Cómputo Evolutivo

Stalin Muñoz Gutiérrez

Centro de Ciencias de la Complejidad
Universidad Nacional Autónoma de Mexico (UNAM)

Cómputo Evolutivo

2019-08-29

Cómputo Evolutivo

Stalin Muñoz Gutiérrez

Centro de Ciencias de la Complejidad
Universidad Nacional Autónoma de Mexico (UNAM)

El cómputo evolutivo es una área de investigación con amplias y muy diversas aplicaciones para la investigación científica y la industria. Este taller es una introducción al área. Aprenderemos de los conceptos básicos dentro del área enfocándonos a su aplicabilidad para la solución de problemas prácticos, más que a la investigación algorítmica o su uso para el estudio de procesos evolutivos de la naturaleza. Los *organismos* que evolucionaremos serán artificiales. Y aunque utilizamos metáforas basadas en procesos naturales no los abordaremos desde el punto de vista de investigación de las teorías de evolución natural.

Resolver problemas por evolución

El cómputo evolutivo es una manera de resolver problemas enfocándose en las preguntas:

- ¿qué resolver?
- ¿qué es una buena solución al problema?

Los algoritmos de cómputo evolutivo son parte de una categoría de algoritmos denominados *algoritmos de búsqueda metaheurísticos*.

Resolver problemas por evolución

El cómputo evolutivo es una manera de resolver problemas enfocándose en las preguntas:
- ¿qué resolver?
- ¿qué es una buena solución al problema?

Los algoritmos de cómputo evolutivo son parte de una categoría de algoritmos denominados *algoritmos de búsqueda metaheurísticos*.

Cuando aplicamos algoritmos genéticos a la resolución de problemas nos va a interesar describir o definir en qué consisten las soluciones y en poder determinar si las soluciones son buenas o malas.

Los algoritmos de cómputo evolutivo pertenecen a una categoría de algoritmo denominados metaheurísticos. Comenzaremos por describir brevemente algunos conceptos relevantes para entender la terminología del área.

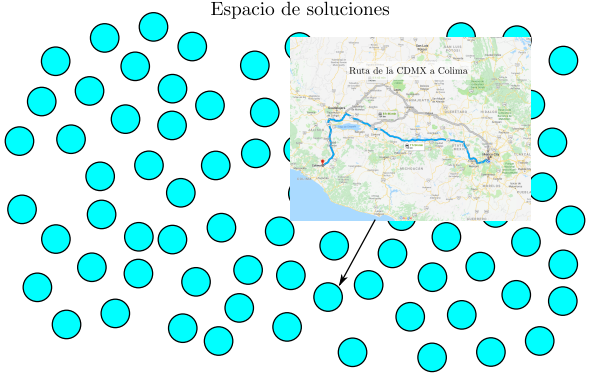
Espacio de soluciones

Hoy trataremos de un tipo especial de algoritmos de búsqueda denominados algoritmos metaheurísticos.

Espacio de soluciones

Espacio de soluciones a un problema

El conjunto donde existen todas las posibles soluciones a un problema.



Antes de abordar los algoritmos metaheurísticos debemos definir algunos conceptos importantes.

Entenderemos al espacio de soluciones a un problema como un conjunto donde existen todas las posibles soluciones a dicho problema.

Aquí los círculos representarían posibles soluciones a un problema.

Por ejemplo en el caso del problema de encontrar una ruta en un mapa, cada círculo representaría una posible secuencia de ciudades, independientemente de si es una ruta correcta o no.

