Metaheurísticas

Tecnología Digital V: Diseño de Algoritmos
Universidad Torcuato Di Tella

UNIVERSIDAD
TORCUATO DI TELLA

Introducción

Consideremos el problema de optimización

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

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

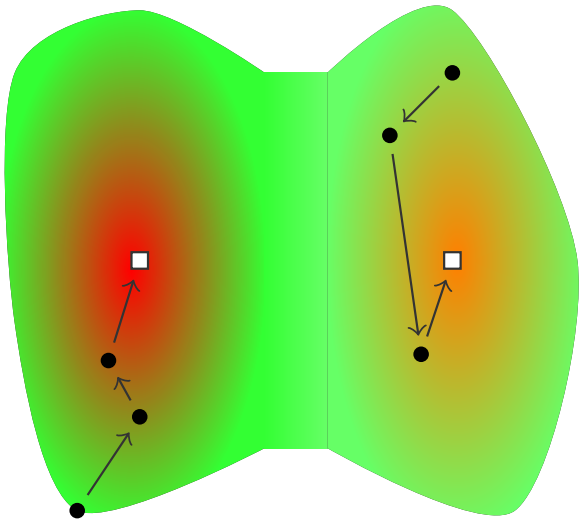
Hasta ahora

- **Heurísticas constructivas.** Proveen soluciones de razonables rápido, pero generalmente con cierto *gap* respecto a la solución óptima.
- **Búsqueda local.** Dada una solución, aplican una secuencia de mejoras. *Convergencia a mínimo local (respecto a un vecindario).*

Idea

Diseñar heurísticas y estrategias más sofisticadas para explorar mejor el espacio de soluciones y obtener soluciones de mejor calidad, **escapando de óptimos locales**, pero manteniendo tiempos de ejecución manejables.

Combinando operadores de búsqueda local



Observaciones

- Un *óptimo local* con respecto a un vecindario no lo es necesariamente para otro.
- Un *óptimo global* es un óptimo local respecto a todos los posibles vecindarios.
- En algunos problemas, *óptimos locales* para vecindarios *distintos* están *relacionados*.

Variable Neighborhood Descent (VND, Hansen et al. 2010)

Método que considera una **múltiples vecindarios**:

- compuesto de distintos vecindarios de búsqueda local;
- son explorados de forma *secuencial* y *determinística*;
- si un vecindario no tiene soluciones candidatas, se pasa al siguiente.

Variable Neighborhood Descent (VND)

Sea s una solución factible, y $k_{\max}$ la secuencia de vecindarios.

VND(s, $k_{\max}$)

1. Definir $k = 1$
2. **do**
3. Explorar el k-ésimo vecindario
   $s' = \arg\min_{x \in N_k(s)} f(x)$
4. Determinar la nueva solución y el próximo vecindario a explorar
   $s, k = \text{NeighbourhoodChange}(s, s', k)$
5. **while** $k \neq k_{\max}$
6. **return** s

NeighbourhoodChange(s, s', k)

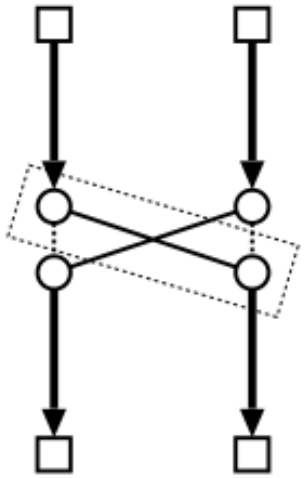
1. **if** $f(s') < f(s)$ **then**
2. $s = s', k = 1$
3. **else**
4. $k = k + 1$
5. **return** s, k

Ejemplos

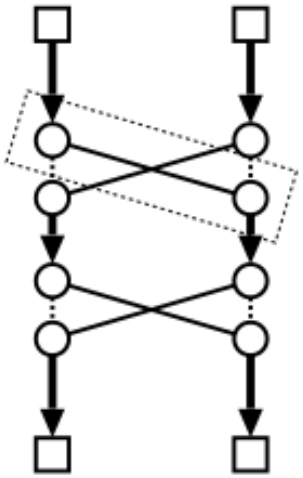
- ATSP: N_1 = relocate, N_2 = swap, N_3 = 2opt.
- GAP: N_1 = relocate, N_2 = swap.

