Heuristicos

Optimización combinatoria

¿Qué es?

Tecnica de investigación de operaciones.

De que se encarga

De encontrar la mejor solución posible(maxima o minima) en un conjunto de soluciones finita.

Heuristicos

¿Qué son?

Son procedimientos eficientes

para encontrar

buenas soluciones aunque no sean optimas

Se consideran

atajos, donde no es necesario explorar todas las soluciones, por que podemos explorar el área donde consideramos se deben encontrar las buenas soluciones.

Los usamos cuando

1	*No se conoce un metodo exacto para resolverlo*
2	*Existe pero es muy costoso, por lo tanto no se puede utilizar*
3	*El problema es muy complejo*
4	*Dar una solución a un método exacto*

Metaheurísticos

¿Qué son?

Métodos heuristicos mas sofisticados.

De que se encarga

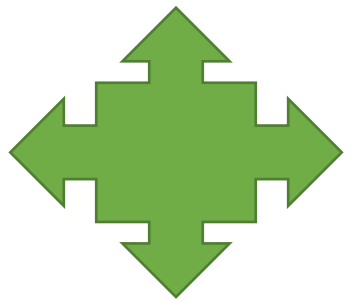
de resolver problemas de optimización combinatoria mas compleja, donde los heruristicos no son capaces de obtener soluciones aceptables.

Sin embargo

Se tiene que justificar su uso, es decir que no se pueda utilizar una solución por metodo exacto.

Ejemplos

1	*Busqueda tabu*
2	*Recocido simulado*
3	*Grasp*
4	*Busqueda dispersa*
5	*Algoritmos bioinspirados*
6	*Colonia de hormigas*
7	*Ejambre de particulas*



Optimización combinatoria

¿Qué es?

De que se encarga

Heuristicos

¿Qué son?

para encontrar

Se consideran

Los usamos cuando

1
2
3
4

Metaheurísticos

¿Qué son?

De que se encarga

Sin embargo

Ejemplos

1	*Busqueda tabu*
2	*Recocido simulado*
3	
4	*Busqueda dispersa*
5	*Algoritmos bioinspirados*
6	
7	*Ejambre de particulas*

Existen 2 tipos de Heuristicos

Métodos constructivos

Tenemos al inicio todos los datos, al final tendremos una solución

La solución se construye paso a paso.

Existen 2 tipos

Métodos de búsqueda local

Son métodos de mejora

Parten de una solución

Se encuentra una mejor solución dada a la inicial

Greedy

consiste en elegir la opción óptima en cada paso

Semi-Greedy

Consiste en seleccionar aleatoriamente un elemento dado una lista restringida de candidatos (LRC) en cada paso

Existen 2 tipos

Lista restringida de cardinalidad

se pone un limite k a la lista de candidatos en cada iteración

Lista restringida por valor

En esta lista **no** se tiene un **tamaño fijo** entran todos los **candidatos** que **cumplen** la **desigualdad**.

$$c_{min} \leq c(e) \leq c_{min} + \alpha(c_{max} - c_{min})$$

Donde

1	c_{min}	*distancia minima*
2	c_{max}	*distancia maxima*
3	$c(e)$	*candidato escogido*
4	α	*valor de relacion alpha*

Búsqueda local

Debido

A que los algoritmos constructivos suelen tener baja efectividad.

Se utilizan

Algoritmos de búsqueda local, para mejorar la solución

Estos realizan

movimientos, los cuales producen una solución cercana a la obtenida

Ejemplo

1. *Intercambio entre 2 nodos*
2. *Inserción hacia adelante y atras*
3. *2-opt (cambiar 2 aristas)*

