71.14 Modelos y Optimización I

Heurísticas y Problemas Combinatorios

Prof. Silvia A. Ramos

"Cierro los ojos y veo una bandada de pájaros. La visión dura un segundo o acaso menos; no sé cuántos pájaros ví. ¿Era definido o indefinido su número? El problema involucra el de la existencia de Dios. Si Dios existe, el número es definido, porque Dios sabe cuantos pájaros vi. Si Dios no existe, el número es indefinido, porque nadie pudo llevar la cuenta. En tal caso, vi menos de diez pájaros (digamos) y más de uno, pero no vi nueve, ocho, siete, seis, cinco, cuatro, tres o dos pájaros. Vi un número entre diez y uno que no es nueve, ocho, siete, seis, cinco, etcétera. Ese número entero es inconcebible, ergo, Dios existe."

Argumentum ornithologicum
Jorge Luis Borges

El número existe, en cualquier problema, ese número es la solución. Pero no siempre podemos hallarlo (no tenemos tanta paciencia) sobre todo si se trata de problemas combinatorios. Aquí aparece el tema que nos ocupa.

Este apunte pretende sólo una introducción al mundo que descubrimos, hace algunos años, cuando empezamos a trabajar en el tema de Heurísticas aplicadas a los modelos de Programación Lineal Entera Combinatoria; y surgió como una guía para el dictado de nuestras clases.

Si, luego de leer el apunte, les gusta el tema, encontrarán en la última página una lista que contiene parte de la bibliografía que pueden consultar para ampliar al respecto.

Quiero agradecer por sus valiosos aportes al Dr. Sebastián Ceria, que nos introdujo en el conocimiento del tema, a Esteban Santellán que colaboró en el desarrollo de las clases teóricas, y a Juan Ramonet, que nos dio su voto de confianza e impulsó su crecimiento, como lo sigue haciendo.

Silvia A. Ramos
Octubre de 1995

Búsqueda Heurística:

Con el fin de resolver problemas complicados con eficiencia, en ocasiones es necesario comprometer algunos requisitos de optimalidad y construir una estructura de control que no garantice encontrar la mejor respuesta pero que casi siempre encuentre una buena solución. De esta forma, surge la idea de **heurística**. La palabra heurística viene de la palabra griega *heuriken* que significa "descubrir", que es también origen de *eureka*, derivado de la famosa exclamación de Arquímedes, *heurika* ("lo encontré").

Así definieron Bartholdi y Platzman (en 1988) lo que es heurística para ellos:

"Una heurística puede verse como un procesador de información que, deliberadamente, peor juiciosamente, ignora cierta información. Ignorando información, una heurística se libra de gran parte del esfuerzo que debió haberse requerido para leer los datos y hacer cálculos con ellos. Por otra parte, la solución producida por tal heurística, es independiente de la información ignorada, y de este modo no se ve afectada por cambios en tal información. Idealmente, se busca ignorar información que resulta muy caro colectar y mantener, esto es, computacionalmente caro de explotar y mantener, y que contribuye en poco a la precisión de la solución."

Podemos definir una heurística como una técnica que aumenta la eficiencia de un proceso de búsqueda, posiblemente sacrificando demandas de completitud. Las heurísticas son como los guías de turismo: resultan adecuados en el sentido de que generalmente suelen indicar las rutas interesantes; son malos en el sentido de que pueden olvidar puntos de interés para ciertas personas. Al usar buenas heurísticas se pueden expresar buenas (aunque posiblemente no óptimas) soluciones a problemas difíciles, como el del viajante de comercio.

Una función heurística es una correspondencia entre las descripciones de estados del problema hacia alguna medida de deseabilidad, normalmente representada por números. Quiere decir que mensura cada estado del problema (solución) y dice qué tan cerca de la solución óptima está.

El propósito de una función heurística es el de guiar el proceso de búsqueda en la dirección más provechosa sugiriendo qué camino tomar cuando hay más de uno disponible. Cuanto más exactamente estime la función heurística los méritos de cada nodo del árbol (o grafo) que representa al problema, más directo será el proceso de solución. En general, hay que hacer una ponderación entre el costo de evaluación de una función heurística y el ahorro de tiempo de búsqueda que proporciona la función.

