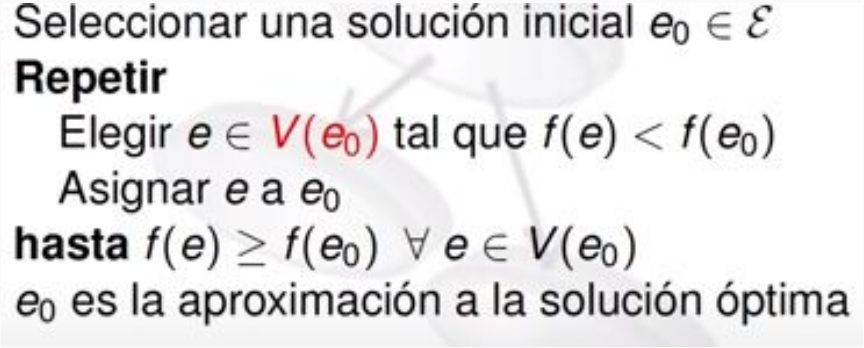
Algoritmos de Búsqueda Local y Problemas de Optimización

Jhon Buesaquillo
Juan Pablo Hernández
Ever Ortega

Algoritmos de Búsqueda Local

Búsqueda Local

- Es la base de muchos de los métodos usados en problemas de optimización.
- Se puede ver como un proceso iterativo que empieza en una solución y la mejora realizando modificaciones locales.
- Básicamente empieza con una solución inicial y busca en su vecindad por una mejor solución. Si la encuentra, reemplaza su solución actual por la nueva y continua con el proceso, hasta que no se pueda mejorar la solución actual.
- La vecindad son todas las posibilidades de soluciones que se consideran en cada punto.

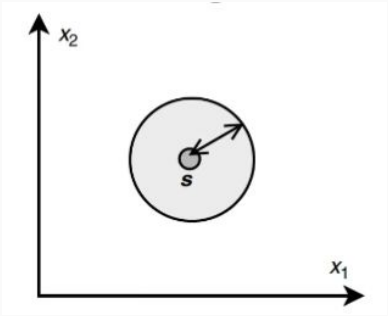


Pseudocódigo para un algoritmo de búsqueda local en un problema de minimización

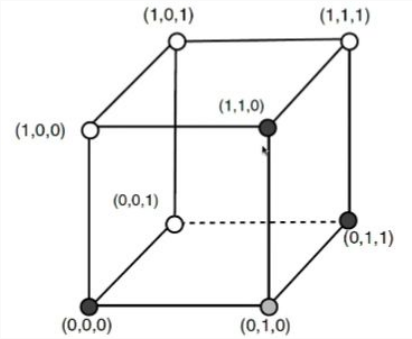
1. Se parte de una solución inicial.
2. Se elige una solución vecina que mejore la función objetivo de la solución actual.
3. Se repite hasta que no exista una solución vecina que mejore la solución actual.
4. La solución final es la aproximación a la solución óptima.

Vecindario

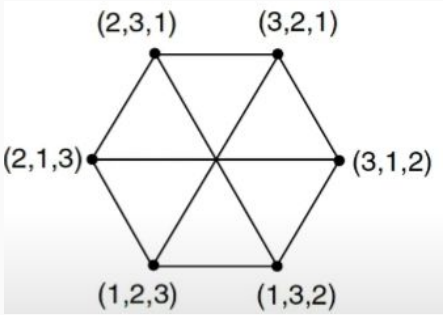
La característica más importante del vecindario es la localidad de las soluciones, es decir que todas ellas tienen atributos similares.



