

# Inteligencia Artificial

## Técnicas de Búsqueda Incompleta

Elizabeth Montero Ureta

Departamento de Informática  
Universidad Técnica Federico Santa María  
Campus Santiago San Joaquín

2do Semestre 2018

## Conceptos Generales

Algoritmos Constructivos

Algoritmos Reparadores

## Algoritmos Constructivos

Greedy

## Algoritmos Reparadores

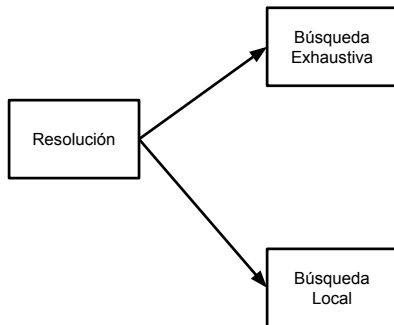
Hill-Climbing

Hill-Climbing+Restart

# Introducción

- ▶ Algoritmos que realizan Búsqueda exhaustiva garantizan obtener el óptimo global
- ▶ Para problemas de la vida real o de mayor complejidad (NP-Complejos), no pueden encontrar solución
- ▶ Sacrificamos el óptimo global por obtener soluciones de buena calidad en un tiempo reducido

# Clasificación de Técnicas de Resolución

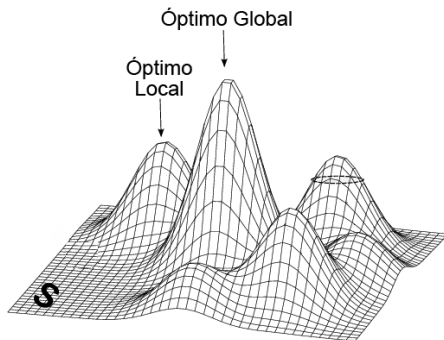


# Búsqueda Incompleta

¿Qué es Heurística?

- ▶ Criterios, principios o métodos que permiten determinar entre un conjunto de posibilidades, aquella que promete ser la más eficaz para resolver un problema.
- ▶ Representan el compromiso entre:
  - ▶ Necesidad de utilizar criterios simples
  - ▶ Distinguir entre buenas y malas elecciones
- ▶ Un método heurístico puede ser un método empírico utilizado para guiar acciones.
- ▶ Su objetivo es encontrar soluciones (c) aceptables.
- ▶ Se utilizan porque son, en general, eficientes computacionalmente y/o fáciles de implementar.
- ▶ No son muy precisas ni predecibles.

# Búsqueda Local



- ▶ Un algoritmo que realiza búsqueda local visita diferentes regiones del espacio de búsqueda
- ▶ El objetivo de estos algoritmos es encontrar el mejor óptimo local buscando un balance entre dos aspectos:
  - ▶ **Diversificación:** el algoritmo visita diferentes regiones para identificar secciones del espacio de búsqueda que son interesantes
  - ▶ **Intensificación:** el algoritmo se enfoca en una región y busca la mejor solución de ella

# Sección 1

## Conceptos Generales

# Conceptos Generales

- ▶ solución candidata: Se refiere a una instanciación completa (global) para el problema.
  - ▶ En estricto rigor no se debería hablar de soluciones puesto que éstas pueden ser *factibles* o *infactibles* (que no cumple con las restricciones).



# Conceptos Generales

- ▶ solución candidata: Se refiere a una instanciación completa (global) para el problema.
  - ▶ En estricto rigor no se debería hablar de soluciones puesto que éstas pueden ser *factibles* o *infactibles* (que no cumple con las restricciones).
- ▶ función de evaluación: Es capaz de representar cero, uno ó varios objetivos.
  - ▶ Informativa

## Conceptos Generales

- ▶ solución candidata: Se refiere a una instancia completa (global) para el problema.
  - ▶ En estricto rigor no se debería hablar de soluciones puesto que éstas pueden ser *factibles* o *infactibles* (que no cumple con las restricciones).
- ▶ función de evaluación: Es capaz de representar cero, uno ó varios objetivos.
  - ▶ Informativa
- ▶ representación: Corresponde a la estructura de la/s solución/es (c).
  - ▶ Coloreo: (V, A, V, V, A, R) (Lista Entera/Simbólica)
  - ▶ Mochila: (1, 0, 1, 0) (Lista Binaria)
  - ▶ TTP:  $E$  vs  $2*(E-1)$  (Matriz Entera/Simbólica)

|   | 1  | 2  | 3  | 4  | 5  | 6  |
|---|----|----|----|----|----|----|
| A | C  | B  | D  | -C | -B | -D |
| B | D  | -A | -C | -D | A  | C  |
| C | -A | D  | B  | A  | -D | -B |
| D | -B | -C | -A | B  | C  | A  |

