METAHEURISTICAS

Ideas, Mitos, Soluciones

Irene Loiseau

2do Cuatrimestre 2017

Ejemplos de problemas de optimización combinatoria:

- *Problema de la suma de subconjuntos*
- *Determinación de caminos mínimos en grafos*
- *Flujo en redes*
- *Asignación de tareas*
- *Problema de la mochila*
- *Problemas de ruteo de vehículos. El problema del Viajante de comercio*
- *Diseño de redes de comunicaciones*
- *Ruteo en redes de comunicaciones*
- *VLSI*

- *Planificación de tareas*
- *Asignación de recursos y horarios en instituciones educativas*
- *Minimizaron de desperdicios en el corte de materiales*
- *Localización de plantas*
- *Planificación financiera*
- *Problemas de energía*
- *Biología Computacional (secuenciamiento de ADN, árboles filogenéticos, doblamiento de proteínas)*
- *etc.*

Cómo se modela matemáticamente un problema de optimización combinatoria?

Minimizar (o maximizar) f(x)

sujeto a $g(x_i) \geq b_i \quad i=1.....m_1$

$h(x_i) = c_i \quad i= m_{1\ +1},..... M$

$x_i \ \varepsilon \ Z$

- función objetivo
- variables de decisión
- restricciones

(No siempre se puede modelar exactamente así un problema de optimización combinatoria)

Cómo se modela como problema de programación lineal entera un problema de optimización combinatoria?

- Problema de la mochila
- Problema de asignación
- Problema de flujo máximo
- Problema del viajante de comercio.

Cómo se resuelve un problema de optimización combinatoria?

- *Enumeración completa o "algoritmo de fuerza bruta". Sirve?*

COMPLEJIDAD COMPUTACIONAL

Qué hacer?

- SOLUCIONES EXACTAS
- HEURISTICAS

HEURISTICAS

- Heurísticas clásicas
- Metaheurísticas o heurísticas “modernas” o sistemas inteligentes

Cuándo usarlas?

- Problemas para los cuales no se conocen “buenos” algoritmos exactos
- Problemas difíciles de modelar

Porqué usarlas?

- Adaptabilidad a modificaciones de los datos o del problema una vez que ya se obtuvo un resultado.
- Fáciles de implementar y programar
- Basadas en tener una gran capacidad de cálculo
- No sólo para problemas de optimización combinatoria

Cómo se evalúan?

- problemas test
- problemas reales
- problemas generados al azar
- cotas inferiores

Problema del viajante de comercio (TSP)

Dado un grafo G con longitudes asignadas a los ejes queremos determinar un circuito hamiltoniano de longitud mínima.

No se conocen algoritmos polinomiales para resolver el problema del viajante de comercio.

Tampoco se conocen algoritmos ε- aproximados polinomiales para el TSP general (si se conocen cuando las distancias son euclideanas)

Es "el" problema de optimización combinatoria más estudiado.

Algunas heurísticas y algoritmos aproximados para el TSP

Heurística del vecino más próximo

Empezar

Elegir un nodo v cualquiera de G

Poner l(v) = 0

Inicializar i = 0

Mientras haya nodos sin marcar hacer:

poner i = i +1

elegir el eje (v,w) "más barato" tal que w no esté marcado.

poner l(w) = i

poner v = w

Fin

Cuál es la complejidad de este algoritmo?.

Heurísticas de inserción

Empezar

- *Construir un circuito de longitud 3.*
- *Marcar los nodos del circuito*
- *Mientras haya nodos sin marcar hacer*

    ELEGIR un nodo v sin marcar

    INSERTAR v en el circuito

Fin

Cómo ELEGIR?. Cómo INSERTAR?.... variantes de la heurística de inserción.

Para INSERTAR el nodo v elegido

si $c_{i\,j}$ es el costo o la longitud de un eje (i,j), elegimos dos nodos i, i+1 que ya están en el circuito y son consecutivos en el mismo y tal que

$$c_{i\,v} + c_{v\,i+1} - c_{i\,i+1}$$

sea mínimo. Insertamos v entre i e i+1.