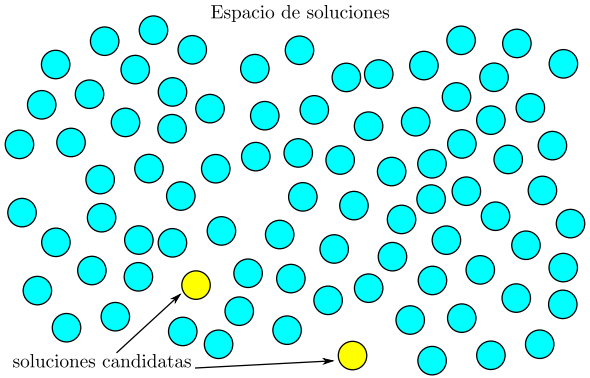
Para la mayoría de los algoritmos metaheurísticos no es importante tener accerso a un grafo de estados-acciones como en el caso de algoritmos clásicos de búsqueda ciega o algoritmos de búsqueda informados.

En los metaheurísticos basta con describir el espacio de soluciones y definir operadores que permitan explorar este espacio.

Espacio de soluciones

Soluciones candidatas

Cualquier elemento del espacio de soluciones.

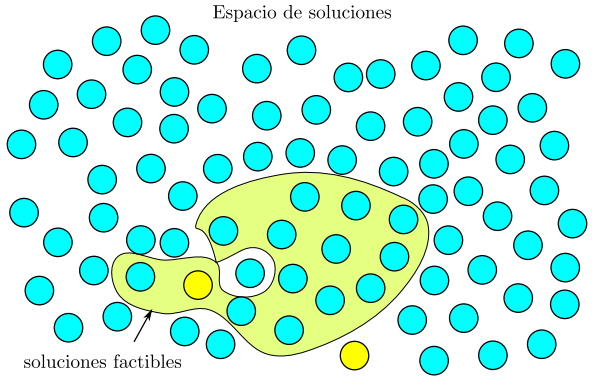


Todo elemento del espacio de soluciones lo denominamos una solución candidata.

Espacio de soluciones

Soluciones factibles

Satisfacen un conjunto de restricciones.



Aquellas soluciones candidatas que cumplen con las restricciones del problema a resolver se denominan *soluciones factibles*.

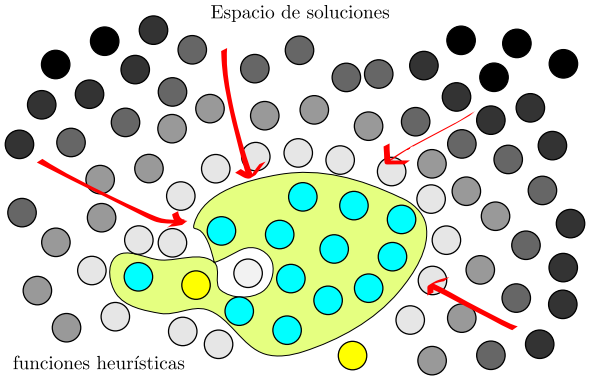
Las soluciones factibles son las soluciones de interés.

En el ejemplo de encontrar una ruta en un mapa, las soluciones factibles son aquellas rutas que empiezan en la ciudad de origen y terminan en el destino deseado.

Espacio de soluciones

Funciones heurísticas

Guían al algoritmo hacia las regiones factibles.

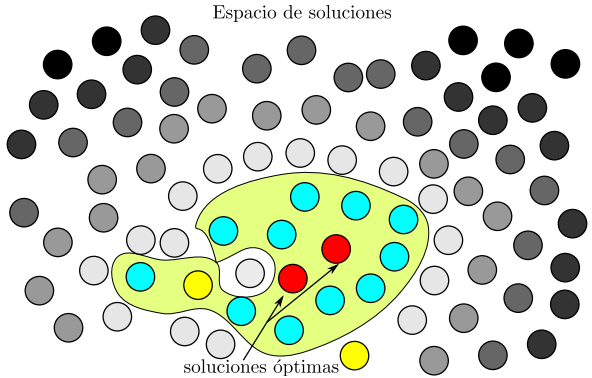


Los algoritmos metaheurísticos también se valdrán de funciones heurísticas que explotan conocimiento del dominio del problema. Una buena heurística guiará al algoritmo hacia las regiones de interés, esto es, hacía las soluciones factibles.