Ejemplo: VRP

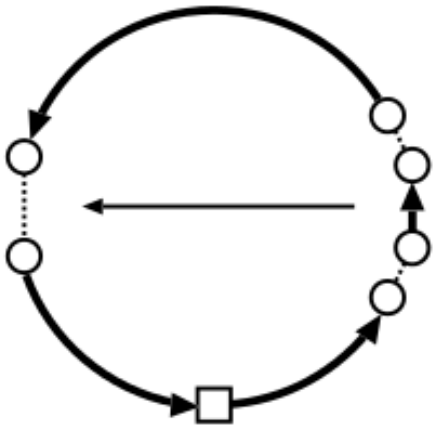
- Posibles vecindarios:



(a) 2-opt*



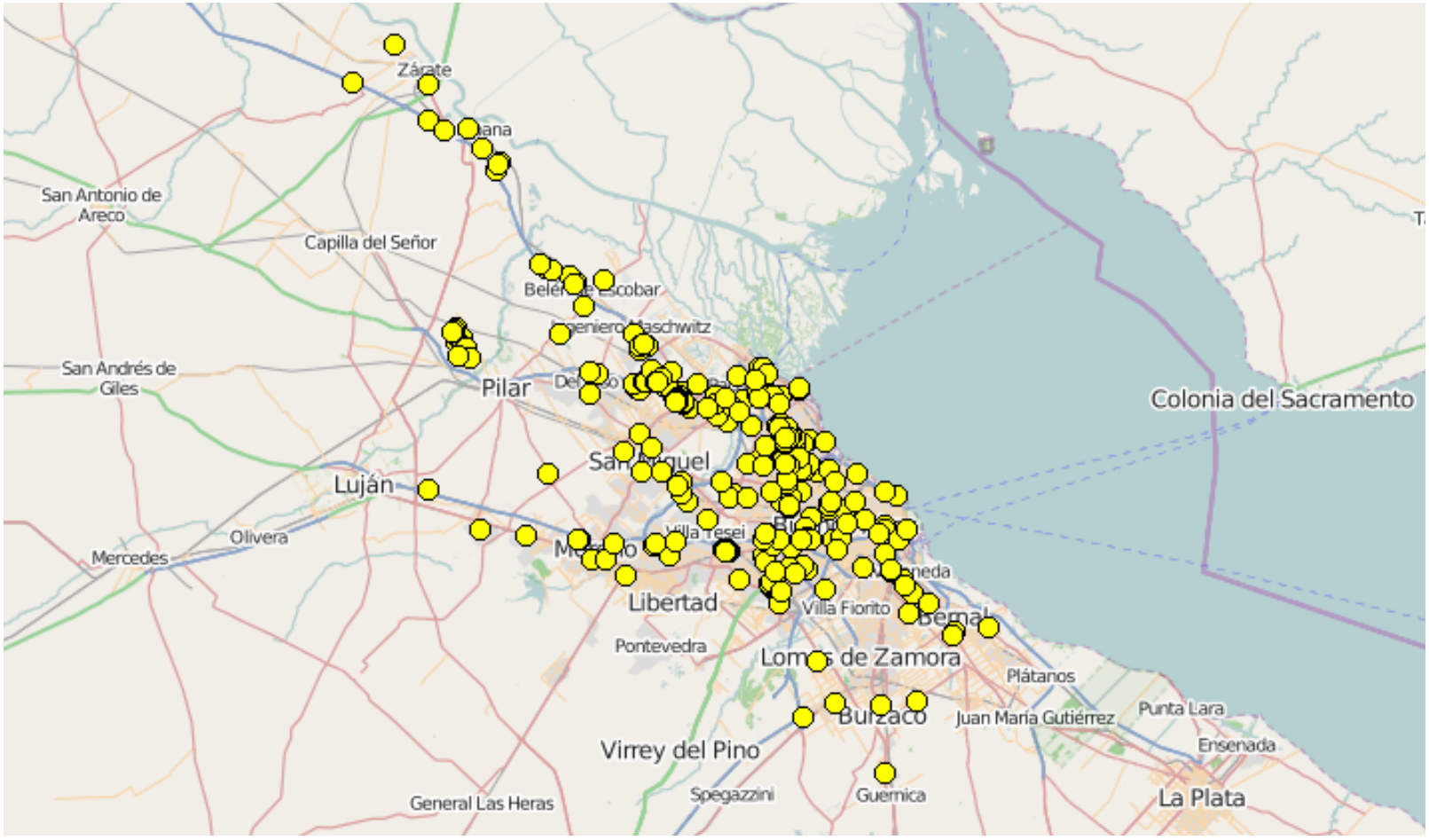