Podemos ELEGIR el nuevo nodo v para agregar al circuito tal que:

- *v sea el nodo más próximo a un nodo que ya está en el circuito.*
- *v sea el nodo más lejano a un nodo que ya está en el circuito.*
- *v sea el nodo "más barato", o sea el que hace crecer menos la longitud del circuito.*
- *v se elige al azar.*

En los dos primeros casos y en el cuarto la complejidad del algoritmo es $O(n^2)$, en el tercero es $O(n^2 \log n)$

En el caso de grafos euclideanos (por ejemplo grafos en el plano R^2), se puede implementar un algoritmo de inserción

- *usando la cápsula convexa de los nodos como circuito inicial*
- *insertando en cada paso un nodo v tal que el ángulo formado por los ejes (w,v) y (v,z), con w y z en el circuito ya construido, sea máximo.*

Hay muchas variantes sobre estas ideas.

Heurística del árbol generador

- *Encontrar un árbol generador mínimo T de G*
- *Duplicar los ejes de T*
- *Armar un circuito euleriano E con los ejes de T y sus "duplicados"*
- *Recorrer E usando DFS y armar un circuito hamiltoniano G*

Cuál es la complejidad de este algoritmo?

Teorema: Si las distancias del grafo cumplen la desigualdad triangular la heurística del árbol generador tiene una perfomance en el peor caso dada por

$$x^H(i)/\, x^*(i) \leq 2$$

O sea si las distancias son euclideanas hay algoritmos polinomiales para resolver el problema del TSP aproximado.

Perfomance de las otras heurísticas en el peor caso:

Si las distancias en G son euclideanas se puede probar que valen las siguientes cotas para la perfomance en el peor caso:

- Vecino más próximo $x^h(i)/x^*(i) \leq \frac{1}{2}(\lceil \log n \rceil + 1)$
- Inserción del más próximo $x^h(i)/x^*(i) \leq 2$
- Inserción del más lejano $x^h(i)/x^*(i) \leq 2 \log n + 0.16$
- Inserción del más barato $x^h(i)/x^*(i) \leq 2$

Heurísticas de mejoramiento

Cómo podemos mejorar la solución obtenida por alguna heurística constructiva como las anteriores?

Heurística 2-opt de Lin y Kernighan

- *Obtener una solución inicial H (o sea un circuito hamiltoniano H), por ejemplo con alguna de las heurísticas anteriores.*
- *Mientras sea posible hacer:*

  . Elegir dos ejes de G tal que al sacarlos de H y reemplazarlos por los dos necesarios para reconstruir un nuevo circuito hamiltoniano H´ obtengamos un H´ de longitud menor a la de H.

  . H = H´

. Fin

Cuándo para este algoritmo?. Se obtiene la solución óptima del TSP de este modo?

Algoritmo de búsqueda local

- En vez de elegir para sacar de H un par de ejes que nos lleve a obtener un circuito de menor longitud podemos elegir el par que nos hace obtener el menor H´ entre todos los pares posibles*.(más trabajo computacional)*

- Esta idea se extiende en las heurísticas k-opt donde se hacen intercambios para cualquier k. O sea en vez de sacar dos ejes, sacamos k ejes de H y vemos cual es la mejor forma de reconstruir el circuito. En la práctica se usa sólo 2-opt o 3 –opt.

ESQUEMA GENERAL DE UN ALGORITMO DE DESCENSO (O BUSQUEDA LOCAL)

S= conjunto de soluciones

N(s) =soluciones "vecinas" de la solución s

Elegir una solución inicial $s_0 \in S$

Repetir

Elegir $s \in N(s_0)$ *tal que* $f(s) < f(s_0)$

Reemplazar s_0 *por s*

Hasta que $f(s) > f(s_0)$ *para todos los* $s \in N(s_0)$

Cómo determinar las soluciones vecinas de una solución s dada?

Qué se obtiene con este procedimiento? Sirve?

Optimos locales y globales

Espacio de búsqueda

Se podría también modificar el algoritmo de búsqueda local poniendo

"*Elegir* $s \in N(s_0)$ *tal que* $f(s) = \min_{s \in N(s0)} \{ f(s_0)\}$

Si $f(s) < f(s_0)$ *Reemplazar* s_0 *por* s "

Qué ventajas o desventajas tiene esta nueva forma del algoritmo de búsqueda local?

Problema de asignación de tareas:

Supongamos que tenemos el problema de asignar tareas a un sola máquina de modo a minimizar el tiempo total de ejecución.

Cada trabajo j tiene un tiempo de procesamiento p_j y una fecha de entrega d_j. El objetivo es entonces minimizar

$$T = \sum_j \max \{(C_j - d_j), 0\}$$

donde C_j es el momento en que se completa el trabajo j.