# Modelado versus Representación

- ▶ ¿Para qué modelar?
- ▶ ¿Siempre debo representar las soluciones (c) según la forma que se especificó en el modelo?

## Ejemplo: Vendedor viajero

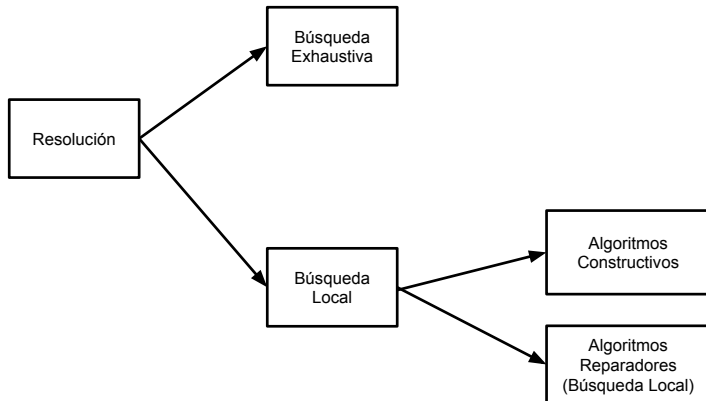
- ▶ Modelo

$$x_{ij} = \begin{cases} 1 & \text{Si el vendedor pasa desde la ciudad } i \text{ a la ciudad } j \\ 0 & \text{si no} \end{cases}$$

versus

- ▶ Representación  
 $x_i$ : Visita la ciudad  $x_i$  en el  $i$ -ésimo paso.

# Clasificación de Técnicas de Resolución



## Sección 1

### Conceptos Generales

# Algoritmos Constructivos

# Algoritmos Constructivos

## Características Algoritmos Constructivos

- ▶ Generan soluciones comenzando desde una solución parcial inicialmente vacía
- ▶ Se agregan componentes a la solución parcial hasta que esté completa
- ▶ Requieren además de una definición previa de:
  - ▶ Punto de Partida: desde donde comienza a construir la solución
  - ▶ Función Miope: toma decisiones localmente óptimas guiada por la función de Evaluación

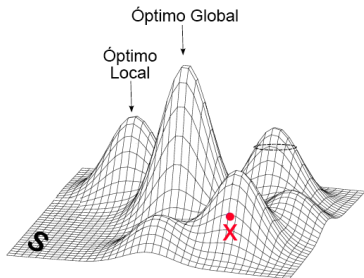
# Sección 1

## Conceptos Generales

## Algoritmos Reparadores

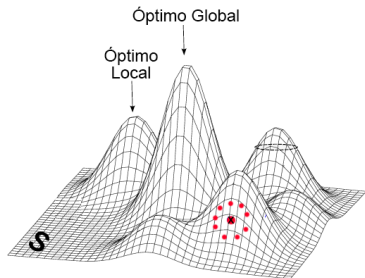
# Algoritmos Reparadores

- Los algoritmos reparadores parten desde una solución inicial ( $x$  en la imagen)



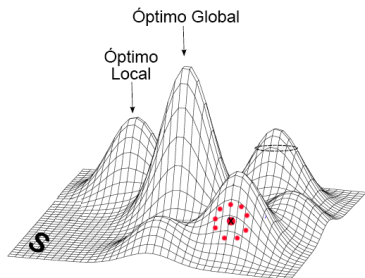


# Algoritmos Reparadores



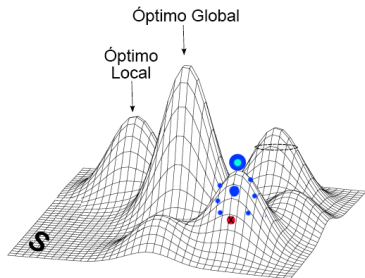
- ▶ Los algoritmos reparadores parten desde una solución inicial ( $x$  en la imagen)
- ▶ Para buscar otras soluciones, podemos aplicar un **movimiento** a  $x$