Ej. 1

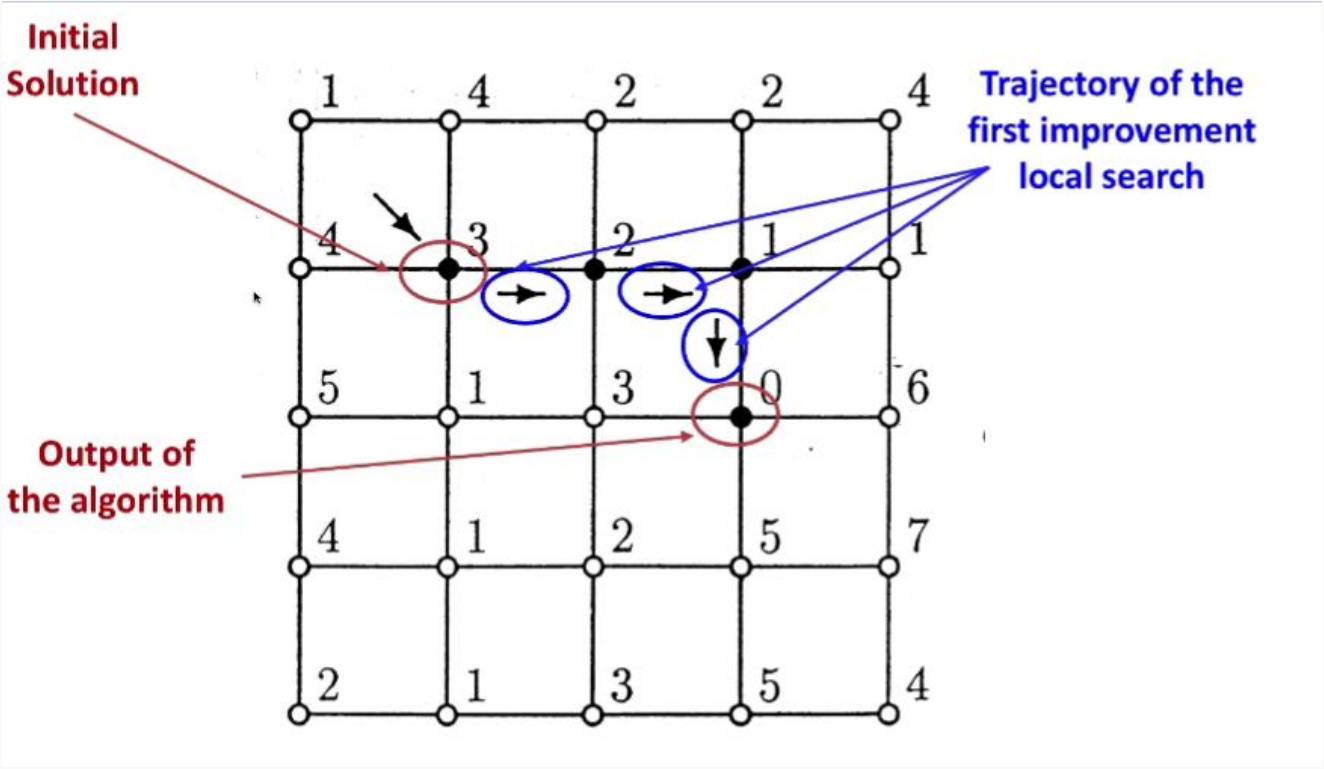


Ej. 2



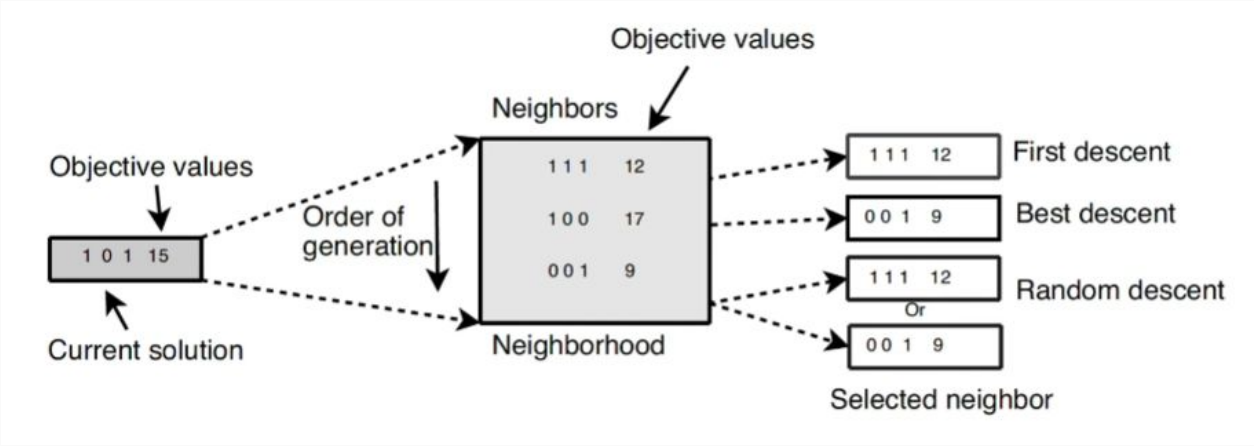
Ej. 3

First Improvement



- Puede encontrar tanto un mínimo local como uno global.
- La respuesta es determinista.
- Si encuentra una solución equivalente se ignora para evitar loops.

Random Selection

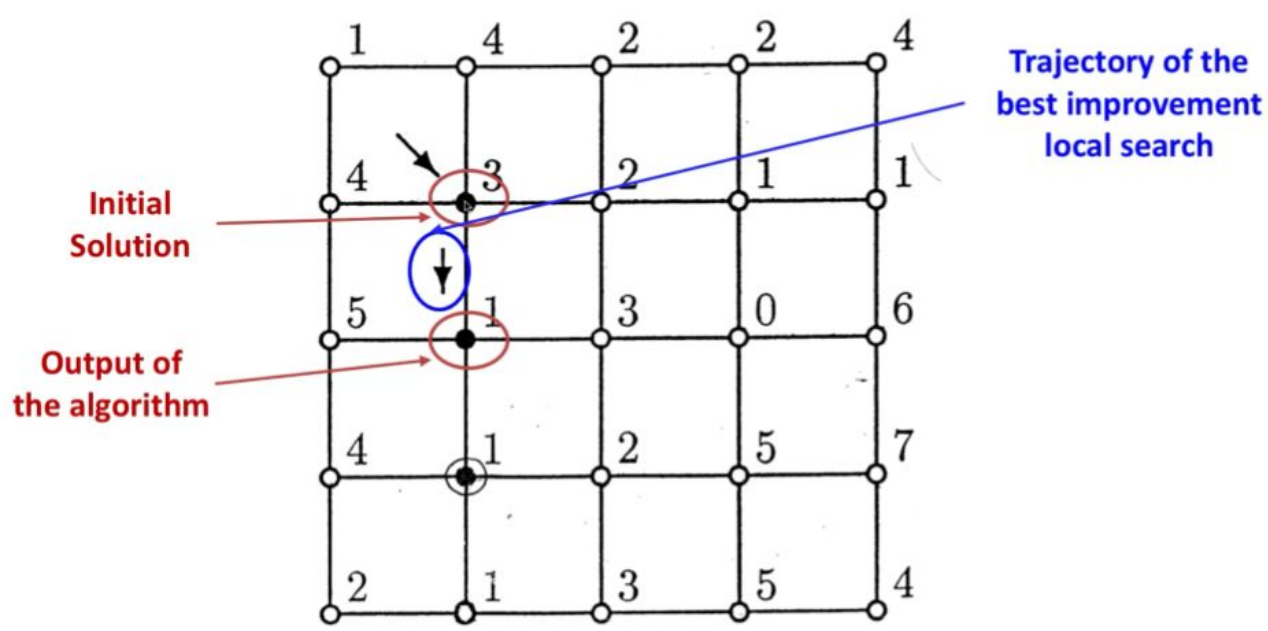


- Puede encontrar tanto un mínimo local como uno global.
- La respuesta es no determinista.
- Si encuentra una solución equivalente se ignora para evitar loops.

Fast Local Search

- Si se consideran muchos vecinos la búsqueda es muy costosa.
- La idea de **Fast Local Search** es ignorar vecinos que probablemente no mejoren la función objetivo.
- La vecindad se divide en sub-vecinos a los cuales se les asigna un bit de activación.
- La idea es revisar sólo los sub-vecinos cuyo bit de activación sea 1.
- Inicialmente todos los bits están activos. Si se prueba que un sub-vecindario no tiene movimientos que produzcan alguna mejora, entonces se desactiva.