Como elegir las soluciones iniciales?. A priori se puede tomar cualquier permutación de las tareas.

Determinación de los vecinos de una solución dada: en este caso podemos tomar los que se obtengan de la solución actual cambiando la posición de un trabajo con otro.

En un problema con 4 trabajos por ejemplo los vecinos de (1,2,3,4) serán:

$$N(s) = \{(1,3,2,4),(3,2,1,4),(1,2,4,3),$$
$$(1,4,3,2),(2,1,3,4),(4,2,3,1)\}$$

La elección de vecinos y de la función objetivo:
En muchos casos no es inmediata.

Ejemplo: Coloreo de grafos

(Ejemplo del libro de Reeves, pág 36)

- Grafo G = (V, X)

Se puede plantear el problema como particionar el conjunto de vértices V en K conjuntos.

Una solución sería entonces

$$s = (V_1, V_2 \ldots\ldots\ldots, V_k)$$

El objetivo es entonces minimizar k.

No es una buena función objetivo……..

- *Qué pasa si se intentan usar intercambios para pasar de una solución factible a otra?.*
- Componentes bicolores. Complicado

Ideas:

- Cambiar la definición de solución factible. Poner como solución factible una partición s cualquiera de V, aunque no defina un coloreo factible.

- Los vecinos se pueden obtener a partir de s cambiando un nodo de un conjunto V_i a otro.

- Agregar a la función objetivo una función de penalidad que cuente cuantos ejes hay dentro de cada subconjunto o sea

$$f(s) = k + \Sigma \mid E_i \mid$$

donde Ei es el conjunto de ejes entre nodos del conjunto Vi

- La idea es que minimizar esta función objetivo fuerza a moverse a solución factibles.

Tampoco es una buena función objetivo……..

- Otra propuesta de función objetivo:

$$f(s) = -\Sigma \mid C_i \mid^2 + \Sigma\, 2 \mid C_i \mid \mid E_i \mid$$

donde $\mid C_i \mid$ es el nro. de nodos y $\mid E_i \mid$ el nro. de ejes del subconjunto V_i

- *Ninguna funciona demasiado bien en problemas grandes.*

- A veces necesitamos sólo una solución factible del problema de coloreo y se usa sólo una función de penalidad.

- *El problema de coloreo es muy difícil de tratar tanto en forma exacta com heurística*

METAHEURISTICAS

NO HAY UNA DEFINICION UNICA PARA ESTE TERMINO

- *"Una metaheurística es un conjunto de conceptos que pueden ser usados para definir algoritmos heurísticos para un amplio espectro de problemas diferentes".*

- *"Las metaheurísticas son estrategias de alto nivel que guían una heurística específica del problema a resolver para mejorar su perfomance"*

Principales características de las metaheurísticas:

- Las metaheuristicas son estrategias que guían un proceso de búsqueda.
- Las metaheurísticas no son técnicas para un problema específico. Sus conceptos básicos se pueden describir con un alto nivel de abstración.
- El objetivo es explorar eficientemente el espacio de búsqueda para encontrar soluciones óptimas o casi óptimas.
- Las técnicas metaheurísticas van desde algoritmos simples de búsqueda local a complejos procesos de aprendizaje.
- Las metaheurísticas son en muchos casos algoritmos no-determinísticos.
- Las metaheurísticas pueden usar conocimiento del dominio específico de aplicación manejando heurísticas controladas por ellas.
- Algunas metaheurísticas hace uso de la “memoria” de la búsqueda para guiar los pasos futuros.

TECNICAS METAHEURISTICAS

- Simulated annealing (primeros trabajos 1953, 1983)
- Tabú Search (primeras aplicaciones a optimización combinatoria en 1986, basado en algunas ideas de los 70)
- Algoritmos genéticos y evolutivos (primeras ideas en los 60, mayormente aplicaciones a problemas de IA).
- BRKGA (biased random keys genetic algorithms).
- GRASP (1989)
- Colonia de hormigas (1992),
- VNS (Variable Neighborhood Search)
- Iterated Local Search
- Scatter Search and Path Relinking
- Honey-bee mating optimization
- Particle Swarm Optimization
- Sistemas Inmunes Artificiales
- Redes neuronales (primeras ideas en los 60, resurgieron en los 80)
- **Etc.**
- Híbridos

SIMULATING ANNEALING

Ideas básicas:

Metropolis, A., Rosenbluth,M., Rosenbluth,A., Teller, A., Teller,E., " Equation of state calculation by fast computing machines", J. of Chemistry Physics, 1953

Algoritmo para simular el proceso de enfriamiento de un material (cristal) en un baño de calor (annealing).