(b) Cross exchange



(c) Intra-route

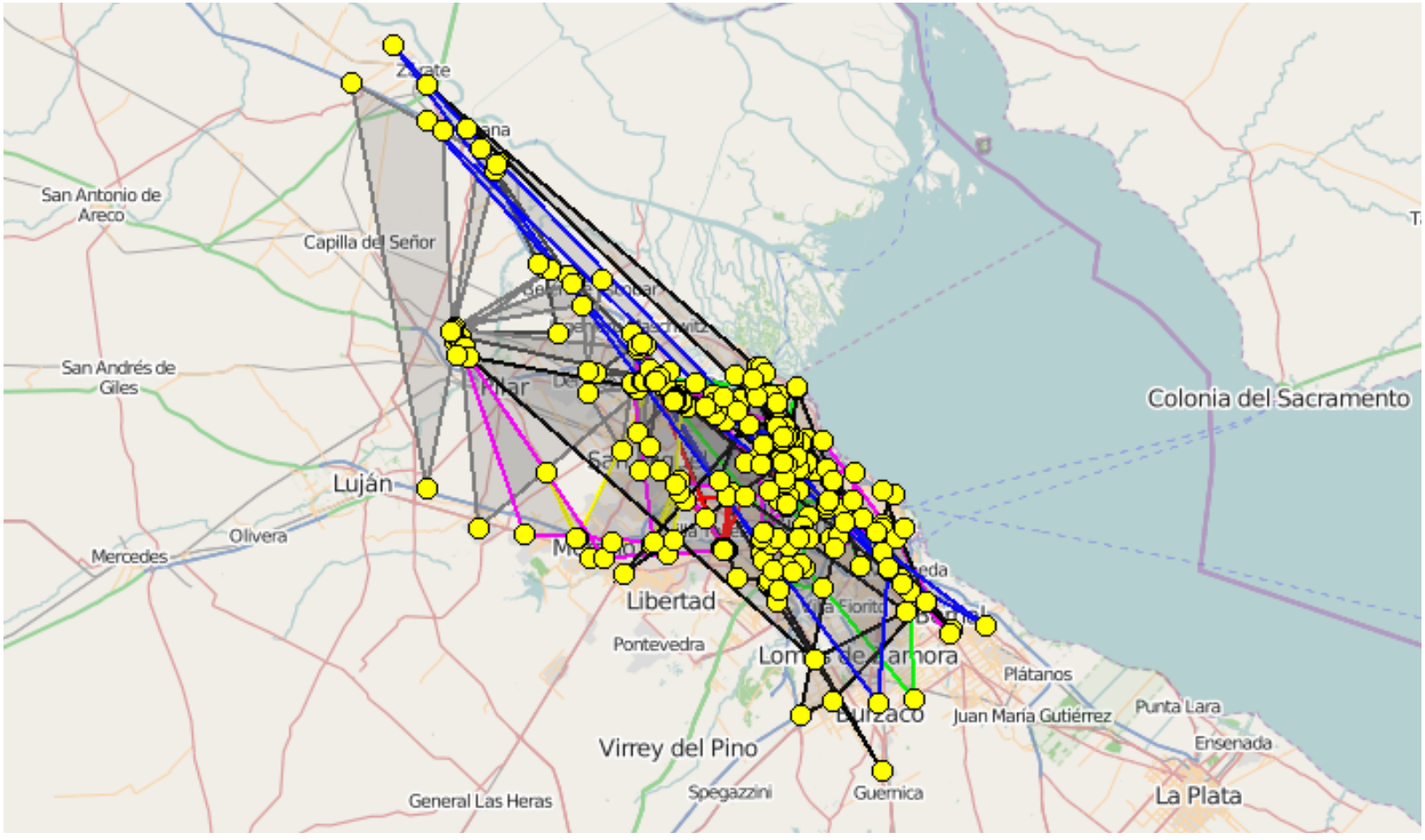
Ejemplo: VRP

- Una instancia con 261 clientes.