Para ejemplificar el tema, usaremos una heurística sencilla (Lenat - 1983), que consiste en lo siguiente: Si hay una función interesante de dos argumentos f(x,y) ver qué ocurre si los dos argumentos son idénticos:

Si la función es...	*La heurística lleva al descubrimiento de...*
multiplicación	elevación al cuadrado
unión de conjuntos	identidad
contemplar	introspección
matar	suicidio

En un problema de minimización, el valor de la solución obtenida por la heurística es menor o igual que una constante multiplicada por el verdadero valor óptimo. Esa constante se llama constante de calidad.

Si llamamos:

OPT = Valor óptimo real.

HEUR = Valor encontrado por la heurística como óptimo.

K = Constante de calidad.

Sabemos que $HEUR \leq K \cdot OPT$.

La constante de calidad nos da un límite superior en el error que podemos llegar a tener entre la solución óptima verdadera y la encontrada por la heurística.

Si para una función heurística se puede encontrar un valor de K que permanezca constante para todos los problemas de un mismo tipo, sin influencia del tamaño del problema o de otros factores, diremos que esa heurística tiene garantía de calidad.

Vamos a enfocar el aspecto de heurísticas a aquellos problemas que son de carácter combinatorio y, por lo tanto, tienen un número finito de soluciones, pero ese número es tan grande que lleva un tiempo enorme llegar a la solución óptima. Por ejemplo, la estrella de los problemas combinatorios, que es el problema del viajante, es NP-Hard, lo cual significa que es no determinístico polinomial. Para cualquier problema de esta clase los algoritmos que se conocen para su resolución llevan, para el peor problema, un número exponencial de pasos, por lo general del orden de 2^n. Un problema del viajante de cien ciudades, en el peor de los casos tardaría 2^{100} pasos, lo cual es mucho más de lo razonable.

Sin las heurísticas, estaríamos desesperadamente enredados en una explosión combinatoria. Sólo este argumento sería suficiente para demostrar la necesidad de su uso. Sin embargo, existen otras razones:

- Con frecuencia no se necesita una solución óptima, una buena aproximación es adecuada.
- Si bien las combinaciones que se logran con una heurística pueden no ser muy buenas en los peores casos, estos peores casos raramente ocurren en el mundo real.
- Intentar comprender por qué funciona una heurística, o por qué no lo hace, normalmente sirve para profundizar en la comprensión del problema.

Por otra parte, el método simplex también aplica heurísticas ¿o no lo es acaso la decisión de que la variable que ingrese a la base sea la de z_j-c_j con mayor valor absoluto?. En realidad no garantizamos que así el problema llegue más rápidamente al óptimo. El simplex mismo se podría ver como un método heurístico, lo que ocurre es que su base matemática es tan fuerte que tiene garantía de calidad uno, es decir, siempre llega al óptimo.

Muy frecuentemente se puede encontrar un límite superior en el error cometido. En muchos problemas del mundo real no es posible dar estos límites tan tranquilizadores. Esto es así por dos motivos.

- Es difícil medir con precisión el valor de una solución particular.
- Es adecuado introducir heurísticas basadas en un conocimiento relativamente desestructura-do. Con frecuencia es imposible definir ese conocimiento de forma tal que pueda llevarse a cabo un análisis matemático de su efecto sobre el proceso de búsqueda.

Dijimos que las heurísticas nos podían ayudar a conocer el problema, pues bien, existen dos formas fundamentales de incorporación de conocimiento heurístico específico del dominio a un proceso de búsqueda basado en reglas:

- ♦ En las mismas reglas (si sabemos diferenciar entre movimientos legales que son los que se pueden hacer, y movimientos "sensatos" que son los que nos van a llevar más rápida y seguramente a una solución óptima).
- ♦ Como una función heurística que evalúa los estados individuales del problema y determina su grado de "deseabilidad".

Características del problema.

A fin de elegir el método más apropiado (o una combinación de métodos) para un problema en particular, es necesario analizarlo de acuerdo con varias dimensiones clave:

1) ¿Puede descomponerse el problema?

Hay problemas que pueden descomponerse de tal modo de encontrar subsoluciones y combinarlas para llegar a la solución del problema. Si los subproblemas no son independientes sino que interactúan entre sí, estas interacciones deben ser consideradas a fin de llegar a una solución para la totalidad del problema.

2) ¿Pueden deshacerse o ignorarse pasos hacia una solución?