Este termina

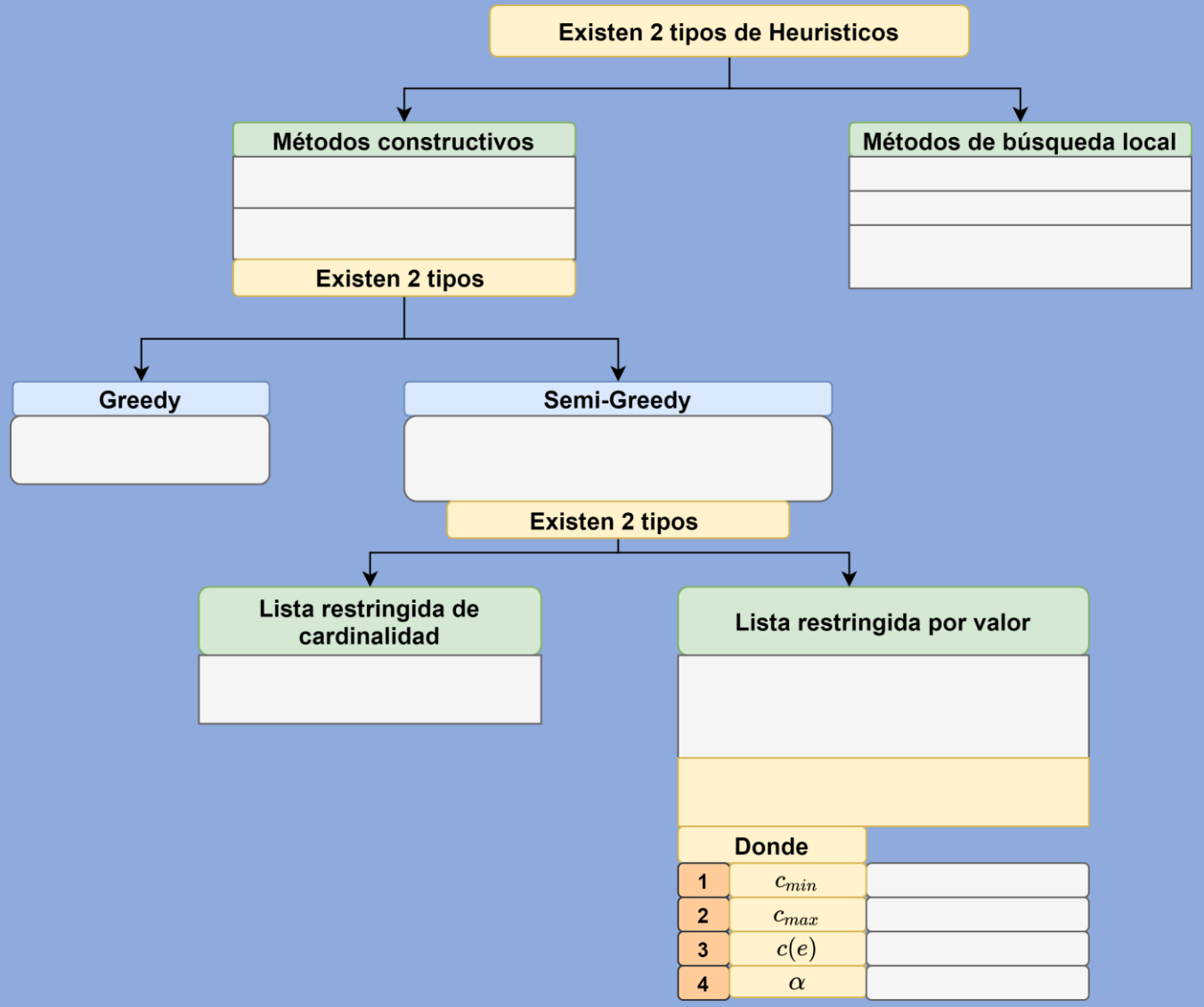
Cuando ya no es posible mejorar la solución actual

Obteniendo

Optimo local

Existen 2 estrategias

First improvement	Best improvement
1 ***Selecciona la primera solución***	1 ***Evalua todas las soluciones en el vecindario***
2 *Se repite la búsqueda con nuevo valor*	2 *Se repite la búsqueda con nuevo valor*
3 *Se para cuando no se encuentra una mejor solución*	3 *Se para cuando no se encuentra una mejor solución*



Búsqueda local

Debido

Se utilizan

Estos realizan

Ejemplo

1

2

3

Este termina

Obteniendo

Existen 2 estrategias

First improvement		Best improvement	
1		1	
2		2	*Se repite la búsqueda con nuevo valor*
3	*Se para cuando no se encuentra una mejor solución*	3	

Problema de la mochila

Tiene 3 heuristicos

1. *Solución por valor*
   1. Empezamos con el valor más alto en cada iteración
   2. Paramos cuando ya no cabe ningun otro objeto.
2. *Solución por peso*
   1. Empezamos con el objeto de menor peso.
   2. Paramos cuando ya no cabe ningun otro objeto.
3. *Solución por Valor/Peso*
   1. Empezamos con objeto que tenga mayor relación valor/peso
   2. Paramos cuando ya no cabe ningun otro objeto.

GAP

Este valor se usa para medir la diferencia entre 2 resultados.