# Algoritmos Reparadores



- ▶ Los algoritmos reparadores parten desde una solución inicial ( $x$  en la imagen)
- ▶ Para buscar otras soluciones, podemos aplicar un **movimiento** a  $x$
- ▶ Todas las soluciones cercanas a  $x$ , respecto a un movimiento, forman el **vecindario** de  $x$

# Algoritmos Reparadores



- ▶ Los algoritmos reparadores parten desde una solución inicial ( $x$  en la imagen)
- ▶ Para buscar otras soluciones, podemos aplicar un **movimiento** a  $x$
- ▶ Todas las soluciones cercanas a  $x$ , respecto a un movimiento, forman el **vecindario** de  $x$
- ▶ Si el algoritmo se enfoca en mejorar la calidad de las soluciones, podemos decir que está **intensificando** la búsqueda

- ▶ Los algoritmos reparadores parten desde una solución inicial ( $x$  en la imagen)
- ▶ Para buscar otras soluciones, podemos aplicar un **movimiento** a  $x$
- ▶ Todas las soluciones cercanas a  $x$ , respecto a un movimiento, forman el **vecindario** de  $x$
- ▶ Si el algoritmo se enfoca en mejorar la calidad de las soluciones, podemos decir que está **intensificando** la búsqueda
- ▶ Si el algoritmo intenta visitar otras regiones del espacio de búsqueda, podemos decir que está **diversificando** la búsqueda

# Algoritmos Reparadores

## Definiciones

### ► Movimiento

Transformación aplicada a una solución candidata. Altera los valores asignados a algunas variables

- Bitflip:  $1111 \rightarrow 0111$

### ► Vecindario

El vecindario de una solución es el conjunto de soluciones generado por la aplicación del movimiento a dicha solución

- $\mathcal{N}$  define para cada solución candidata  $x \in \mathcal{S}$  un conjunto  $\mathcal{N}(x) \subseteq \mathcal{S}$
- $\mathcal{N}(x)$  son en algún sentido “cercanas” a  $x$ .
- Ejemplo con Bitflip aplicado a  $x = 1111$ 
  - $\mathcal{N}(x) = [0111, 1011, 1101, 1110]$

# Algoritmos Reparadores

## Definiciones

### ► Óptimo local

Sea  $(\mathcal{S}, f)$  un problema de optimización y  $\mathcal{N}$  la función vecindario. Una solución  $\hat{x} \in \mathcal{S}$  es un Óptimo Local con respecto a  $\mathcal{N}$  si

$$f(\hat{x}) \geq f(x), \forall x \in \mathcal{N}(\hat{x}).$$

- Un algoritmo que realiza búsqueda local, puede estancarse en algún óptimo local sin conocer otras soluciones

# Diversificación & Intensificación

Debe existir un compromiso entre dos factores:

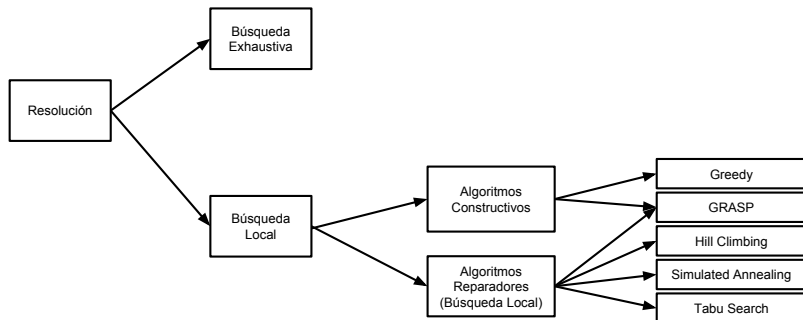
- ▶ *Intensificación*: Consiste en enfatizar el proceso de búsqueda en el vecindario actual de la solución candidata, con el objetivo de encontrar el óptimo local de la región.
  - ▶ *Explotación* del espacio de búsqueda, obteniendo las mejores soluciones
- ▶ *Diversificación*: Consiste en estrategias de salto a áreas inexploradas del espacio de búsqueda.
  - ▶ *Exploración* del espacio de soluciones, localizando regiones interesantes

