando alguna medida de *calidad* de los individuos (no necesariamente f.o.).
- 7: **until** **Evolución:** Alguna condición de término
- 8: **return** Mejor solución en la población actual.

Análisis y Complejidad

- Salvo para vecindades exactas, imposible dar cotas de calidad (GAP) o de tiempo de ejecución.
- En la práctica, LS converge a buenas soluciones en poco tiempo en una variedad de problemas.
- Resultados empíricos:
 - Graph Partitioning: Lin-Kernighan [1970],
GAP $\leq 2\text{-}3\%$, tiempo $\approx \mathcal{O}(n^{2.4})$.
 - Graph Partitioning: Fiduccia-Mattheyses,
GAP $\leq 2\text{-}3\%$, tiempo $\approx \mathcal{O}(n \log n)$.
 - Job Shop: Lenstra et. al [1992], 2-exchange,
GAP $\approx 15\%$.
 - Job Shop: Nowicki-Smutnicki [1996],
Balas-Vazacopoulos [1998], GAP $\approx 0.5\%$, tiempo \approx
minutos para 100 trabajos, 20 máquinas.

Análisis y Complejidad

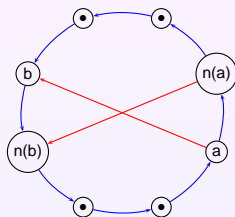
- Resultados empíricos (cont.):
 - TSP: Reiter-Sherman [1965], óptimo hasta 57 ciudades, tiempo ≤ 20 segundos.
 - TSP: 2-Opt y 3-Opt, GAP $\approx 3\text{-}6\%$.
 - TSP: Lin-Kernighan [1973], GAP $\leq 2\%$, tiempo 1.000.000 ciudades ≤ 1 hora.
 - TSP: Lin-Kernighan-Helsgun, GAP $\leq 0,5\%$, tiempo 1.000.000 ciudades \approx minutos.
- Resultados teóricos:
 - Peor caso: tiempo puede ser exponencial (simplex).
 - Vecindades exactas para TSP, deben tener tamaño al menos exponencial.
 - Si $\mathcal{P} \neq \mathcal{NP}$, vecindades certificadas en tiempo polinomial para TSP, no pueden ser exactas.

Análisis y Complejidad

- Resultados teóricos (cont.):
 - TSP simétrico con desigualdad triangular, vecindades de arcos, k -Opt tiene $\text{GAP} \geq \frac{1}{4} \sqrt[2k]{n}$.
 - TSP euclidianos, vecindades de arcos, k -Opt tiene $\text{GAP} \approx \mathcal{O}(\log n)$.
 - TSP euclidianos, vecindades de arcos, existen instancias con $\text{GAP} \Theta(\log(n)/\log(\log(n)))$.
 - TSP, si $\mathcal{P} \neq \mathcal{NP}$, no existen vecindades certificables en tiempo polinomial que aseguren un GAP fijo independiente del tamaño de la instancia.
 - TSP, vecindades de arcos, 2-Opt, Existen instancias y puntos de partida que requieren una cantidad exponencial de pasos para alcanzar óptimo local.

Lin-Kernighan Heuristic

- Basic Idea: improve one edge at a time.
 - Ask for $c(a, n(a)) - c(n(a), n(b)) > 0$
 - Such nodes called *promising*
- Basic Algorithm:
 - 1 $\Delta \leftarrow 0$
 - 2 while \exists promising nodes.
 - 3 Choose b promising,
 $\Delta \leftarrow \Delta + c(a, n(a)) + c(b, n(b)) - c(a, b) - c(n(a), n(b))$.
 - 4 do *flip*($n(a), b$).
 - 5 If $\Delta > 0$ return current tour.



Lin-Kernighan Refinements

- How do we choose b ?
 - Maximize $c(b, n(b)) - c(n(a), n(b))$.
 - Only consider k closest neighbors of $n(b)$.
- What if we do not succeed?
 - Allow backtracking.
 - More at lower levels.
 - Try also to replace $(p(a), a)$.
 - Sort promising nodes by $c(n(a), n(b))$.
- Basic Algorithm:
 - 1 $\Delta \leftarrow 0$
 - 2 while \exists promising nodes.
 - 3 Choose b promising, $\Delta \leftarrow \Delta + c(a, n(a)) + c(b, n(b)) - c(a, b) - c(n(a), n(b))$.
 - 4 do *flip*($n(a), b$).
 - 5 If $\Delta > 0$ return current tour.

Lin-Kernighan Refinements

- How do we choose a ?
 - Set all nodes as marked.
 - While there are marked nodes.
 - Call `lk_search(v , T)` for some marked node v .
 - if unsuccessful, unmark v .
 - if successful, mark all endpoints in the flip sequence.
- Can we do better?
 - While there is available time, generate a new initial tour T , call `lin_kernighan(T)`, keep best tour.
 - Called repeated Lin-Kernighan.
 - Best approach up to 1991.
 - Introduction of the *kick* concept.
 - Idea is to look harder close to *good* tours.
 - Called chained Lin-Kernighan.
 - Usual kick is the 4-bridge perturbation.

Basic Routines

- `flip(a,b)` - inverts the segment from a to b .
- `next(a)` - returns node after a in the tour.
- `prev(a)` - returns node before a in the tour.
- `sequence(a,b,c)` - returns 1 if b lies in the segment $a - c$ of the tour.

Instances

Name	Size	Target tour
pcb3038	3038	139070
usa13509	13509	20172983
pla85900	85900	143564780

Number of operations

Function	pcb3038	usa13509	pla85900
lin_kernighan	141	468	1842
lin_kernighan winners	91	261	1169
average number of flip	61	99	108
lk_search	19,855	95,315	376,897
lk_search winners	1,657	9,206	29,126
flip	180,073	1,380,545	5,110,340
undo flip	172,396	1,336,428	4,925,574
size of flip	75	195	607
flip size ≤ 5	67,645	647,293	1,463,090
next	662,436	6,019,892	14,177,723
prev	715,192	4,817,483	13,758,748
sequence	89,755	773,750	2,637,757

Implementations:

Implementation	Instance			% total time		
	pcb	usa	pla	flip %	n/p %	seq %
arrays	7.2	246.6	10422.5	97	1	1
A + RB	1.6	21.6	265.9	85	1	1
list	50.8	5929.4	¿50000	0	93	6
L + 2-search	15.7	426.7	24047.9	0	55	44
L + index + n/p	1.8	65.6	697.3	92	1	1
L + 3 levels	2.3	18.5	81.4	38	19	4
L + 2 layers	1.2	10.1	43.9	26	6	1
Binary Tree	1.4	12.6	52.9	17	24	6