Espacio de soluciones

Soluciones óptimas

Las mejores soluciones factibles posibles.



Típicamente, se desea un algoritmo que busque las mejores soluciones posibles.

En nuestro ejemplo de las rutas en un mapa, nos interesan las rutas más cortas en distancia o las más rápidas en tiempo.

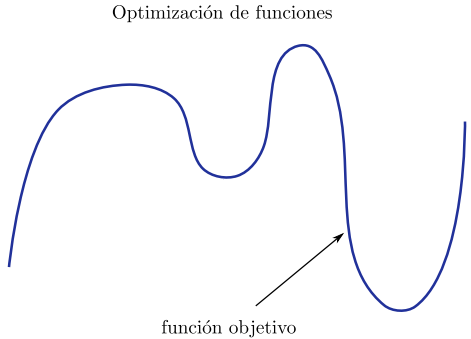
Por esta razón los algoritmos metaheurísticos muchas veces se denominan algoritmos de optimización metaheurísticos.

Resolver un problema, en este caso, es encontrar el mínimo o el máximo de cierta función objetivo.

Optimización de funciones

Problema de optimización

Encontrar los valores extremos, máximos o mínimos, de una función objetivo.



Optimización de funciones
Problema de optimización
Encontrar los valores extremos, máximos o mínimos, de una función objetivo.

Una vez que planteamos el problema como un problema de optimización, el algoritmo metaheurístico tratará de encontrar los valores extremos de la función objetivo.

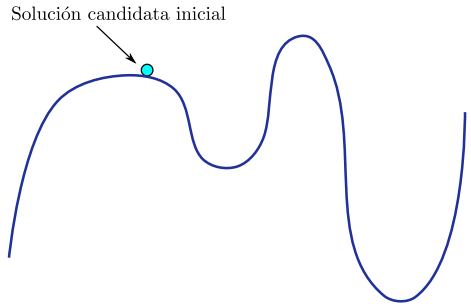
Esto es el valor mínimo o máximo de la función.

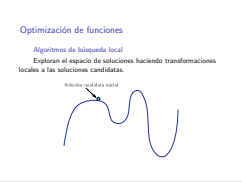
La función puede ser una función escalar, es decir un solo número o puede ser una función multidimensional, en cuyo caso se denomina *problema de optimización multi-objetivo*.

Optimización de funciones

Algoritmos de búsqueda local

Exploran el espacio de soluciones haciendo transformaciones locales a las soluciones candidatas.





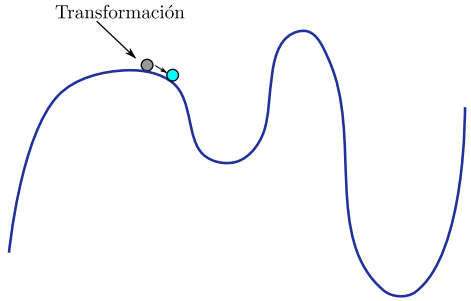
Un tipo de estrategia para buscar soluciones es el utilizado en los denominados algoritmos de búsqueda local.

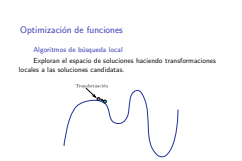
Un algoritmo de búsqueda local comienza con una solución candidata inicial, típicamente generada de manera aleatoria.

Optimización de funciones

Algoritmos de búsqueda local

Exploran el espacio de soluciones haciendo transformaciones locales a las soluciones candidatas.





Este tipo de algoritmos es muy general en su aplicabilidad.

No asume propiedades matemáticas de la función objetivo, como en el caso de métodos clásicos basados en descenso de gradiente, donde se requiere que la función sea derivable.

La búsqueda local puede aplicarse aún sin existir una función matemática explícita de la función objetivo.

Podemos definir la transformación en términos muy generales, como una transformación que resulta en una solución que se parece mucho, pero no es igual, a la solución candidata original.