Si un material sólido se calienta por encima de su punto de fundición y después se enfría hasta quedar en estado sólido, las propiedades de este sólido dependen de la velocidad de enfriamiento.

El proceso de "annealing" se puede simular modelando el material como un sistema de partículas. El algoritmo de Metropolis simula matemáticamente los cambios de energía del sistema cuando se lo somete a un proceso de enfriamiento hasta que converge a un estado de "congelamiento"

30 años más tarde:

Kirkpatrick,S., Gellat,C., Vecchi,M., "Optimization by simulating annealing", Science, 1983.

Tomando como base el trabajo de Metropolis et all, proponen usar estas ideas para buscar soluciones óptimas de problemas de optimización. Esto implica mejorar el algoritmo básico de (descenso o búsqueda local), permitiendo movimientos donde el valor de la función empeora con una frecuencia gobernada por una función de probabilidad que cambia a lo largo del algoritmo.

Las leyes de la termodinámica dicen que a temperatura t la probabilidad de un crecimiento de la energía de magnitud δE está dada por

$$p(\delta E) = \exp(-\delta E / kt) \qquad (1)$$

donde k es la constante de Boltzman.

Metropolis et all. generan una perturbación y calculan el cambio de energía. Si la energía decrece el sistema se mueve al nuevo estado. Si crece el nuevo estado es aceptado con probabilidad (1). El procedimiento se repite para un número predeterminado de iteraciones para cada temperatura, después de lo cual la temperatura se disminuye hasta que el sistema se congela en estado sólido.

Kirkpatrick et al., y Cerny (1985) propusieron en forma independiente usar este algoritmo en problemas de optimización combinatoria haciendo un paralelo entre el proceso de enfriamiento y los elementos de este tipo de problemas

Si hacemos la siguiente correspondencia:

estado del sistema	**Solución factible**
Energía	**Costo**
cambio de estado	**Solución vecina**
Temperatura	**Parámetro de control**
estado de congelamiento	**Solución heurística**

Estas correspondencias dan una idea de como cualquier algoritmo de búsqueda local puede convertirse en un algoritmo “annealing” tomando al azar un cierto numero de vecinos y permitiendo que elija una solución peor de acuerdo a la ecuación (1)

- *Cómo es la función de energía para la mayor parte de los materiales?.*

- *Cómo es la función objetivo en problemas de optimización?.*

ESQUEMA GENERAL DE UN ALGORITMO SIMULATING ANNEALING

Elegir una solución inicial s_0
Elegir una temperatura inicial $t_0 > 0$
Elegir una función de reducción de temperatura $\alpha(t)$

Repetir ***mientras no se verifique la condición de parada***
Repetir ***hasta icount = nrep (t)***
Elegir al azar $s \in N(s_0)$
$\delta = f(s) - f(s_0)$
Si $\delta < 0$
entonces $s_0 := s$
Sino
Generar x al azar en (0,1)
Si x < exp $(-\delta / t)$ *entonces* $s_0 := s$
Fin
Poner $t = \alpha(t)$
Fin

Qué hay que hacer para usar este esquema en la práctica?

- Definir conjunto de soluciones
- Definir función de costo
- Definir vecinos
- “Elegir” parámetros: **temperatura, nrep, α**
- “Elegir” criterio de parada

Más Detalles

La temperatura inicial debe ser suficientemente “alta” como para permitir un intercambio casi libre de las soluciones vecinas.

La forma de reducir la temperatura es vital para el éxito de la heurística. Hay que determinar:

- *número de repeticiones en cada temperatura.*
- *forma de reducción de la temperatura*

Muchas iteraciones por temperatura para pocos valores de temperatura o al revés.

Por ejemplo: $\alpha(t) =$ a t con a < 1

Se acostumbra usar 0.8 < a < 0.99

- *nrep* puede crecer geométricamente o aritméticamente en cada temperatura.
- En algunos casos se usa la historia para determinar *nrep*
- **Otra propuesta:** $\alpha(t) = t/(1 + b\ t)$ con b chico

En la práctica.....

Criterios de parada:

- Resultados teóricos. Se han demostrado teoremas de convergencia que garantizan que bajo ciertas hipótesis SA converge a la solución óptima.

