Búsqueda local

Tecnología Digital V: Diseño de Algoritmos
Universidad Torcuato Di Tella

UNIVERSIDAD
TORCUATO DI TELLA

Introducción

Consideremos el problema de optimización

$$\min_{s \in S} f(s)$$

donde

- S es el conjunto (discreto) de soluciones factibles.
- $f(s) : S \to \mathbb{R}$ es la función objetivo.

Motivación

- Para los *algoritmos exactos*, exploramos el espacio de soluciones de forma **exhaustiva**.
- Si esto se vuelve prohitivo, otra alternativa es considerar una solución inicial y aplicar una **secuencia** de mejoras, apuntando a encontrar al final una solución de buena calidad (eventualmente, la *óptima*).
- Algunas preguntas: *Cómo definimos esta secuencia? Convergencia? Complejidad?*

Sea $s \in S$ para nuestro problema.

Vecindario

Todas las soluciones que se pueden obtener mediante modificaciones simples de una solución $s \in S$. Notamos $N(s) \subseteq S$ a un vecindario de la solución $s \in S$.

Movida

Actualización de la solución por otra (con mejor función objetivo).

Criterio de aceptación

Condición mediante la cual se acepta una movida. Ejemplos:

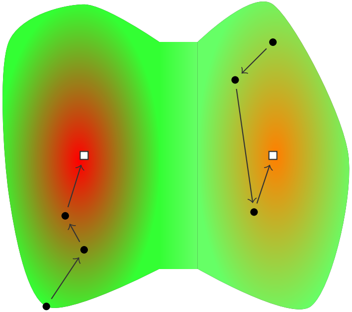
- *Best improvement*: $s' \in N(s)$ tal que

$$s' = \arg \min_{s \in N(s)} f(s)$$

  y $f(s') < f(s)$.
- *First improvement*: primer $s' \in N(s)$, $f(s') < f(s)$, que identifiquemos durante la exploración de $N(s)$.

BusquedaLocal()

1. Construir una solución $s^* \in S$
2. **repetir**
3. elegir $s \in N(s^*)$ tal que $f(s) < f(s^*)$ y actualizar $s^* = s$
4. **mientras** exista $s \in N(s^*)$ tal que $f(s) < f(s^*)$



Convergencia

Un algoritmo de búsqueda local converge a un *mínimo local* s^* respecto a $N(s)$, es decir, $f(s^*) \leq f(s)$ para todo $s \in N(s^*)$.

Observación

La convergencia a un *óptimo global* no está garantizada.

BusquedaLocal()

1. Construir una solución $s^* \in S$
2. **repetir**
3. elegir $s \in N(s^*)$ tal que $f(s) < f(s^*)$ y actualizar $s^* = s$
4. **mientras** exista $s \in N(s^*)$ tal que $f(s) < f(s^*)$

En la práctica

Un algoritmo de búsqueda local requiere definir (al menos) los siguientes elementos:

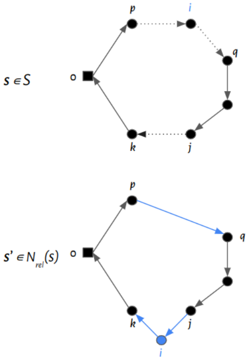
- Cómo se representa una solución factible?
- Cómo se construye la primera solución factible?
- Qué *vecindario(s)* considera el algoritmo?
- Qué *criterio de aceptación* se implementa en cada caso?

Ejemplo: relocate para TSP

Definición

Dado un tour factible s para el TSP, el operador *relocate* (o también llamado *node insertion*), N_{rel}, se define como:

- considerar un vértice $i \in s$, adyacente a los vértices p y q.
- Dado un arco $(j, k) \in s$, $j \neq p$ y $k \neq q$, un vecino de s está dado por:
  - remover a i de su posición;
  - agregar el arco (p, q);
  - insertar i entre j y k definiendo los ejes (j, i) e (i, k).
- Aplicar estos pasos para todo i y $(j, k) \in s$.