Como las soluciones se parecen mucho, decimos que son soluciones vecinas. Nuevamente, no se requiere de una definición matemática de vecindad para una solución.

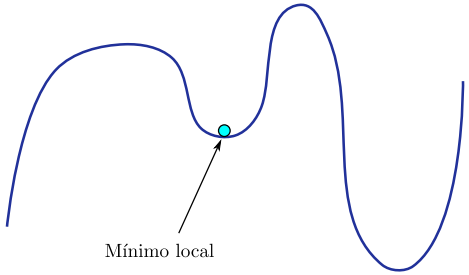
Las transformaciones pueden estar definidas de manera algorítmica.

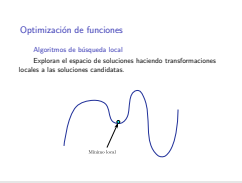
Generalmente las transformaciones se hacen con algún criterio aleatorio.

Optimización de funciones

Algoritmos de búsqueda local

Exploran el espacio de soluciones haciendo transformaciones locales a las soluciones candidatas.





El algoritmo de búsqueda local, transforma de manera iterada la solución inicial, decidiendo la dirección de búsqueda en función de la evaluación de soluciones vecinas.

Esto hace que el algoritmo vaya convergiendo a las soluciones que minimizan o maximizan el valor de la función objetivo.

Si la transformación resulta en una mejora. El algoritmo puede decidir aceptar la solución. Si siempre la acepta, entonces se denomina algoritmo de ascención de colinas (Hill climbing).

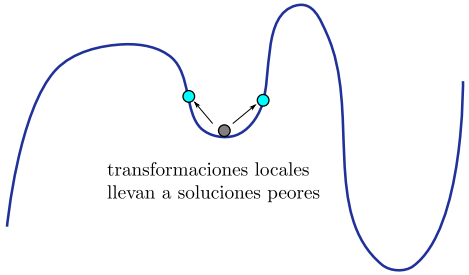
Si el algoritmo explora toda la vecindad antes de decidir sobre la dirección a tomar, el algoritmo se denomina de ascención de colinas de máxima pendiente.

Sin embargo, nada garantiza que convergerá al mínimo global.

Optimización de funciones

Algoritmos de búsqueda local

Exploran el espacio de soluciones haciendo transformaciones locales a las soluciones candidatas.



Optimización de funciones
Algoritmos de búsqueda local
Exploran el espacio de soluciones haciendo transformaciones locales a las soluciones candidatas.

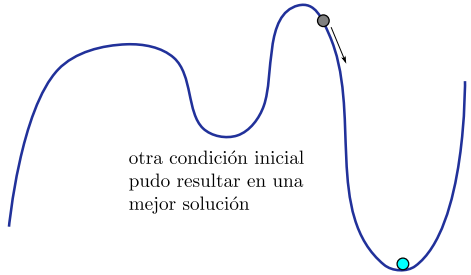
Aquí observamos que el algoritmo converge a un mínimo de la función objetivo.

Cualquier movimiento local, empeorará el valor de la función objetivo.

Optimización de funciones

Algoritmos de búsqueda local

Exploran el espacio de soluciones haciendo transformaciones locales a las soluciones candidatas.



Optimización de funciones
Algoritmos de búsqueda local
Exploran el espacio de soluciones haciendo transformaciones locales a las soluciones candidatas.

Si el algoritmo hubiera empezado en otra solución candidata, es posible que encontrara una mejor solución.

Una estrategia posible para los algoritmos de búsqueda local es lanzar múltiples búsquedas con diferentes condiciones iniciales.

El algoritmo de recocido simulado (SA)

Un algoritmo capaz de escapar óptimos locales

El algoritmo de recocido (o templado) simulado (*simulated annealing*) es un algoritmo metaheurístico capaz de escapar de óptimos locales. Esta inspirado en un proceso de calentamiento y enfriamiento controlado en el cual un sólido puede alcanzar un arreglo cristalino muy regular al ser enfriado de manera suficientemente lenta.