Ejemplo



- Puede encontrar tanto un mínimo local como uno global.
- La respuesta es determinista.
- Si encuentra una solución equivalente se ignora para evitar loops.

Random Restart & Multistart Methods

- La forma más simple para mejorar búsqueda local es repitiendo el proceso varias veces con puntos iniciales aleatorios, es decir, **Random Restart**.
- La idea es lograr diversidad y salir de mínimos locales al volver a empezar en otra zona.
- Aunque es fácil de implementar, su efectividad decrece al aumentar la complejidad del problema.
- Para los algoritmos de construcción, se puede hablar de estrategias de re-inicio (**Multistart**). Estas tienen dos fases: (i) generar (construir) una solución y (ii) tratar de mejorar esa solución.

procedimiento MultiStart
while no se satisaface criterio de paro
    1) Construye una solución s (generación)
    2) Busca una mejor solución s'
    Si $s' < s, s \leftarrow s'$
return Mejor Solución

Algoritmo genérico de re-inicio

Guided Local Search

- Es una alternativa a búsqueda local para hacerla más efectiva.
- La idea básica de GLS es aumentar la función objetivo con penalizaciones.
- Utiliza una función de costo aumentado, para permitirle guiar el algoritmo de búsqueda local fuera del mínimo local, a través de funciones de penalización presentes en ese mínimo local.

$$f'(s) = f(s) + \lambda \sum_{i=1}^{m} p_i * I_i(s)$$

Donde:

p_i, son los parámetros de penalidad, que ponderan la importancia de las características: conforme mayor es p_i, mayor es la importancia de la característica i, y por tanto, mayor el costo que tiene la característica en la solución.

$$I_i(s) = \begin{cases} 1 & : i \text{ presente en la solución } s \\ 0 & : i \text{ no está presente en la solución } s \end{cases}$$

Para que búsqueda local pueda salir de óptimos locales, GLS añade penalizaciones a ciertos atributos o características.

La utilidad de penalizar un atributo ***i*** dado un óptimo local ***s**** es:

$$util_i(s^*) = I_i(s^*) \times \frac{c_i}{1 + p_i}$$

Donde c_i es el costo del atributo y p_i es la penalización actual del atributo ***i***. El atributo de mayor utilidad es el penalizado (se incrementa su penalización actual).

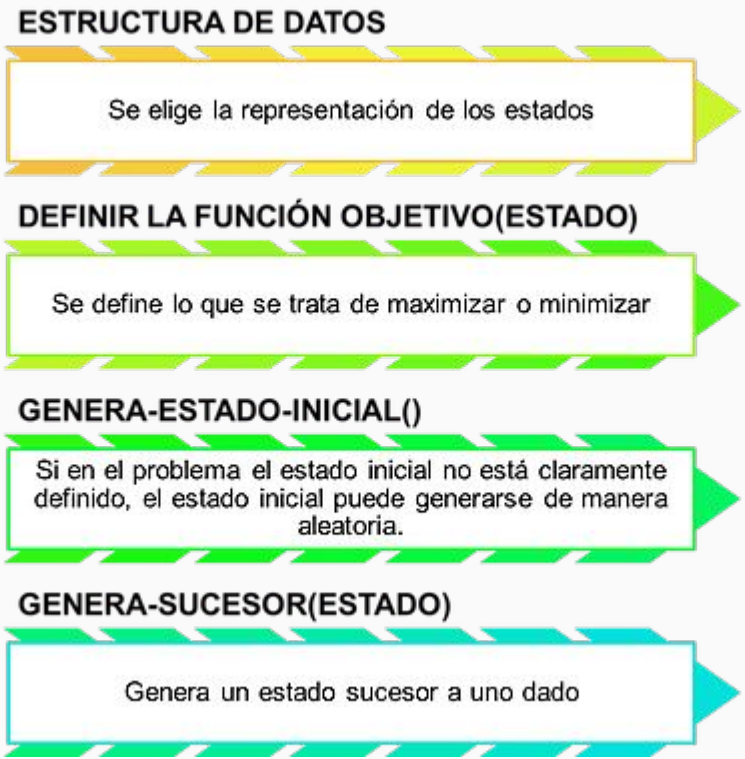
$k \leftarrow 0$
$s_0 \leftarrow$ genera una solición inicial
for $i = 1$ until M **do**
    $p_i = 0$
$h(s) = g(s) + \lambda \times \sum(p_i \times I_i(s))$
while criterio de paro **do**
    $s_{k+1} =$ búsqueda local(s_k, h)
    for $i = 1$ until M **do**
        $util_i(s^*) = I_i(s^*) \times \frac{c_i}{1+p_i}$
    for each i tal que $util_i$ es máxima **do**
        $p_i \leftarrow p_i + 1$
    $k \leftarrow k + 1$
return mejor solución encontrada