Según este aspecto hay tres importantes clases de problemas:

- Ignorables: (Por ejemplo, la demostración de teoremas) en los cuales pueden ignorarse pasos dados.
- Recuperables (Por ejemplo: el rompecabezas) en los cuales pueden deshacerse pasos dados.
- No recuperables: (Por ejemplo: el ajedrez), en los cuales no pueden deshacerse pasos dados.

La recuperabilidad de un problema juega un papel importante en la determinación de la complejidad de la estructura de control necesaria para resolver el problema. Los problemas ignorables se resuelven usando una sencilla estructura de control que nunca vuelve hacia atrás. Los problemas recuperables se resuelven con estrategias un poco más complicadas que a veces cometen errores. Para recuperarse de estos errores será necesario una vuelta atrás, de forma que la estructura de control debe implementarse con una pila "push-down" en la que las decisiones se conservan en el caso de que necesiten ser deshechas más tarde. Los problemas no recuperables, se resuelven mediante un sistema que debe aplicar muchísimo esfuerzo en la toma de decisiones ya que éstas son irrevocables. Algunos problemas irrecuperables se resuelven con métodos de estilo recuperable usando un proceso de planificación, en el cual se analiza por adelantado una secuencia de pasos para descubrir a dónde conducirá antes de dar el primer paso.

3) ¿Es predecible el universo?.

Hay problemas de consecuencia cierta y problemas de consecuencia incierta. En los problemas de consecuencia cierta, el resultado de una acción se puede predecir perfectamente. Así la planificación puede utilizarse para generar una secuencia de operaciones que garantizan llegar a una solución. Para los problemas de consecuencia incierta, sin embargo, la planificación puede al menos generar una secuencia de operaciones que tiene una buena probabilidad de conducir a una solución.

Se combina con la característica anterior de modo de que los problemas más difíciles de resolver son los irrecuperables de consecuencia incierta.

4) Una solución adecuada ¿es absoluta o relativa?.

Esta característica clasifica a los problemas de algún camino y problemas de mejor camino. Estos últimos son más difíciles de resolver que los de algún camino. Estos se resuelven frecuentemente en un tiempo razonable mediante heurísticas que sugieran rutas adecuadas para explorar.

5) La solución ¿es un estado o una ruta?.

A veces la solución es un estado y en otras, la solución consiste en especificar los distintos estados (el camino que se ha seguido para llegar al estado final).

La respuesta a esta cuestión nos indica si va a ser necesario almacenar el camino del proceso de resolución del problema.

6) ¿Cuál es el papel del conocimiento?.

Hay problemas en los cuales mucho conocimiento sólo es necesario para acotar la búsqueda y otras en las cuales el conocimiento se emplea para poder reconocer una solución.

7) ¿Necesita la tarea interactuar con una persona?.

Se puede hacer una distinción entre dos tipos de problemas:

- Los "solitarios", en los cuales a la computadora se le da una descripción del problema y ésta proporciona una respuesta sin ningún tipo de comunicación ni necesidad de explicaciones del proceso de razonamiento.
- Los "conversacionales", en los cuales existe una comunicación intermedia entre el hombre y la computadora, bien para proporcionar una ayuda adicional a la máquina, bien para que la computadora proporcione información al usuario.

8) La clasificación del problema.

Dependiendo de la granularidad con la cual se clasifiquen los problemas y las estrategias de control, surgen distintas listas de tareas genéricas y procedimientos. Si se analizan con cuidado los problemas y se clasifican los métodos de resolución de los mismos, atendiendo a la clase de problemas para los cuales son adecuados, podrán reutilizarse muchas cosas aprendidas en la anterior resolución de problemas similares.

Heurísticas de construcción y de mejoramiento.

Hay heurísticas que se utilizan para encontrar una solución del problema, tratando, desde ya de que ésta sea lo más próxima al óptimo que se pueda. A este tipo de heurísticas se las llama heurísticas de construcción. Otras heurísticas parten de una solución ya conocida y tratan de mejorarla para que se aproxime al óptimo. A este tipo de heurísticas se las conoce como heurísticas de mejoramiento. Las heurísticas de mejoramiento funcionan tanto mejor cuanto más alejada esté del óptimo la solución de la cual se parte (lógicamente, cuanto peor es una cosa, más fácil es mejorarla). Las heurísticas de construcción son aquellas para las cuales se estudia su garantía de calidad.