El algoritmo de recocido simulado (SA)

Un algoritmo capaz de escapar óptimos locales
El algoritmo de recocido (o templado) simulado (*simulated annealing*) es un algoritmo metaheurístico capaz de escapar de óptimos locales. Esta inspirado en un proceso de calentamiento y enfriamiento controlado en el cual un sólido puede alcanzar un arreglo cristalino muy regular al ser enfriado de manera suficientemente lenta.

El primer algoritmo metaheurístico que presentaremos es el algoritmo de recocido o templado simulado, al que denominaremos *SA* por las siglas de su nombre en inglés *Simulated Annealing*.

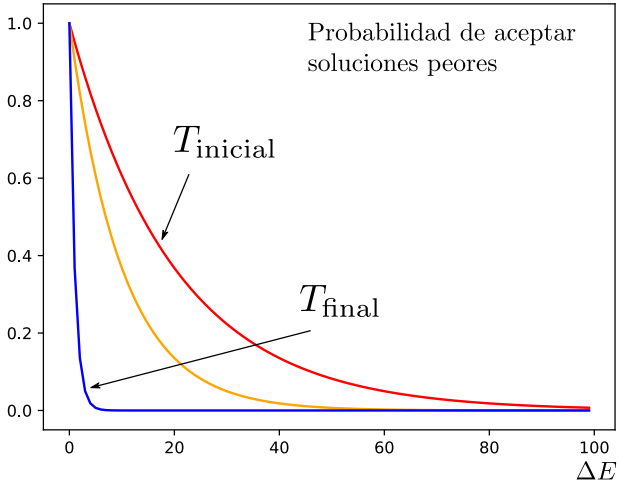
El algoritmo fue propuesto por Kirkpatrick, Gelatt y Vecchi quienes en su revolucionario artículo de 1983 en la revista Science mostraron que SA puede converger a soluciones óptimas de un problema de optimización combinatorio si se le deja correr el tiempo suficiente.

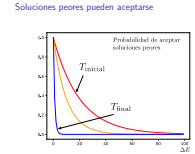
Como muchos de los algoritmos de optimización metaheurísticos, esta inspirado en un proceso de optimización observado en la naturaleza. En este caso en un proceso físico.

En un proceso de templado, el material se calienta más allá de un punto de recristalización. Después se enfria de manera controlada.

Desde el punto de vista del algoritmo de búsqueda existirá una variable de temperatura que decidirá el régimen de exploración del algoritmo.

Soluciones peores pueden aceptarse





La característica más interesante del algoritmo de templado simulado es que, a diferencia de un algoritmo ascensor de colinas estándar que puede fácilmente quedar atrapado en un óptimo local, SA no siempre se dirige en la dirección de máximo beneficio.

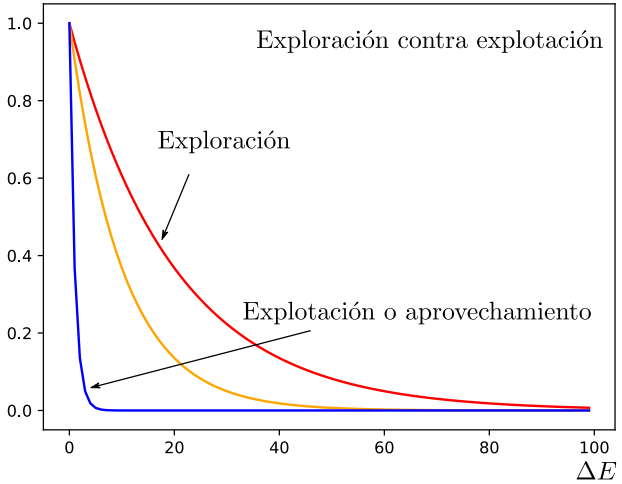
El algoritmo puede acepetar soluciones de mayor energía con cierta probabilidad.

La probabilidad de aceptar una solución que es peor que la solución candidata actual, esta dada por la distribución de Boltzman.