Algoritmo de búsqueda guiada

Problemas de Optimización

Representación de un problema de optimización

- Elección de una representación para los estados (estructura de datos)
- Función **F-OBJETIVO(ESTADO)**
  - Función cuyo valor se trata de optimizar
  - Minimizar o maximizar
- Función **GENERA-ESTADO-INICIAL()**
  - Si en el problema el estado inicial no está claramente definido, el estado inicial puede generarse de manera aleatoria, o usando alguna técnica heurística
- Función **GENERA-SUCESOR(ESTADO)**
  - Genera un estado sucesor a uno dado
  - Define la noción de “vecindad” para el problema concreto
  - Usualmente, existe cierta componente aleatoria y heurística en la generación del sucesor

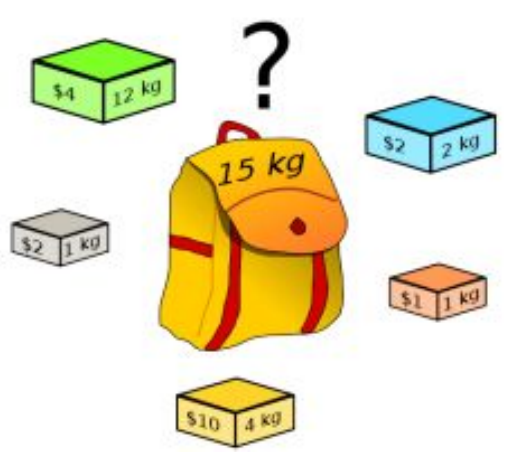


Representación de problemas de optimización

Características de un problema de optimización

- Estado inicial: no está claramente definido
- Operadores:
  - Se puede definir cierta noción de nodo “sucesor” o “vecino”
  - En algunos casos, gran cantidad de vecinos
- Estados finales y soluciones:
  - Todos los estados son posibles soluciones, pero se trata de encontrar una solución “buena” (cuantificada por una función objetivo)
  - Si es posible, la mejor
  - Se busca el estado con un óptimo valor (máximo o mínimo) de función objetivo

Ejemplo



- **Solución:** Cualquier combinación de objetos en la mochila
- **Solución Inicial:** Mochila vacía
- **Operadores:** Meter y sacar objetos de la mochila
- **Función heurística:** $max \sum_i Valor_i$ o $max \sum_i \frac{Valor_i}{Peso_i}$

Referencias

- Morales, E. (2004). Búsqueda Local. Disponible en: https://ccc.inaoep.mx/~emorales/Cursos/Busqueda/node58.html#:~:text=B%C3%BAsqueda%20local%20es%20la%20base,vecindad%20por%20una%20mejor%20soluci%C3%B3n.
- Ruiz, J. (2012). Tema 6: Búsqueda local y algoritmos genéticos. Disponible en: https://www.cs.us.es/cursos/ia1/temas/tema-06.pdf
- Murillo, R. (2015). TEMA 5: BÚSQUEDA LOCAL Y PROBLEMAS DE OPTIMIZACIÓN. Disponible en: https://jraquelm2.wixsite.com/ia2raquelmurillo/single-post/2015/06/10/TEMA-5-B%C3%9ASQUEDA-LOCAL-Y-PROBLEMAS-DE-OPTIMIZACI%C3%93N
- (2012). Búsqueda Local. Disponible en: https://www.cs.upc.edu/~bejar/ia/transpas/teoria/2-BH3-Busqueda_local.pdf