# Manejo de Restricciones

- ▶ Espacio infactible:
  - ▶ Función de evaluación: Penalizar restricciones insatisfechas
- ▶ Espacio factible:
  - ▶ Representación: Representación que permita trabajar con soluciones factibles
  - ▶ Inicialización: Que permita construir soluciones factibles
  - ▶ Movimiento: Que mantenga la factibilidad de la solución
- ▶ Espacio infactible  $\Rightarrow$  Espacio factible
  - ▶ Función de evaluación: Penalizar mientras nos movemos en el espacio infactible
  - ▶ Movimiento: Que repare las soluciones (las vuelva factibles)



# Clasificación de Técnicas de Resolución



## Sección 2

# Algoritmos Constructivos

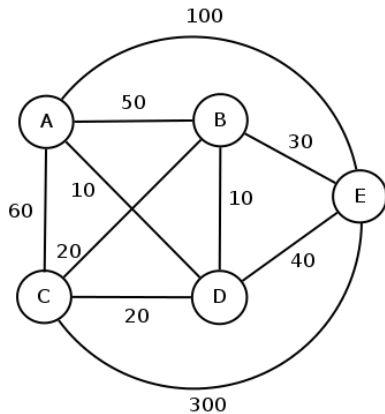
## Greedy

# Greedy

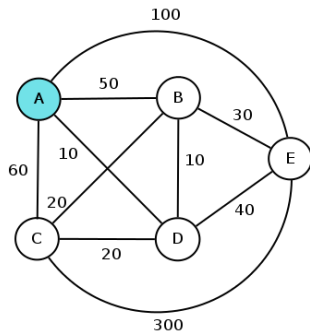
## Greedy

- ▶ Algoritmo constructivo que decide de manera localmente óptima qué componentes agregar en la solución candidata
- ▶ Para el problema del vendedor viajero:
  - ▶ Representación: Tour factible
  - ▶ Función Miope: Agregar la siguiente ciudad más cercana no visitada
  - ▶ Punto de Partida: Ciudad inicial A
  - ▶ Función de Evaluación: Largo del Tour