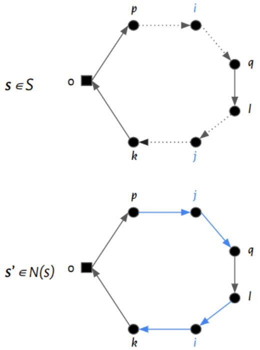
Propiedades

- **Mejora**. Una solución $s' \in S$ es una mejora de s si

$$c_{pq} + c_{ji} + c_{ik} < c_{jk} + c_{pi} + c_{iq},$$

  que puede ser verificado en $O(1)$.
- **Complejidad**. La complejidad computacional de explorar el vecindario completo $N_{\text{rel}}(s)$ es $O(n^2)$.

Ejemplo: swap para TSP



Definición

Dado un tour factible s para el TSP, el operador *swap*, N_{swap}, se define como:

- considerar dos vértices distintos i, j;
- intecambiar las posiciones de i y j en s;
- aplicar estos cambios para todo par de vértices i, j.

Propiedades

- **Mejora**. $s' \in N_{\text{swap}}$ es una mejora de s si

$$c_{pj} + c_{jq} + c_{li} + c_{ik} < c_{pi} + c_{iq} + c_{lj} + c_{jk},$$

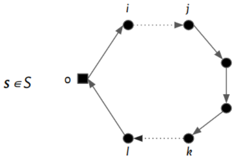
  que puede ser verificado en $O(1)$.
- **Complejidad**. La complejidad computacional de explorar el vecindario N_{swap} es $O(n^2)$.

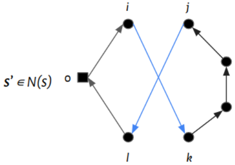
Ejemplo: 2-opt para TSP

Definición

Dado un tour factible s para el TSP, el operador *2opt*, $N_{2\text{opt}}$, se define como:

- considerar dos ejes distintos $(i, j), (k, l) \in s$;
- conectar $i \to k$, $j \to l$ y ajustar el tour;
- aplicar estos cambios para todo par de ejes (i, j), $(k, l) \in s$.





Propiedades

- **Mejora**. $s' \in N_{2\text{opt}}$ es una mejora de s si

$$c_{ik} + c_{k\to j} + c_{jl} < c_{ij} + c_{j\to k} + c_{kl},$$

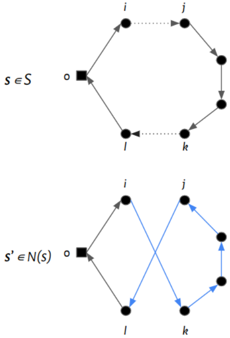
  que puede ser verificado en $O(1)$.
- **Complejidad**. La complejidad computacional de explorar el vecindario $N_{2\text{opt}}$ es $O(n^2)$.

Ejemplo: 2-opt para ATSP

Definición

Dado un tour factible s para el ATSP, el operador *2opt*, $N_{2\text{opt}}$, se define como:

- considerar dos ejes distintos $(i, j), (k, l) \in s$;
- conectar $i \to k$, $j \to l$ y ajustar el tour;
- aplicar estos cambios para todo par de ejes (i, j), $(k, l) \in s$.



Propiedades

- **Mejora**. $s' \in N_{2\text{opt}}$ es una mejora de s si

$$c_{ik} + c_{k \to j} + c_{jl} < c_{ij} + c_{j \to k} + c_{kl},$$

  que puede ser verificado en ¿?.

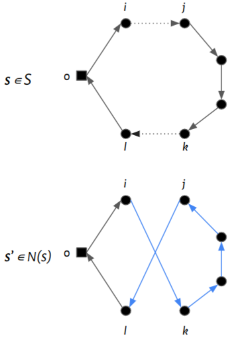
- **Complejidad**. La complejidad computacional de explorar el vecindario $N_{2\text{opt}}$ es ¿?.

Ejemplo: 2-opt para ATSP

Definición

Dado un tour factible s para el ATSP, el operador *2opt*, $N_{2\text{opt}}$, se define como:

- considerar dos ejes distintos $(i, j), (k, l) \in s$;
- conectar $i \rightarrow k$, $j \rightarrow l$ y ajustar el tour;
- aplicar estos cambios para todo par de ejes (i, j), $(k, l) \in s$.