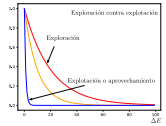
Aquí vemos como cambia la distribución de probabilidad para aceptar estados de mayor energía como una función de la temperatura.

La temperatura es alta al comenzar las iteraciones, T_{inicial}, y es baja al finalizar las iteraciones, T_{final}.

Soluciones peores pueden aceptarse



Soluciones peores pueden aceptarse



En la gráfica se ilustra que a temperaturas altas,
esto es, en las primeras iteraciones,
la probabilidad cae lentamente, entonces la probabilidad de aceptar soluciones de mayor energía es alta.

Por lo cual, al inicio de las iteraciones el algoritmo esta en un régimen de exploración.

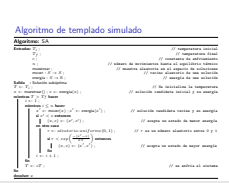
Conforme el sistema se enfria, la distribución es cada vez más estricta, indicando que se aceptará un número menor de soluciones que tengan mayor energía.

Al alcanzar la temperatura final, el sistema esta frio. La probabilidad de aceptar soluciones de mayor energía es casi nula. El algoritmo se encuentra en un régimen de explotación o aprovechamiento. En este régimen se comporta como un ascensor de colinas.

Algoritmo de templado simulado

Algoritmo: SA

Entradas: T_i ; // temperatura inicial
T_f ; // temperatura final
c ; // constante de enfriamiento
n ; // número de movimientos hasta el equilibrio térmico
muestrear ; // muestra aleatoria en el espacio de soluciones
mover : $S \to S$; // vecino aleatorio de una solución
energia : $S \to \mathbb{R}$; // energía de una solución
Salida : Solución subóptima
$T \leftarrow T_i$; // Se inicializa la temperatura
$x \leftarrow$ muestrear() ; $e \leftarrow$ energia(x) ; // solución candidata inicial y su energía
mientras $T > T_f$ **hacer**
  $i \leftarrow 1$;
  mientras $i \leq n$ **hacer**
    $x' \leftarrow$ mover(x) ; $e' \leftarrow$ energia(x') ; // solución candidata vecina y su energía
    si $e' < e$ **entonces**
      $(x, e) \leftarrow (x', e')$; // acepta un estado de menor energía
    en otro caso
      $r \leftarrow aleatorio.uniforme(0, 1)$; // r es un número aleatorio entre 0 y 1
      si $r < exp\left(\frac{-(e'-e)}{kT}\right)$ **entonces**
        $(x, e) \leftarrow (x', e')$; // acepta un estado de mayor energía
      fin
    $i \leftarrow i + 1$;
  fin
  $T \leftarrow cT$; // se enfria el sistema
fin
devolver x

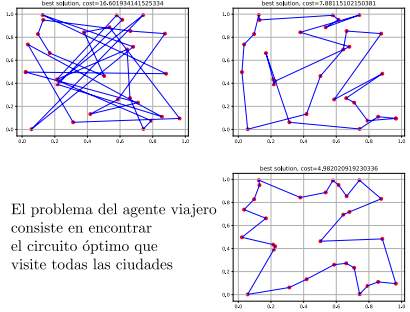


Presentamos el algoritmo SA.

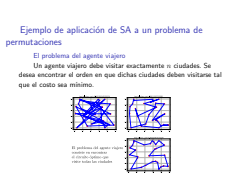
Ejemplo de aplicación de SA a un problema de permutaciones

El problema del agente viajero

Un agente viajero debe visitar exactamente n ciudades. Se desea encontrar el orden en que dichas ciudades deben visitarse tal que el costo sea mínimo.



El problema del agente viajero consiste en encontrar el circuito óptimo que visite todas las ciudades



Vamos a presentar un problema interesante y computacionalmente dificil de resolver.

El problema del agente viajero.

Un agente viajero, desea encontrar un circuito óptimo para visitar exactamente n ciudades, es decir, una secuencia ordenada de ciudades que minimice el costo de la ruta.