# Greedy

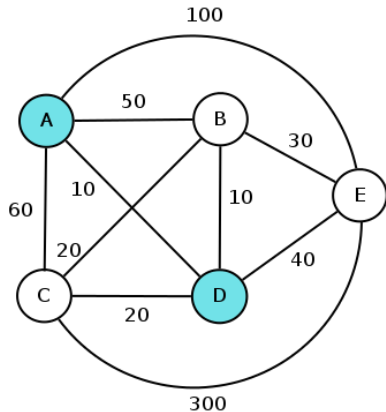


# Greedy



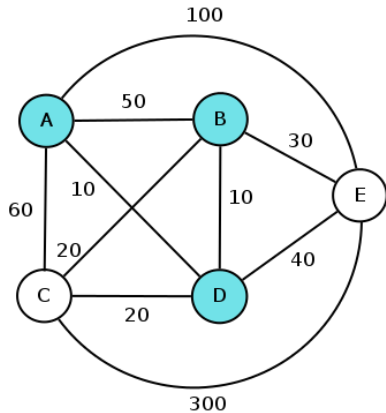
► Tour: A

# Greedy



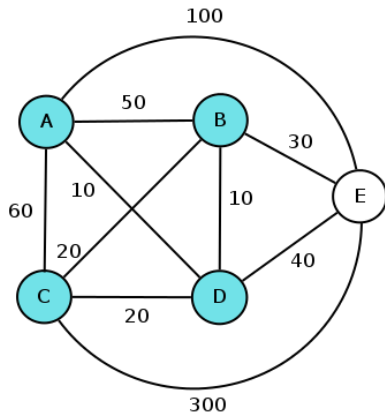
► Tour: A - D

# Greedy



► Tour: A - D - B

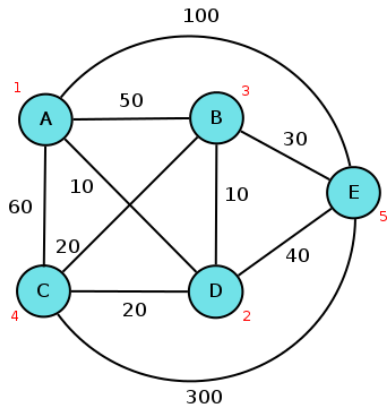
# Greedy



► Tour: A - D - B - C



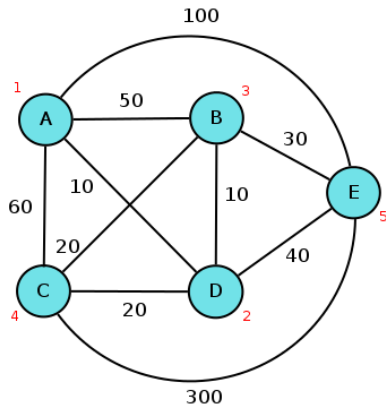
# Greedy



► Tour: A - D - B - C - E

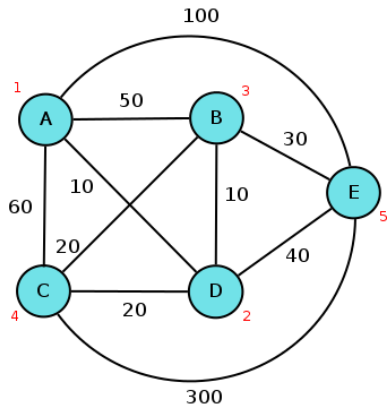
► Costo Tour = 440

# Greedy



- ▶ Tour: A - D - B - C - E
- ▶ Costo Tour = 440
- ▶ Tour: **D** - **B** - C - A - E
- ▶ Costo Tour = 230

# Greedy



- ▶ Tour: A - D - B - C - E
- ▶ Costo Tour = 440
- ▶ Tour: **D** - **B** - C - A - E
- ▶ Costo Tour = 230
- ▶ Tour: A - D - E - B - C
- ▶ Costo Tour = 160

## Algoritmos basados en construcción: Mochila

$$\text{máx } 18 \cdot x_1 + 25 \cdot x_2 + 11 \cdot x_3 + 14 \cdot x_4$$

$$\text{s.a. } 2 \cdot x_1 + 2 \cdot x_2 + x_3 + x_4 \leq 3$$

$$x_1, x_2, x_3, x_4 \in \{0, 1\}$$

- ▶ Representación: Lista binaria
- ▶ Función objetivo: Ganancia total
  - ▶ Función de evaluación?
- ▶ Punto de partida: Objeto 3
- ▶ Función miope: Agregar el siguiente objeto de mayor ganancia [factible?]

## Sección 3

# Algoritmos Reparadores

# Algoritmos basados en reparación

- ▶ Se mueven en el espacio de soluciones (c)
- ▶ Procesos iterativos que empiezan en una solución (c) y a través de modificaciones locales la va mejorando.
- ▶ Generalmente los algoritmos que construyen soluciones son más rápidos, pero sus resultados no siempre son buenos.

## Sección 3

### Algoritmos Reparadores

### Hill-Climbing

# Hill-Climbing

- ▶ Algoritmo que busca ir mejorando el valor de la función de evaluación de una solución candidata, al aplicar iterativamente un movimiento
- ▶ Usualmente, comienza con una solución candidata generada de manera aleatoria
- ▶ En cada iteración genera un vecindario aplicando el movimiento a la solución actual
- ▶ Si algún vecino mejora la calidad de la solución actual, éste reemplaza la solución actual
- ▶ En caso contrario, termina la ejecución



# Hill-Climbing

- ▶ Podemos identificar dos versiones de HC:
  - ▶ Alguna Mejora (*en inglés First Improvement*): Seleccionar el primer vecino que mejora la solución candidata actual
  - ▶ Mejor Mejora (*en inglés Best Improvement*): Selecciona el mejor vecino de todos, que mejora la solución candidata actual