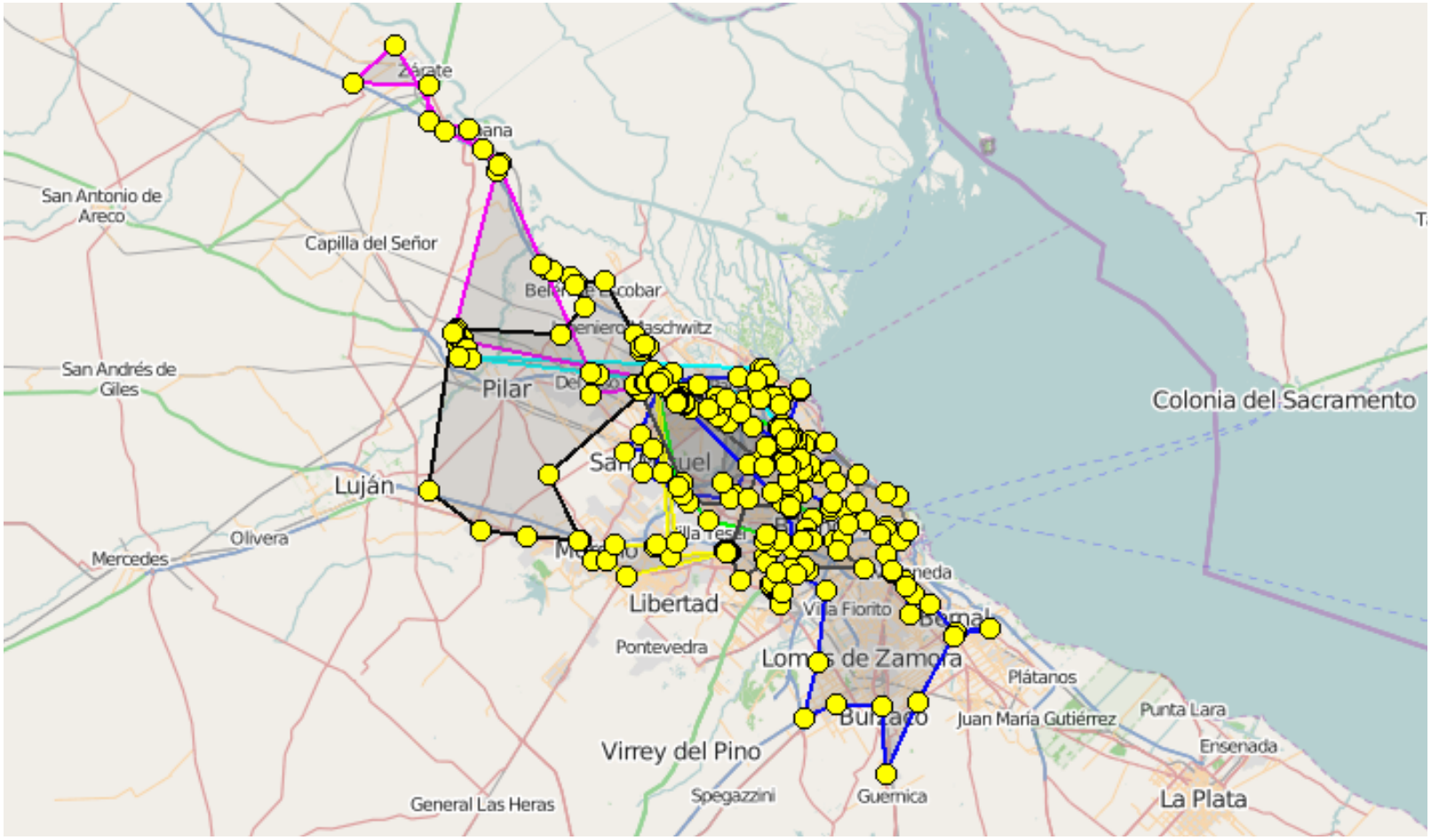
Ejemplo: VRP

- Resultado de un algoritmo goloso: 3885.97 km.