$$GAP = 100 - (\frac{(R_t * 100)}{R_o})$$

Problema del viajero

Tiene 4 heuristicos

1. *Algoritmo vecino más cercano* — $T = \{V_1, V_2, \ldots, V_k\}$
   1. Iniciamos con una ciudad arbitraria ***V_1*** *de* ***n*** *ciudades*
   2. Seleccionar el vertice mas cercano a ***V_k***
   3. Mientras ***k < n***, repetir paso 2, Si ***k = n*** parar
2. *Algoritmo inserción más cercana* — $T = \{V_1, V_2, \ldots, V_k\}$
   1. Iniciamos con una ciudad arbitraria ***V_1*** *de* ***n*** *ciudades*
   2. Seleccionar el vertice mas cercano a ***V_1*** y agregalo al subtour ***T***
   3. Encuentra un nodo ***k*** que este lo mas cerca posible de cualquier nodo ***T***
   4. Encuenta un arco ***(i,j)*** de ***T,*** para el cual la insercion de ***k*** da el menor incremento en longitud, e insertelo — $\Delta = c_{i-k} + c_{k-j} + c_{i-j}$
   5. vaya al paso 3 hasta que se haya formado el Tour
3. *Algoritmo inserción más lejana* — $T = \{V_1, V_2, \ldots, V_k\}$
   1. Iniciamos con una ciudad arbitraria ***V_1*** *de* ***n*** *ciudades*
   2. Seleccionar el vertice mas lejano a ***V_1*** y agregalo al subtour ***T***
   3. Encuentra un nodo ***k*** que este lo mas lejano posible de cualquier nodo ***T***
   4. Encuenta un arco ***(i,j)*** de ***T,*** para el cual la insercion de ***k*** da el menor incremento en longitud, e insertelo — $\Delta = c_{i-k} + c_{k-j} + c_{i-j}$
   5. vaya al paso 3 hasta que se haya formado el Tour
4. *Algoritmo inserción más barata* — $T = \{V_1, V_2, \ldots, V_k\}$
   1. Iniciamos con una ciudad arbitraria ***V_1*** *de* ***n*** *ciudades*
   2. Seleccionar el vertice mas cercano ***V_1*** y agregalo al subtour ***T***
   3. Encuenta un arco ***(i,j)*** de ***T,*** y un nodo ***k*** que no este en ***T***, y busca el que de menor incremento a la longitud, e insertelo. — $\Delta = c_{i-k} + c_{k-j} + c_{i-j}$
   4. vaya al paso 3 hasta que se haya formado el Tour

Problema de la mochila

Tiene 3 heuristicos

1.
   1.
   2. Paramos cuando ya no cabe ningun otro objeto.
2.
   1.
   2. Paramos cuando ya no cabe ningun otro objeto.
3.
   1.
   2. Paramos cuando ya no cabe ningun otro objeto.

GAP

Este valor se usa para medir la diferencia entre 2 resultados.

Problema del viajero

Tiene 4 heuristicos

1. $T = \{V_1, V_2, \ldots, V_k\}$
   1. Iniciamos con una ciudad arbitraria ***V1*** *de* ***n*** *ciudades*
   2.
   3. Mientras ***k < n***, repetir paso 2, Si ***k = n*** parar
2. $T = \{V_1, V_2, \ldots, V_k\}$
   1. Iniciamos con una ciudad arbitraria ***V1*** *de* ***n*** *ciudades*
   2.
   3.
   4. $\Delta = c_{i-k} + c_{k-j} + c_{i-j}$
   5. vaya al paso 3 hasta que se haya formado el Tour
3. $T = \{V_1, V_2, \ldots, V_k\}$
   1. Iniciamos con una ciudad arbitraria ***V1*** *de* ***n*** *ciudades*
   2.
   3.
   4. $\Delta = c_{i-k} + c_{k-j} + c_{i-j}$
   5. vaya al paso 3 hasta que se haya formado el Tour
4. $T = \{V_1, V_2, \ldots, V_k\}$
   1. Iniciamos con una ciudad arbitraria ***V1*** *de* ***n*** *ciudades*
   2.
   3. $\Delta = c_{i-k} + c_{k-j} + c_{i-j}$
   4. vaya al paso 3 hasta que se haya formado el Tour

GRASP

G	***Greedy***
R	***Randomized***
A	***Adaptative***
S	***Search***
P	***Procedure***

Deinicion

metodo multiarranque, en el que cada iteracion

Consiste

En la construccion de una solución miope aleatorizada

Seguida

de una busqueda local, usando la solución construida como punto inicial

Esto se repite

La mejor solución se devuelve

Lo compone

1	*Función de evalución miope*
2	*Procedimiento de eleccion al azar*
3	*Proceso de actualización adaptativo*
4	*Postprocesamiento*

ESQUEMA GRASP

1	***While (no se cumpla criterio de parada)***
2	***Construir solución -> x***
3	***Construir solucion de manera secuencial, utilizando la función de evaluación greedy y las listas restringidas de candidatos***
4	***Mejorar solución (x) —> x'*** (Busqueda local)
5	***if(x' mejor que x)***
5	***x = x'***
6	***\end while***

GRASP

G	
R	
A	
S	
P	

Deinicion

Consiste

Seguida

Esto se repite

Lo compone

1	
2	
3	
4	

ESQUEMA GRASP

1	***While (no se cumpla criterio de parada)***
2	
3	
4	
5	
5	
6	***\end while***