El espacio de posibles rutas es muy grande, es n factorial.

En la parte superior izquierda vemos una ruta arbitraria para un problema con 30 ciudades, tiene un costo alto, pues hay muchos cruces con distancias muy grandes.

Una buena ruta no tiene cruces, como la mostrada en la grafica inferior.

El algoritmo SA, teoricamente puede encontrar la solución óptima, sin embargo en la práctica el tiempo que le tomaría encontrar la ruta óptima es muy grande, por lo que sólo podemos garantizar que la solución será subóptima.

Soluciones candidatas y vecinas

Solución candidata

Una solución candidata es una permutación de ciudades:

$x = \begin{bmatrix} B & C & G & H & J & D & A & E & G & F \end{bmatrix}$

Solución vecina

1. intercambiando dos ciudades adjacentes:

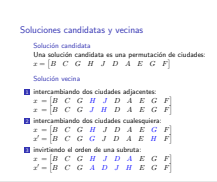
$x = \begin{bmatrix} B & C & G & H & J & D & A & E & G & F \end{bmatrix}$
$x = \begin{bmatrix} B & C & G & J & H & D & A & E & G & F \end{bmatrix}$

2. intercambiando dos ciudades cualesquiera:

$x = \begin{bmatrix} B & C & G & H & J & D & A & E & G & F \end{bmatrix}$
$x' = \begin{bmatrix} B & C & G & G & J & D & A & E & H & F \end{bmatrix}$

3. invirtiendo el orden de una subruta:

$x = \begin{bmatrix} B & C & G & H & J & D & A & E & G & F \end{bmatrix}$
$x' = \begin{bmatrix} B & C & G & A & D & J & H & E & G & F \end{bmatrix}$



En el problema del agente viajero una solución candidata es una permutación de ciudades.

En este ejemplo identificamos cada ciudad con una letra distinta.

Para encontrar soluciones vecinas, podemos definir un operador de intercambio de ciudades.

Aleatoriamente seleccionamos dos ciudades e intercambiamos su posición en la ruta.

Esta operación también la podemos restringir para ciudades que son adyacentes en la ruta.

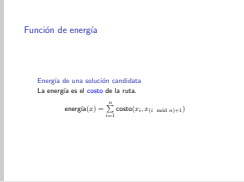
Otra estrategia para generar permutaciones vecinas es invertir el orden de la secuencia en una subruta aleatoria.

Función de energía

Energía de una solución candidata

La energía es el costo de la ruta.

$$\text{energía}(x) = \sum_{i=1}^{n} \text{costo}(x_i, x_{(i \mod n)+1})$$



Finalmente la energía de la solución es simplemente el costo sobre la trayectoria. En este caso la suma de los costos de cada transición.

Búsquedas metaheurísticas

[Gendreau y Potvin 2010; *Handbook of metaheuristics*]

Las metaheurísticas, son métodos de solución que orquestan procedimientos de mejora local con estrategias de alto nivel para crear un proceso capaz de escapar de un óptimo local y desempeñar una búsqueda robusta en el espacio de soluciones.

Búsquedas metaheurísticas

[Gendreau y Potvin 2010; *Handbook of metaheuristics*]
Las metaheurísticas, son métodos de solución que orquestan procedimientos de mejora local con estrategias de alto nivel para crear un proceso capaz de escapar de un óptimo local y desempeñar una búsqueda robusta en el espacio de soluciones.

El término *metaheurística* nombra a un número muy amplio de algoritmos de búsqueda.

Gendreau y Potvin las definen como métodos de solución que combinan procedimientos de mejora local con estrategias de alto nivel para evitar la convergencia temprana a soluciones subóptimas en un espacio de soluciones.

Búsquedas metaheurísticas

[Gendreau y Potvin 2010; *Handbook of metaheuristics*]

Las metaheurísticas, son métodos de solución que orquestan procedimientos de mejora local con estrategias de alto nivel para crear un proceso capaz de escapar de un óptimo local y desempeñar una búsqueda robusta en el espacio de soluciones.