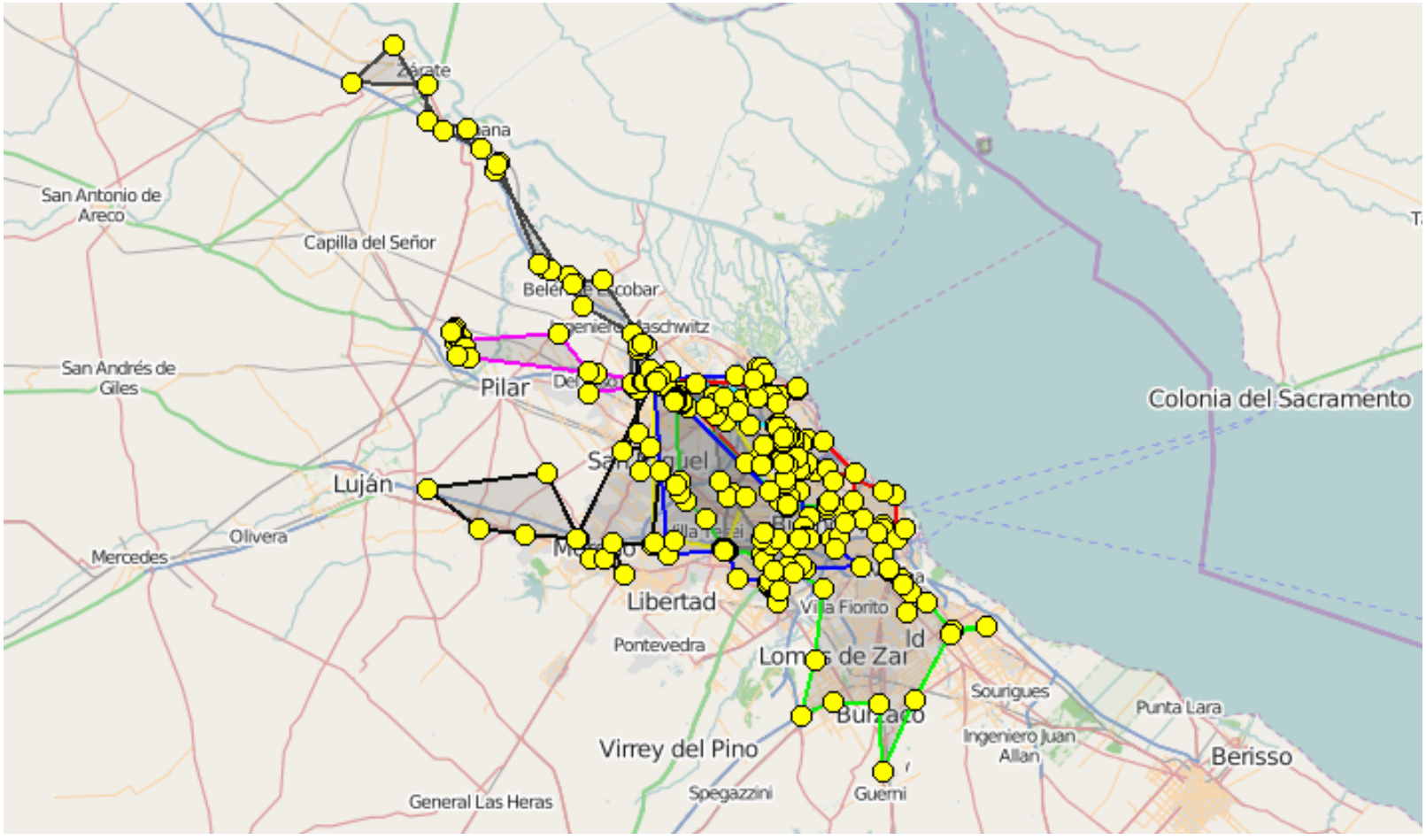
Ejemplo: VRP

- Resultado de la búsqueda local: 1383.29 km.

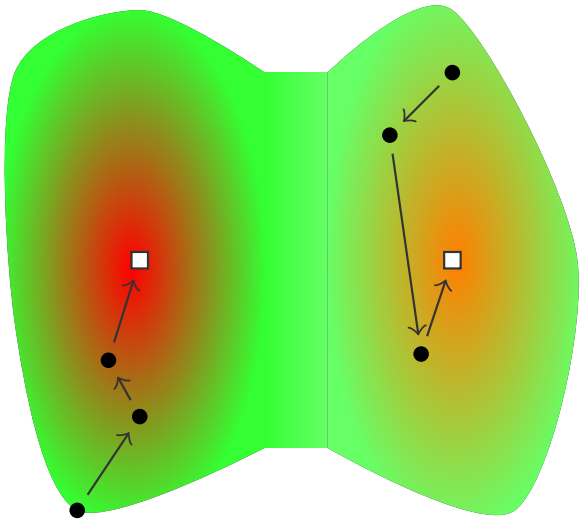


Ejemplo: VRP

- Resultado de varias búsquedas locales: 1269.54 km.



Escapando de óptimos locales



- **Algoritmos multistart.** Inicializar algorítmos clásicos desde diferentes puntos o utilizar técnicas de *randomización* para construir diferentes soluciones.
- **Estrategias de perturbación.** Dada una solución (factible), moverse a una nueva solución que tenga algunas partes en común con la original, pero (idealmente) descartando aquellas partes que no son importantes para la solución óptima.