- *Estos resultados permiten esperar un buen comportamiento del algoritmo pero no sirven en la práctica porque la convergencia se garantiza en un nro exponencial de iteraciones respecto del tamaño del problema.*

- en la práctica.....se puede usar como criterio de parada el número de iteraciones sucesivas sin mejora

Elección del espacio de soluciones y definición de vecinos: *muy importante*.

Factibilidad
Penalidades en la función objetivo
Evitar funciones “chatas”

Problemas de horarios en instituciones educativas

- Cada institución tiene sus propias reglas.
- Muchas restricciones y objetivos variados.

Consideremos acá el problema de programar un conjunto dado de exámenes en un numero fijo de franjas horarias, de modo que ningún estudiante tenga que dar dos exámenes a la vez, y que haya aulas del tamaño necesario para los mismos.

Puede haber muchas otras restricciones.

Presentaremos un caso tratado por K. Dowsland
(libro de Rayward-Smith, Osman, Reeves, Smith, 1996).

Se quieren programar los exámenes de un periodo en la Swansea University

Restricciones "fuertes":

- 600 exámenes
- 3000 alumnos, 24 franjas horarias disponibles
- ningún alumno puede dar dos exámenes al mismo tiempo.
- algunos pares de exámenes pueden ser programados en el mismo horario.
- algunos pares de exámenes NO pueden ser programados en el mismo horario.
- algunos grupos de exámenes tienen que ser programados en un cierto orden.
- algunos exámenes tienen que ser programados dentro de determinadas ventanas de tiempo.
- no puede haber mas de 1200 alumnos dando examen al mismo tiempo.

Objetivos secundarios:

- minimizar el número de exámenes de mas de 100 alumnos después del periodo 10.
- minimizar la cantidad de alumnos que tienen que dar dos exámenes en periodos consecutivos.

No es un problema de optimización. Cuál es el objetivo?
Difícil encontrar una solución factible.

Primer paso: decidir cuales serán las soluciones factibles y cuales restricciones se incorporaran a la función objetivo como penalidades.

El problema básico de horarios se puede modelar como un problema de coloreo de grafos. La solución propuesta se basa en algoritmos Simulating Annealing para ese problema.

Espacio de soluciones: asignación de exámenes a las 24 franjas que verifiquen las restricciones de orden.

Vecinos: inicialmente, cambiar una franja horaria para un examen. En una segunda etapa se consideró otra definición también.

Costo:

$w_1 c_1 + w_2 c_2 + w_3 c_3 + w_4 c_4 + w_5 c_5 + w_6 c_6$

donde

- c_1 *es el número de conflictos de los estudiantes*
- c_2 *es el número de exámenes que colisionan por otras razones*
- c_3 *es el número de violaciones de ventanas de tiempo*
- c_4 *es el número de violaciones de la capacidad de las aulas*
- c_5 *es el número de exámenes que no verifican la segunda restricción secundaria*
- c_6 *es el número de exámenes de mas de 100 alumnos fuera del periodo preferido.*
- $w_1, w_2, w_3, w_4, w_5, w_6$ *pesos*

RESULTADOS:

Mucho experimentos con enfriamiento muy lento.

Pesos altos para las restricciones mas importantes, permitieron que cuando quedaban satisfechas no volvieran a ser violadas.

Basada en las observaciones de estos primeros experimentos, se intento un procedimiento en 2 etapas.

La función objetivo para la primera fase fue:

$f_1 = w_1 c_1 + w_2 c_2 + w_3 (c_3 + c_6) + w_{4\ c4}$

Este primer problema se resolvió con relativa facilidad poniendo todos los pesos en 1y usando una tasa de enfriamiento de 0.99 cada 1000 iteraciones. Se obtuvieron entonces soluciones con costo 0.

La fase 2 partió de esta solución final y trabajo en un espacio de soluciones en las cuales todas las soluciones con f1 > 0 habían sido eliminadas.

La función objetivo fue

$$f_2 = c_5$$

Fue necesario usar un esquema de enfriamiento muy lento para obtener buenos resultados. Se uso una temperatura inicial de 20, reduciéndola a 0.99 t cada 5000 iteraciones.

El manejo de vecinos en ambas fases fue diferente.

- En la fase 1 se uso un intercambio simple: intercambiar un examen de una franja horaria a otra.
- Partiendo de soluciones con valor $f_1 = 0$, en la fase 2 se experimentó también con intercambios basada en las componentes bicolores (intercambio de los colores de los nodos que están en la misma componente bicolor). De esta forma si las soluciones eran factibles para la fase 1 siguen siéndolo.