La mayoría de las heurísticas comunes (las que se nos ocurrirían a nosotros), son heurísticas de construcción. Es un hecho frecuente el crear heurísticas para tratar de llegar a una primera solución, ya que no tenemos ninguna, para un problema. Es el caso de las heurísticas que aplicamos todos los días, aún sin darnos cuenta para poder resolver nuestros problemas, como el hecho de ordenar una serie de trámites que tenemos que hacer, o las tareas que hacen a la preparación de nuestro desayuno. Crear heurísticas de mejoramiento es más complicado, primero, porque hay que conocer el problema (sino, no sabremos cuándo estamos mejorando) y segundo porque requieren mayor rigurosidad. Una mala heurística de construcción puede ser salvada con una buena heurística de mejoramiento, pero una mala heurística de mejoramiento implica un desconocimiento del problema que nos puede llevar a que ni siquiera nos demos cuenta de que la heurística no nos conduce a u resultado razonable. Volviendo a uno de los ejemplos que vimos anteriormente como heurísticas de construcción, una vez que hemos realizado una serie de trámites, si algún día tenemos que realizar en el mismo tiempo los mismos trámites, es de esperar que tengamos una idea acerca de qué cosas nos convendría volver a hacer de la misma forma que la vez anterior y cuáles nos convendría hacer de una forma diferente. Estamos aplicando una heurística de mejoramiento a la solución encontrada con nuestra heurística de construcción anterior.

Las heurísticas y el problema del viajante.

Recordemos el problema: Un viajante sale de una ciudad inicial (llamada ciudad de partida o ciudad cero) y debe recorrer N ciudades, pasando una sola vez por cada una de ellas y volviendo a la ciudad de partida. Hay varios recorridos posibles que puede hacer el viajante, se trata de encontrar el de menor distancia total o el de menor costo total. Al recorrido del viajante se lo llama "tour".

Aquí debemos encontrar el tour más barato, esto es, la ruta más barata que visite todas las ciudades pasando una sola vez por cada una de ellas y regrese a la inicial. Se puede ver al problema como un grafo de N nodos (ciudades), en el cual cada eje (ruta que une cada par de ciudades) tiene asignado un peso o costo no negativo (distancia entre el par de ciudades que une o costo de viajar entre esas dos ciudades). Usualmente es un grafo completo, es decir, uno en el cual todo nodo está vinculado con cada uno de los demás por exactamente un eje.

El problema del viajante es uno de los problemas de clase NP-Hard, como ya dijimos en otras oportunidades, con lo cual todos los algoritmos conocidos requieren un tiempo exponencial en el peor caso.

En la siguiente tabla se muestra, según la cantidad de ciudades que tenga el problema del viajante, la cantidad de tiempo que se demorará para encontrar en un computador la solución óptima, en el peor de los casos:

	Cantidad de ciudades:				
	10	20	30	50	60
Tiempo que demora en ser resuelto	0,001 segundos	1,0 segundos	17,9 minutos	35,7 años	366 siglos

Sin embargo, se pueden usar funciones límites para encontrar el tour óptimo en un tiempo menor. El asunto es encontrar las funciones límites. Para el problema del viajante, la solución de menor costo está dada por el costo del tour inicial (un determinado tour para el cual se comienza a analizar y que no contiene todas las ciudades) más el costo mínimo de completar el tour. Pero el problema de completar el tour es tan NP-Hard como el de encontrar el costo mínimo de todo el tour, entonces debemos establecer una estimación heurística del costo de completar el tour. Dadas tales estimaciones, la decisión de cuál tour parcial extender primero debe depender de, cuál de ellos, después de combinar los costos de la parte explorada con la estimación del costo de completarlo, origina la mejor solución.

Se puede ver que, si en cada etapa seleccionamos para extender el tour que tiene la menor estimación del costo para ser completado y, si la función heurística es optimística (es decir, es una estimación consistente del costo real de completar un tour), entonces, el primer tour parcial que es elegido para su extensión, es, una vez que se ha encontrado el tour completo, también el tour más barato. Lamentablemente, la función heurística que garantice siempre encontrar el tour de costo más barato, aún no ha sido hallada, porque todas dependen de las características y del tamaño del problema y aquellas que tienen garantía de calidad, no aseguran llegar al óptimo verdadero, como las de grafo recubridor y árbol generador de costo mínimo. Esta última tiene una garantía de calidad dos, porque lo que se hace es tratar al problema como un grafo, encontrando el arbol de expansión mínima y luego recorriéndolo con el método depth-first search (profundiza cada rama antes de pasar a otra) se recorre dos veces cada eje, con lo cual el costo final del recorrido del árbol termina siendo el doble del peso del árbol (porque pasa dos veces por cada vértice o nodo).