Otras ideas

Aceptar soluciones que empeoran (e.g., *Simmulated Annealing* o *Tabú Search*).

En general

Ideas simples de explicar (y, a veces, programar) pero que requieren testing y tunning significativo.

Greedy Randomized Adaptive Search Procedure (Feo & Resende, 1995)

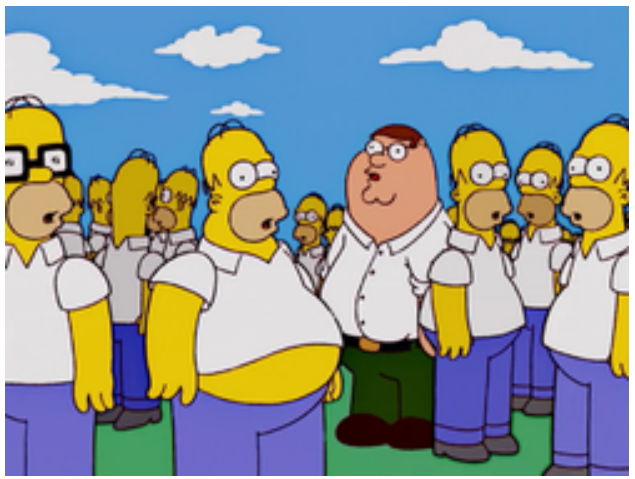
Idea principal

Algoritmo multistart.

- Elegir un algoritmo goloso y un valor rcl_size.
- En cada paso de la heurística golosa, en lugar de elegir la **mejor opción local**, definir una **lista restringida de candidatos (RCL)**.
- Elegir de forma aleatoria de la RCL la siguiente acción a tomar.