Resultados: cantidad promedio de conflictos c_5 en la solución final. Corridas con igual nro de iteraciones total.

	Promedio de varias corridas
Una fase, intercambio simple	372,3
Dos fases, intercambio simple	330
Dos fases, intercambio componentes bicolor	262,3

Problema de Ruteo de Vehículos

- El problema básico de Ruteo de Vehículos (Vehicle Routing Problem, VRP) requiere que se planifiquen de forma óptima las rutas de un conjunto de vehículos que tienen que visitar a un conjunto de clientes partiendo de un depósito.

- Hay muchas variantes del VRP, que puede incluir restricciones en la capacidad de los vehículos, en la longitud de los recorridos, tener horarios de entrega o recogida, etc.

- Problemas de optimización combinatoria de gran importancia económica y muy estudiados.

- El ejemplo que vamos a presentar fue propuesto en:

Chiang, W,Russell, R.,"*Simulating annealing metaheuritics for the vehicle routing problem with time windows*", Annals of Operations Research 63 (1996)

Para cada cliente i definimos:

- *qi , carga o demanda del cliente i*
- *si, tiempo de servicio del cliente i*
- *ei, hora más temprana en que puede iniciarse el servicio en i*
- *li, última hora en que puede iniciarse el servicio en i*
- *tij, tiempo de viaje entre i y j*
- *dij, distancia entre i y j*
- *Qv, capacidad del vehículo v*
- *bi, momento en que se empieza el servicio en i*
- *bij, momento en que puede empezar a servirse al cliente j suponiendo que el cliente j va después del i en una ruta.*
- $bij = max \{ ej, bi+si+tij \}$
- *wj, tiempo de espera si el vehículo llega a j antes de ej.*
- $wj = ej - (bi+si+tij)$

- En este trabajo se construye una solución inicial usando un procedimiento de construcción de rutas en paralelo basado en una heurística inserción propuesta por Solomon.

- Se estima en primer lugar la cantidad de vehículos necesarios V y se genera esa cantidad de rutas. Al final se agregan rutas si quedaron clientes sin servir, o se trata de disminuir el nro de rutas si la cantidad inicial V alcanzó.

Se usan 3 reglas para elegir que cliente insertar:

- elegir el cliente con el menor e_j,

- elegir un cliente tomando en cuenta el tamaño de la ventana de tiempo. Se considera el valor 100 $(lj - ej) - d_{0j}$ donde d_{0j} es la distancia al depósito.
-
- Elegir el cliente con el mayor d_{0j}

- El lugar para insertar al cliente está determinado por una combinación de la distancia a la ruta y por el aumento del tiempo de viaje que produce insertar ese cliente.

Sean

$$c_1(i,j,k) = d_{ij} + d_{jk} - d_{ik}$$

$$c_2(i,j,k) = b_{jk} - b_k$$

$$c_3(r) = \alpha_1 c_1(i,j,k) + \alpha_2 c_2(i,j,k) \quad \text{con } \alpha_1 + \alpha_2 = 1$$

Se usa el valor de $c_3(r)$ para calcular el mejor lugar para insertar el cliente j. O sea se inserta en cliente j en la ruta r para la cual

$$r = \arg\min \{c_3(v) / v = 1 \ldots V\}$$

La solución de la heurística de inserción es la mejor de las seis combinaciones de inserción y selección de ruta (consideraron los valores con $\alpha_1 = 1$ o con $\alpha_2 = 1$).

A partir de la solución inicial anterior los autores aplican un algoritmo Simulating Annealing:

- **Soluciones factibles**: conjunto de rutas que verifican las ventanas de tiempo {R1,…….Rv} donde Rp es el conjunto de clientes visitados por el vehículo p

- **Vecinos**: se obtienen cambiando un cliente de una ruta a otra o intercambiando 2 clientes. Se consideraron 2 vecindarios diferentes:

N1(S): construido a partir de elegir los dos vecinos siguientes del vecino i en la ruta p y los dos vecinos que no están en la ruta p que tienen menor costo de inserción en dicha ruta. Se evalúa la posibilidad de insertar cada uno de estos cuatro elementos para cada cliente i, se repite el procedimiento para todos los i y al final se tiene una nueva solución S que es aceptada o no por el SA.