# Pseudo-código (Best Improvement)

**Procedure** hill-climbing

$local \leftarrow FALSE$

$s_c \leftarrow$  select a point at random

**Repeat**

        select  $s_n$  the best quality point in  $\mathcal{N}(s_c)$

**If**  $f(s_n)$  is better than  $f(s_c)$  **Then**

$s_c \leftarrow s_n$

**Else**

$local \leftarrow TRUE$

**Until**  $local$

**End**

# Pseudo-código (First-Improvement Rule)

**Procedure** hill-climbing

*local*  $\leftarrow$  *FALSE*

$s_c \leftarrow$  select a point at random

*neighbor*  $\leftarrow$  0

**Repeat**

    generate  $s'_n$  a neighbor point in  $\mathcal{N}(s_c)$

*neighbor* ++

**If**  $f(s'_n)$  is better than  $f(s_c)$  **Then**

$s_c \leftarrow s'_n$

*neighbor*  $\leftarrow$  0

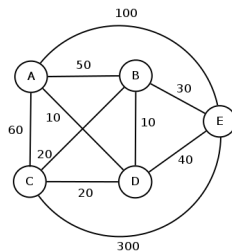
**If** *neighbor* == *max\_neighbors* **Then**

*local*  $\leftarrow$  *TRUE*

**Until** *local*

**End**

# Hill-Climbing



Tomando como solución inicial aleatoria A-E-D-B-C con costo 230 y utilizando Mejor Mejora, su vecindario usando un intercambio entre ciudades contiguas en el tour, sería:

1. E - A - D - B - C = 440
2. **A - D - E - B - C = 160**
3. A - E - B - D - C = 220
4. A - E - D - C - B = 210
5. C - E - D - B - A = 460

Si consideramos un algoritmo Alguna Mejora, para este caso, el primer vecino que mejora la solución es la número 2.

# Hill-Climbing

- ▶ Cuando HC encuentra la mejor solución candidata de la región, termina su ejecución
- ▶ Hill Climbing no define ninguna estrategia para escapar de óptimos locales

## Ejemplo

- Considerando el problema de la mochila

$$\begin{aligned} \text{máx } & 18 \cdot x_1 + 25 \cdot x_2 + 11 \cdot x_3 + 14 \cdot x_4 \\ \text{s.a. } & 2 \cdot x_1 + 2 \cdot x_2 + x_3 + x_4 \leq 3 \\ & x_1, x_2, x_3, x_4 \in \{0, 1\} \end{aligned}$$

- Proponga una representación acorde al problema
- Proponga un movimiento acorde al problema
- Describa el proceso de búsqueda realizado por hill-climbing<sup>1</sup>

---

<sup>1</sup>1000, 0000

# Hill-Climbing

- ▶ Cuando HC encuentra la mejor solución candidata de la región, termina su ejecución
- ▶ Hill Climbing no define ninguna estrategia para escapar de óptimos locales
- ▶ Una solución a este problema es Hill Climbing + Restart

## Sección 3

### Algoritmos Reparadores

### Hill-Climbing+Restart



# Hill-Climbing + Restart

- ▶ La idea es re-comenzar el algoritmo con una solución nueva cuando éste se ha quedado estancado.

# Pseudo-código

**Procedure** hill-climbing

$t \leftarrow 0$

**initialize**  $s_{best}$

**Repeat**

$local \leftarrow FALSE$

$s_c \leftarrow$  select a point at random

evaluate  $s_c$

**Repeat**

select  $s_n$  the best quality point in  $\mathcal{N}(s_c)$

**If**  $f(s_n)$  is better than  $f(s_c)$  **Then**

$s_c \leftarrow s_n$

**Else**

$local \leftarrow TRUE$

**Until**  $local$

$t \leftarrow t + 1$

**if**  $s_c$  is better than  $s_{best}$  **then**

$s_{best} \leftarrow s_c$

**Until**  $t = MAX$

# Hill-Climbing + Restart

- ▶ Desventaja de H-C W/ Restart: Pérdida de información valiosa del proceso de búsqueda actual.

# Hill-Climbing + Restart

- ▶ Desventaja de H-C W/ Restart: Pérdida de información valiosa del proceso de búsqueda actual.
- ▶ Una alternativa es aceptar movimientos que empeoran la calidad de la solución actual.

# Hill-Climbing + Restart

- ▶ Desventaja de H-C W/ Restart: Pérdida de información valiosa del proceso de búsqueda actual.
- ▶ Una alternativa es aceptar movimientos que empeoran la calidad de la solución actual.
- ▶ La desventaja de aceptar soluciones de peor calidad es que se pueden producir ciclos en la búsqueda.