GRASP(n_iters, rcl_size)

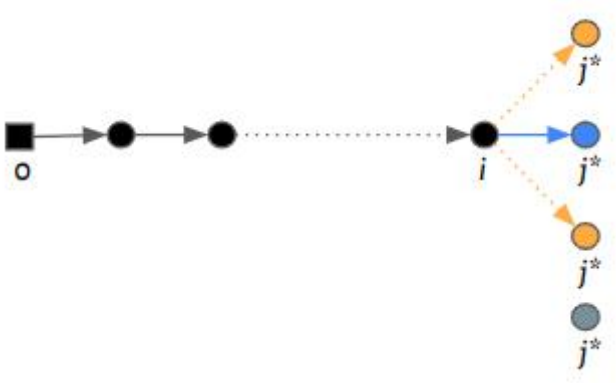
1. $s_{\text{best}} = \text{null}, z_{\text{best}} = \infty$
2. **for** $k = 1, \ldots, n\_iters$
3. $\quad s = \text{GreedyRandomized}(rcl\_size)$
4. $\quad s = \text{LocalSearch}(s)$
5. $\quad$ **if** $f(s) < f(s_{\text{best}})$
6. $\quad\quad s_{\text{best}} = s, z_{\text{best}} = f(s)$
7. **return** s_{best}

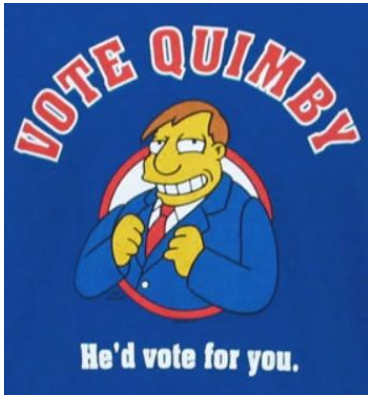


GRASP: ejemplo para ATSP

RandomizedNearestNeighbor($D = (V, A)$, $C = (c_{ij})$)

1. Definir $T = (0)$ con el depósito.Sea $U = V \backslash \{0\}$.
2. **while** $U \neq \emptyset$:
   Sea i el último vértice en T, i.e. $T = (0, \ldots, i)$.
   Sea de $RCL \subseteq U$ el conjunto con los k vecinos más cercanos a i.
   Elegir j^* de forma aleatoria de RCL.
   Definir $T = (0, \ldots, i, j^*)$, $U = U \backslash \{j^*\}$.
3. **return** T





Pregunta

Qué podemos usar como búsqueda local?

Iterated Local Search (ILS)

Idea principal

- Definir un algoritmo de búsqueda local para el problema.
- Tomar una solución inicial calculada por una heurística simple (eventualmente, algo random). Tomar esta solución como *incumbent*.
- Aplicar el algorimo de búsqueda local hasta llegar a un mínimo local.
- Tomando la solución actual y/o la mejor solución encontrada, construir la próxima solución a ser considerar *incumbent*.

IteratedLocalSearch(s_0, n_iters)

1. $s = s_0$, $s_{\text{best}} = s$
2. **for** $k = 1, \ldots, n\_iters$
3. $\quad s = \text{LocalSearch}(s)$
4. $\quad$ **if** $f(s) < f(s_{\text{best}})$
5. $\quad\quad s_{\text{best}} = s$
6. $\quad s = \text{GetNextSolution}(s, s_{\text{best}})$
7. **return** s_{best}

Intuición

Tratar de aprovechar que la solución actual es un *mínimo local* y, potencialmente, comparta parte de su estructura con el *óptimo global*.

ILS: ejemplo para el ATSP

IteratedLocalSearch(s_0, n_iters)

1. $s = s_0$, $s_{\text{best}} = s$
2. **for** $k = 1, \ldots, n\_iters$
3. $\quad s =$ LocalSearch(s)
4. $\quad$ **if** $f(s) < f(s_{\text{best}})$
5. $\quad\quad s_{\text{best}} = s$
6. $\quad s =$ GetNextSolution(s,s_{best})
7. **return** s_{best}

Búsqueda local

VND con:

- $N_1(s)$ = relocate.
- $N_2(s)$ = swap.
- $N_3(s)$ = 2opt.