A partir de la solución inicial anterior los autores aplican un algoritmo Simulating Annealing:

- **Soluciones factibles**: conjunto de rutas que verifican las ventanas de tiempo {R1,…….Rv} donde Rp es el conjunto de clientes visitados por el vehículo p

- **Vecinos**: se obtienen cambiando un cliente de una ruta a otra o intercambiando 2 clientes. Se consideraron 2 vecindarios diferentes:

N1(S): construido a partir de elegir los dos vecinos siguientes del vecino i en la ruta p y los dos vecinos que no están en la ruta p que tienen menor costo de inserción en dicha ruta. Se evalúa la posibilidad de insertar cada uno de estos cuatro elementos para cada cliente i, se repite el procedimiento para todos los i y al final se tiene una nueva solución S que es aceptada o no por el SA.

N2(S): se analizan todos los pares de rutas (R_s ,R_t) en un cierto orden. Hay 3 tipos de operadores. Cambiar un cliente de R_s a R_t o cambiar uno de R_t a R_s o intercambiar dos clientes entre las rutas. Se busca la primera posición en las rutas en la cual se produce una mejora (si existe).

Función de Costo a optimizar

$$C(S) = b1\ R + b2\ T + b3\ D$$

con pesos $b1 >> b2 > b3$ y donde

R es el total de vehículos usados

D la distancia total recorrida

T el tiempo total de recorrer todas las rutas

Parámetros y criterios de parada

Probabilidad inicial $P_0 = 0.05$

Temperatura inicial T_0 : se elige de forma que un movimiento peor sea aceptado con probabilidad P_o.(se generan al azar movimientos para determinar esto)

Cambio de la temperatura $\alpha(t)=0.95\ t$

Nrep proporcional al tamaño del vecindario (0.95%)

Criterio de parada: si los valores de la función objetivo no mejoran después de un número determinado de iteraciones o si el nro de movimientos aceptados está por debajo de un parámetro predeterminado.

Resultados computacionales

- Se testeó el algoritmo en las instancias de Solomon (100 clientes) y en 2 problemas reales grandes.

- Se compararon los resultados y tiempos de ejecución con los de otras heurísticas.

- En 7 de los ejemplos se pudo comparar con el óptimo conocido. El error obtenido promedio fue de 1.17% en el caso de la cantidad de vehículos y de 7.3% en la distancia recorrida.

Problema de Job Shop Scheduling

Sadeh,N.,Nakakuki, Y., ***Focused simulated annealing search: an application to job shop scheduling, Annals of Operations Research 63, 1996.***

Objetivo: minimizar la suma de los atrasos y costos de inventario.

- Una fábrica en la cual un conjunto finito $J = \{j_1, j_2, \ldots\ldots j_n\}$ trabajos tiene que se realizada en un conjunto de recursos *(máquinas)* $RES = \{R_1, R_2 \ldots\ldots\ldots\ldots R_m\}$.
- Cada R_i puede hacer una sola tarea por vez y no puede interrumpirla.
- Los trabajos se conocen al comienzo de la programación
- Los R_i están disponibles durante todo el período de planificación.
- Para completar cada trabajo j_l se tiene que realizar en un orden preestablecido un conjunto de tareas $O^l = \{O^l_1, O^l_2 \ldots\ldots\ldots O^l_{nl}\}$. Asumimos que O^l_i tiene que completarse antes de que empiece O^l_{i+1}
- Cada trabajo tiene una fecha prevista dd_l para ser completado.
- Cada operación O^l_i tiene una duración du^l_i

- Cada trabajo j_l tiene una fecha erd_l antes de la cual no se puede empezar a hacer.
- Cada trabajo j_l tiene una fecha tope lcd_l de terminación, después de la cual, por ejemplo el cliente ya no lo acepta.
- Se supone que $erd_l \leq dd_l \leq lcd_l$

- st^l_i es el momento en que empieza la operación O^l_i
- Si el trabajo j_l se termina después de dd_l le corresponde un costo $tard_l$ por cada unidad de tiempo que se atrase. El costo por atraso de j_l es entonces $TARD^l = tard_l\ (max\{0, C_l - dd_l\})$ donde $C_l = st^l_{nl} + du^l_{nl}$

- in^l_i es el costo de inventario de la tarea O^l_i o sea
  $INV^l_i = in^l_i\ (max\{C_l, dd_l\} - st^l_i)$

- El costo total de una planificación es entonces:

$$\sum_l TARD^l + \sum_l \sum_i INV^l_i$$

El problema es entonces encontrar una planificación que satisfaga estas restricciones minimizando los costos por atrasos y los costos de inventario.