Existen algunas heurísticas de propósito general que son adecuadas para una amplia variedad de dominios de problemas. Además es posible construir heurísticas de propósito especial que exploten el conocimiento específico del dominio para resolver problemas particulares.

La heurística del vecino más próximo es un ejemplo de una buena heurística de propósito general válida para varios problemas combinatorios. Consiste en seleccionar en cada paso la alternativa localmente mejor. Al aplicarse al problema del viajante de comercio, surge el siguiente proceso:

1) Seleccionar arbitrariamente una ciudad de comienzo.

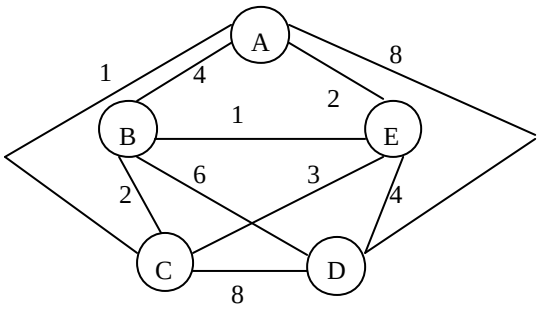
2) Para seleccionar la siguiente ciudad, fijarse en las ciudades que todavía no se han visitado y seleccionar aquella que sea más cercana. Ir a esa ciudad.

3) Repetir 2 hasta que todas las ciudades hayan sido visitadas.

Este procedimiento se ejecuta en un tiempo proporcional a N^2, lo cual es una mejora significativa frente a N!. Lo que sucede es que, dependiendo de las características de cada problema en particular (sobre todo en lo referente a qué tan uniforme es la distribución de las ciudades en el plano), la solución puede estar más o menos cerca de la óptima. Inclusive, en ocasiones esta heurística puede llevar a encontrar la solución exacta, aunque, si no sabemos cuál es, no nos vamos a dar cuenta de que llegamos a ella (los ensayos se hacen en general sobre problemas para los cuales ya se conoce el valor del óptimo o puede estimarse exactamente). Este tipo de algoritmos como el del vecino más próximo se llaman algoritmos heurísticos de tipo goloso o greedy, porque en cada paso hacen lo que es mejor para ese estado, sin tener en cuenta los estados posteriores y anteriores. Las heurísticas de este estilo son muy sencillas de implementar, pero la solución puede estar más alejada del óptimo de lo que podríamos llegar a tolerar.

Heurísticas de construcción para el problema del viajante.

Vamos a presentar un problema del viajante con el cual analizaremos varias heurísticas.



Hay distintos tipos de heurísticas de construcción para los problemas del viajante:

a) Heurísticas que van formando el tour. De este tipo veremos la del vecino más próximo.

b) Heurísticas que generan múltiples fragmentos de tour. Analizaremos la de emparchamiento más cercano.

c) Heurísticas que cierran el tour en cada paso. Daremos como ejemplo de esta clase la heurística de inserción más cercana.

d) Heurísticas basadas en árboles. Su fundamento consiste en ver la estructura del problema como un árbol. Algunas de ellas, como la del árbol generador mínimo, pueden encontrarse en libros acerca de la teoría de grafos.

Vecino más próximo (Nearest Neighbor).

1) Se empieza por un tour parcial trivial que contenga una ciudad cualquiera.
2) La próxima ciudad elegida es la más cercana a la última del tour siempre que no esté aún incluida en el tour.
3) Repetir el paso 2, hasta que todas las ciudades estén en el tour.

En nuestro ejemplo, si partimos de A, se elige el eje A - C, desde C, el eje de menor costo es que va hacia B. Desde B el menor eje es el que une con E. Una vez que estamos en E, el menor eje es el que une con A, pero cerraríamos el tour y aún no hemos incluido a D, con lo cual no queda otro remedio que conectar E con D y luego, para cerrar, unir D con A.

El tour es A - C - B - E - D - A y su costo es 16.

Emparchamiento más cercano (Multiple Fragment).