El término fue acuñado por Fred Glover en 1986 para categorizar su algoritmo *Búsqueda Tabú* (*Tabu Search (TS)*).

Búsquedas metaheurísticas

[Gendreau y Potvin 2010; *Handbook of metaheuristics*]
Las metaheurísticas, son métodos de solución que orquestan procedimientos de mejora local con estrategias de alto nivel para crear un proceso capaz de escapar de un óptimo local y desempeñar una búsqueda robusta en el espacio de soluciones.
El término fue acuñado por Fred Glover en 1986 para categorizar su algoritmo *Búsqueda Tabú* (*Tabu Search (TS)*).

El nombre fue acuñado por Fred Glover en su artículo de 1986: *Future paths to integer programming and links to artificial intelligence* y fue usado para describir el algoritmo *Búsqueda Tabú*, al que denominamos TS, siglas en inglés de *Tabu Search*.

Glover nos dice que TS es una metaheurística superimpuesta en otra heurística.

En la búsqueda Tabú se prohibe volver a visitar ciertos estados almacenados en una memoria, lo que le permite escapar de óptimos locales.

Metáforas

Matemáticas + Inspiración

Las metaheurísticas muchas veces combinan técnicas formales con estrategias informales observadas en procesos naturales (incluye humanos y sus artefactos).

Parten de la aceptación de que el espacio de búsqueda es computacionalmente intratable. Es decir, de que no conocemos si existen algoritmos eficientes para resolverlos de manera exacta. Por lo tanto, se vale utilizar métodos informales, estratégias algoritmicas que subjetivamente intuímos pueden funcionar para abordar el problema a resolver.

Los algoritmos admiten *hibridación* de técnicas.

Metáforas

Matemáticas + Inspiración

Las metaheurísticas muchas veces combinan técnicas formales con estrategias informales observadas en procesos naturales (incluye humanos y sus artefactos).

Parten de la aceptación de que el espacio de búsqueda es computacionalmente intratable. Es decir, de que no conocemos si existen algoritmos eficientes para resolverlos de manera exacta. Por lo tanto, se vale utilizar métodos informales, estratégias algoritmicas que subjetivamente intuímos pueden funcionar para abordar el problema a resolver.

Los algoritmos admiten hibridación de técnicas.

En la práctica, la mayoría de los problemas interesantes a resolver en inteligencia artificial no admiten algoritmos exactos de solución. Muchos entran dentro de la categoría de los problemas NP-completos, lo que significa que no pueden solucionarse en un tiempo razonable.

Los algoritmos metaheurísticos tienen un espíritu pragmático y generalmente combinan resultados teóricos formales con estrategias de solución informales.

Frecuentemente la fuente de inspiración para diseñar dichas estrategias es la naturaleza. Como es el caso del algortimo SA que hemos visto hoy.

Una estrategia frecuentemente utilizada en la práctica es la hibridación o combinación de algoritmos. Por ejemplo, los algoritmos denominados *meméticos* combinan la evolución de una población de soluciones con aprendizaje individual.

¿Cuándo usar algoritmos metaheurísticos?

La última línea de defensa

En palabras del investigador Carlos Coello Coello del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, "los algoritmos metaheurísticos constituyen la última línea de defensa en optimización".

¿Cuándo usar algoritmos metaheurísticos?

La última línea de defensa
En palabras del investigador Carlos Coello Coello del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, "los algoritmos metaheurísticos constituyen la última línea de defensa en optimización".

Los metaheurísticos, sin embargo, pueden consumir muchos recursos computacionales y no hay garantía de que funcionarán en un problema dado.

Son algoritmos incompletos, en el sentido de que no siempre darán una respuesta.

En palabras del investigador Carlos Coello Coello, "los algoritmos metaheurísticos constituyen la última línea de defensa en optimización".