Simulating annealing para este problema

- Solución inicial: se genera al azar cuidando de que ningún par de operaciones usen el mismo recurso al mismo tiempo.

- Definición de vecindario: se usaron 3 operadores:

RIGHT_SHIFT: se incrementa en 1 el tiempo de comienzo de una operación si eso es factible.

LEFT_SHIFT: de la misma forma se retrasa el tiempo de comienzo de una operación.

EXCHANGE: Se intercambia el orden de dos operaciones consecutivas en un mismo recurso R

Para garantizar que se cumplan las relaciones de precedencia se introduce un costo de penalidad en la función objetivo.

Los resultados se compararon con planificaciones obtenidas mediante heurísticas o reglas empíricas en 8 conjuntos de problemas

Se introdujeron mejoras a este algoritmo SA, dando un mayor peso en la función de costo a las restricciones que están más lejos de cumplirse, tanto respecto de los trabajos como de los recursos.

TABU SEARCH

CONCEPTOS BASICOS:

- Permitir elegir una solución vecina que no sea estrictamente mejor que la actual para “salir” de un mínimo local.
- Usar una lista Tabú de soluciones (o movimientos) para evitar que el algoritmo cicle.
- Usar una función de aspiración que permita en algunos casos elegir un elemento o movimiento Tabú.

ESQUEMA GENERAL DE TABU SEARCH

Inicialización
Elegir una solución inicial s en S
Niter:=0
bestiter:=0
bestsol:= s
T:=∅
Inicializar la función de aspiración A
Mientras *no se cumpla el criterio de parada* ***hacer***

- *niter := niter + 1*
- *generar un conjunto V* de soluciones* $s_v \in N(s)$ *que no sean Tabu o tales que* $f(s_v) \leq A(f(s))$
- *elegir una solución s* que minimice f en V**
- *actualizar la función de aspiración A y la lista Tabú T*
- *s:=s**

si *f(s*) < f(bestsol)* ***entonces***
*bestsol:= s**
bestiter := niter
End

Qué hay que hacer para usar este esquema?:

- Determinar el conjunto de soluciones factibles S.
- Determinar la función objetivo f.
- Dar un procedimiento para generar los elementos de N(s), “vecinos de s”.
- Decidir el tamaño del conjunto $V^* \subset N(s)$ que será considerado en cada iteración
- Definir el tamaño de la lista Tabú T.
- De ser posible definir una cota inferior para la función objetivo f.
- Definir la función de Aspiración A(z) para todos los valores z que puede tomar la función objetivo.
- Definir criterios de parada (nbmax y/o comparación con la cota inferior si la hay)

Volvemos al ejemplo de asignación de tareas que presentamos en "Búsqueda Local":

Como construir el conjunto de soluciones posibles V ?*

En este caso, si, cuando la solución actual es (1,2,3,4) la lista Tabu, proveniente de los pasos anteriores del algoritmo es

T= {(1,3,2,4),(3,1,2,4)(3,2,1,4)}

Entonces V* tiene solo cuatro elementos

(1,2,4,3), (1,4,3,2),(2,1,3,4),(4,2,3,1)}

Posibles reglas Tabu a usar en este caso:

- impedir todos los movimientos donde i ocupa la posición p(i) y j ocupa la posición p(j)
- impedir los movimientos donde alguna de las situaciones arriba suceda
- impedir que el trabajo i vuelva a una posición k con k < p(i)
- impedir que el trabajo i cambie de posición
- impedir que i y j cambien de posición

Como elegir el tiempo de permanencia en la lista Tabu:

- valor fijo (a ser ajustado en la experimentación)
- valor aleatorio entre un tmin y tmax dados a priori.
- valor variable de acuerdo al tamaño de la lista y las variaciones del valor de la función objetivo.

Ejemplos de criterios de aspiración:

- cuando todos los movimientos o vecinos posibles son Tabu, se elige alguno de ellos (“el menos tabu”)
- cuando con un movimiento tabu se obtiene una solución mejor que la mejor hasta ese momento (global o en la región)

Ejercicio:

Pensar cómo implementaría un algoritmo tipo Tabu search para el siguiente problema:

Encontrar el árbol de mínimo peso de k ejes (MWKT) en un grafo G = (V,X) con pesos en los ejes.