Propiedades

- **Mejora**. $s' \in N_{2\text{opt}}$ es una mejora de s si

$$c_{ik} + c_{k \rightarrow j} + c_{jl} < c_{ij} + c_{j \rightarrow k} + c_{kl},$$

  que puede ser verificado en $O(n)$.
- **Complejidad**. La complejidad computacional de explorar el vecindario $N_{2\text{opt}}$ es $O(n^3)$. **¿Podemos mejorarla?**

Ejemplo: 2-opt para ATSP

Para reducir la complejidad de validar si un movimiento 2opt mejora la solución en el ATSP, se pueden seguir dos caminos:

- asumir que el cálculo se hace en un determinado orden, e ir actualizando ciertos valores convenientemente;
- incorporar estructuras de datos que mantienen información sobre $s = (0, v_1, \ldots, v_n, 0)$;
  - $s[i] = c_{0,v_1,\ldots v_i}$ el costo parcial de la ruta $0 \longrightarrow v_i$.
  - $rev\_s[i] = c_{0,v_n,v_{n-1},\ldots,v_i}$ el costo parcial del reverso de la ruta $v_i \longleftarrow 0$.
  - Si j aparece despupés de i,

$$\begin{aligned} c_{i\to j} &= s[j] - s[i] \\ c_{j\to i} &= rev\_s[i] - rev\_s[j] \end{aligned}$$

Propiedades

- **Mejora**. $s' \in N_{2\mathrm{opt}}$ es una mejora de s si

$$c_{ik} + c_{k\to j} + c_{jl} < c_{ij} + c_{j\to k} + c_{kl},$$

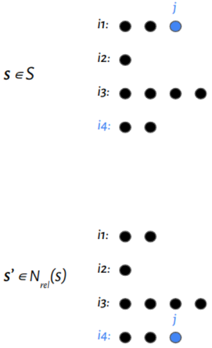
  que puede ser verificado en $O(1)$.
- **Complejidad**. La complejidad computacional de explorar el vecindario $N_{2\mathrm{opt}}$ es $O(n^2)$.
  Qué debemos hacer luego de actualizar la solución en caso de mejora?

Ejemplo: relocate para GAP

Definición

Dada una solución factible para el GAP, el operador *relocate*, N_{rel}, se define como:

- considerar un cliente $j \in s$ asignado al depósito i;
- considerar un depósito $k \neq i$;
  - remover j de su ubicación actual;
  - insertar j en el depósito k;
- Aplicar estos pasos para todos los clientes $j \in s$ y los depósitos distintos al actual.



Propiedades

- **Mejora**. Una solución $s' \in N_{\text{rel}}(s)$ es una mejora de s si

$$c_{kj} < c_{ij} \text{ y } d_{kj} \leq \bar{c}_k,$$

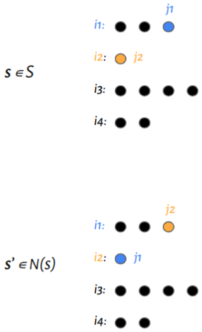
  con $\bar{c}_k$ la capacidad remanente del depósito k, que puede ser verificado en $O(1)$.
- **Complejidad**. La complejidad computacional de explorar el vecindario completo $N_{\text{rel}}(s)$ es $O(n \times m)$.

Ejemplo: swap para GAP

Definición

Dada una solución factible para el GAP, el operador *swap*, N_{swap}, se define como:

- considerar dos clientes distintos $j_1, j_2 \in s$ asignado a los depósitos $i_1 \neq i_2$, respectivamente;
  - remover j_1 de i_1 e insertarlo en i_2;
  - remover j_2 de i_2 e insertarlo en i_1;
- Aplicar estos pasos para todos los clientes $j_1, j_2 \in s$ asignados a depósitos distintos.



Propiedades

- **Mejora**. Una solución $s' \in N_{\text{swap}}(s)$ es una mejora de s si

$$\begin{aligned} c_{i_2 j_1} + c_{i_1 j_2} &< c_{i_1 j_1} + c_{i_2 j_2} \\ d_{i_2 j_1} &\leq \bar{c}_{i_2} + d_{i_2 j_2} \\ d_{i_1 j_2} &\leq \bar{c}_{i_1} + d_{i_1 j_1} \end{aligned}$$

  con $\bar{c}_k$ la capacidad remanente del depósito k, que puede ser verificado en $O(1)$.
- **Complejidad**. La complejidad de explorar el vecindario $N_{\text{rel}}(s)$ es $O(n^2)$.

Y ahora?