Siguiente solución

Perturbar la solución

ILS: ejemplo para el ATSP

IteratedLocalSearch(s_0, n_iters)

1. $s = s_0$, $s_{\text{best}} = s$
2. **for** $k = 1, \ldots, n\_iters$
3. $\quad s = \text{LocalSearch}(s)$
4. $\quad$ **if** $f(s) < f(s_{\text{best}})$
5. $\quad\quad s_{\text{best}} = s$
6. $\quad s = \text{GetNextSolution}(s, s_{\text{best}})$
7. **return** s_{best}

Búsqueda local

VND con:

- $N_1(s)$ = relocate.
- $N_2(s)$ = swap.
- $N_3(s)$ = 2opt.

Siguiente solución

Perturbar la solución

Destroy & Repair

ILS: ejemplo para el ATSP

IteratedLocalSearch(s_0, n_iters)

1. $s = s_0$, $s_{best} = s$
2. **for** $k = 1, \ldots, n\_iters$
3. $\quad s = \text{LocalSearch}(s)$
4. $\quad$ **if** $f(s) < f(s_{best})$
5. $\quad\quad s_{best} = s$
6. $\quad s = \text{GetNextSolution}(s, s_{best})$
7. **return** s_{best}

Búsqueda local

VND con:

- $N_1(s)$ = relocate.
- $N_2(s)$ = swap.
- $N_3(s)$ = 2opt.

Siguiente solución

Perturbar la solución

Destroy & Repair

1. **Destroy.** Remover de s_{best} algunos vértices. Cómo?

ILS: ejemplo para el ATSP

IteratedLocalSearch(s_0, n_iters)

1. $s = s_0$, $s_{\text{best}} = s$
2. **for** $k = 1, \ldots, n\_iters$
3. $\quad s = \text{LocalSearch}(s)$
4. $\quad$ **if** $f(s) < f(s_{\text{best}})$
5. $\quad\quad s_{\text{best}} = s$
6. $\quad s = \text{GetNextSolution}(s, s_{\text{best}})$
7. **return** s_{best}

Búsqueda local

VND con:

- $N_1(s)$ = relocate.
- $N_2(s)$ = swap.
- $N_3(s)$ = 2opt.

Siguiente solución

Perturbar la solución

Destroy & Repair

1. **Destroy.** Remover de s_{best} algunos vértices. Cómo?
2. **Solución restringida.** Construir una solución parcial s', usando *shortcuts* entre clientes vértices desconectados.

ILS: ejemplo para el ATSP

IteratedLocalSearch(s_0, n_iters)

1. $s = s_0$, $s_{\text{best}} = s$
2. **for** $k = 1, \ldots, n\_iters$
3. $\quad s = \text{LocalSearch}(s)$
4. $\quad$ **if** $f(s) < f(s_{\text{best}})$
5. $\quad\quad s_{\text{best}} = s$
6. $\quad s = \text{GetNextSolution}(s, s_{\text{best}})$
7. **return** s_{best}

Búsqueda local

VND con:

- $N_1(s)$ = relocate.
- $N_2(s)$ = swap.
- $N_3(s)$ = 2opt.

Siguiente solución

Perturbar la solución

Destroy & Repair

1. **Destroy.** Remover de s_{best} algunos vértices. Cómo?
2. **Solución restringida.** Construir una solución parcial s', usando *shortcuts* entre clientes vértices desconectados.
3. **Repair.** Reinsertar en s' los clientes removidos. Cómo?

Comparación

- **GRASP**.
  - Simple e intuitiva. Puede ser fácilmente implementada si se tienen otros componentes.
  - Diversificar la búsqueda.
  - Restarts? El esfuerzo hecho en búsquedas locales previas se descarta.
- **ILS**.
  - Explota la estrucutra de mínimos locales.
  - Explota la existencia de operadores de búsqueda local.
  - Perturbación? Requiere pensar estrategias efectivas para obtener la próxima solución.
- **Otros enfoques?**
  - Esquemas bio-insipirados?
  - Intensificación / diversificación.
  - Y muchas otras ideas...