En esta heurística no se trata de formar un grafo sino de ir determinando ejes que no impidan cerrar el tour. Es mucho más complicado de implementar pero bastante fácil de enunciar:

1) Elegir el eje con menor costo.
2) Ir eligiendo de los ejes restantes el que tenga menor costo y me permita seguir agregando hasta formar un tour.

En nuestro ejemplo, el paso 1 consiste en elegir el eje A - C (podría ser el B - E también). Luego, lógicamente, hay que elegir A - E ó B - C.

Si elegimos A - E, estas dos ciudades ya tienen sus dos ejes, y B y C tienen un eje cada uno. D no tiene ninguno, con lo cual tanto B como C tienen que conectarse con ella. El tour que se forma es A - C - D - B - E - A cuyo costo es 18.

Si elegimos B - C en lugar de A - E, B y C quedan con dos ejes, A y E tienen un eje y D no tiene ninguna. Hay que conectar A con D y D con E. Así el tour queda A - C - B - E - D - A con un costo de 16.

Para poder implementar esta heurística es necesario contar con las siguientes estructuras de datos:

- Un arreglo o vector que en cada posición i contenga el número de ejes del tour adyacentes al nodo i (el 'grado' del nodo).
- Un árbol que contiene sólo puntos que se pueden elegir para el tour (que tengan grado 0 ó 1).
- Otro vector para el cual, cada una de sus posiciones i contiene el índice del vecino más cercano a i, que no sea el fragmento que contiene actualmente a ese vértice.
- Una cola de prioridad que contiene los enlaces con los vecinos más cercanos.
- Otro arreglo que representa los fragmentos. Si el punto i es de grado 1, el elemento i del arreglo, contiene el índice del punto que es el otro extremo del fragmento.

Inserción más cercana (Nearest Addition).

Este algoritmo se basa en ir construyendo (cerrando) tours parciales.

1) Empezar con un tour de una sola ciudad.
2) La próxima ciudad a agregar es la que menos cueste al tour parcial (teniendo en cuenta que para formar ese tour parcial se elimina un eje y se agregan dos, los que conectan con la ciudad que se agrega).
3) Se repite 2 hasta que no queden más ciudades por agregar.

En nuestro ejemplo:

a) Si empezamos con A agregamos la C con lo cual tenemos el tour A - C con un costo de 2 (1 de ida y 1 de vuelta).
b) Luego tenemos que agregar una ciudad que conecte con A (la de menor eje es E) o conecte con C (la de menor eje es B). Si agregamos E queda el tour A - C - E - A con un costo adicional de 2 (A - E) + 3 (C - E) y se quita un eje 1 (A - C). Si agregamos B queda A - C - B - A con un costo adicional de 2 (C - B) + 4 (B - A) y quitamos el mismo eje 1 (C - A). De esto resulta que conviene agregar E.
c) Al tour A - C - E - A debemos agregarle D ó B. En este punto disminuyen las posibilidades de ciudades a agregar pero tenemos más lugares en los cuales agregar los ejes (no es necesario conectarla con A).
   Si agregamos B conviene agregarla entre C y E porque se adiciona un eje de 2 (C - B) y uno de 1 (B - E) y eliminamos un eje de 3 (C - E). Si lo agregamos entre A y C el costo adicional es 4 (A - B) + 2 (B - C) y eliminamos un eje de 1 (A - C). Si lo agregamos entre A y E se adiciona un costo de 4 (A - B) + 1 (B - E) y se elimina un eje de 2 (A - E).
   Si agregamos D y lo hacemos entre A y E se adiciona un costo de 8 (A - D) + 4 (D - E) y se quita un eje de 2 (A - E) con lo que le cuesta 10 al tour. Si lo agregamos entre E y C se adiciona un costo de 4 (E - D) + 8 (D - C), quitando un eje de 3 (C - E), con lo cual le cuesta 9 al tour. Si se agrega entre A y C se adiciona un costo de 8 (A - D) + 8 (D - C) y se quita un eje de 1 (A - C) costándole 15 al tour.
   La opción que menos le cuesta al tour actual (cuesta 0) es agregar B entre C y E.
d) Al tour A - C - B - E - A debemos agregarle la ciudad D. Les dejamos a ustedes los cálculos de los costos para ver entre qué dos ciudades agregarla.

Con esta heurística, si se ha comprendido correctamente el concepto de "ciudad que menos cuesta al tour actual", como para aplicarlo en el punto d, queda un tour de costo total 15 ¿son capaces de encontrarlo?.

Nuevas heurísticas de construcción (otros criterios):

Heurística de barrido: Se rota una semirecta alrededor del centro del plano y la ubicación de las ciudades en el tour se determina a medida que la semirecta las toca.

Curvas de llenado: Mapean el plano a una línea. Es algo así como si las ciudades fueran clavos y el tour fuera una cuerda. La idea es ir recorriendo todos los clavos con la cuerda. Si los clavos fueran de colores y pintaran la cuerda del color de la ciudad que se va tocando, al quitar la cuerda tendríamos el orden de las ciudades. Uno de los algoritmos más conocidos de los que siguen este concepto es el de las curvas de Sierpinski.

Algoritmos Genéticos: En este caso, se interpreta cada solución como una secuencia de ceros y unos, poniendo un 1 en la posición i de la cadena si el eje i está en el tour óptimo en esa solución y 0 si no está.

Consiste en imitar mutaciones genéticas mejorando las soluciones mediante apareamiento de dos soluciones existentes, como si fueran cadenas de ADN que van generando otras cadenas.

Las últimas investigaciones van aún más allá. Leonard Adelman, de la Universidad del Sur de California resolvió el problema del viajante de comercio para 100 ciudades. Su razonamiento partió de la base de que el ADN forma secuencias químicas que son las que permiten codificar los rasgos hereditarios. Entonces, utilizando ingeniería genética, generó una molécula de ADN con una secuencia especial para cada ciudad y para cada trayecto y las metió en un tubo de ensayo para que se combinaran entre sí, siguiendo las leyes de la biología molecular. Se trabaja sobre la idea de que, tantas moléculas se generan, que alguna debe contener el itinerario correcto (se la reconoce por su ciudad de partida y llegada, porque no repite ninguna ciudad y por su longitud). Esa identificación se hace con métodos de ingeniería genética. Nos falta saber cómo hace el señor Adelman para reconocer la que tiene menor camino (el mejor camino), pero suponemos que lo hace por algún método de reconocimiento. En realidad, como dice un buen amigo mío, generar las soluciones es fácil, lo complicado es saber cuál de todas esas es la mejor.

Lo importante es que se sigue tratando de resolver estos problemas, de las más diversas formas. Más cosas hay en el cielo y en la tierra de las que conoce nuestra sabiduría.

Heurísticas de mejoramiento para el problema del viajante.

Hay diferentes tipos de heurísticas de mejoramiento. Las más usadas consisten en hacer intercambios de ejes, mejorando con cada una (se llaman k-intercambios). En base al problema antes mencionado y resuelto, explicaremos estas heurísticas. Mencionaremos luego otros tipos.

K-intercambios.

Dada una solución factible (un tour), trata de mejorarla cambiando k ejes de la solución. El caso más simple es el de 2-intercambios. Veamos el algoritmo:

1) Sea S la solución actual.
2) Sea S' la solución obtenida al hacer algún k-intercambio. S' es un vecino de S.
3) Si S' es mejor que S, definir S = S'.
4) Sino, determinar si existe otro k-intercambio que aún no fue examinado. Si hay, ir a 1, sino terminar.

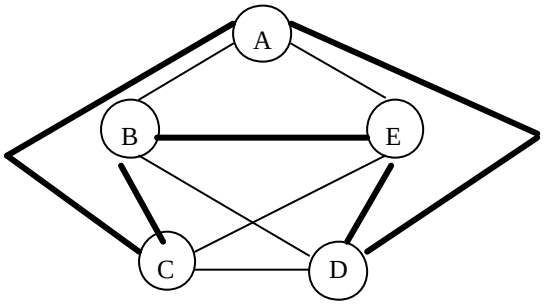
Consiste en moverse de vecino en vecino hasta encontrar una solución que no se pueda mejorar, es decir, un óptimo local.

A este tipo de heurísticas se las llama de optimización local. Depende de la forma de la función objetivo el hecho de que se pueda encontrar el óptimo global o no porque en estos algoritmos, si no se les implementa un mecanismo que permita 'soluciones peores' (que son las que hacen caer en un valle desde el cual podemos subir a una montaña más alta que la cima que hemos alcanzado hasta hora) sólo permiten pasar de un óptimo local al óptimo global por vecinos que incrementen la función objetivo.

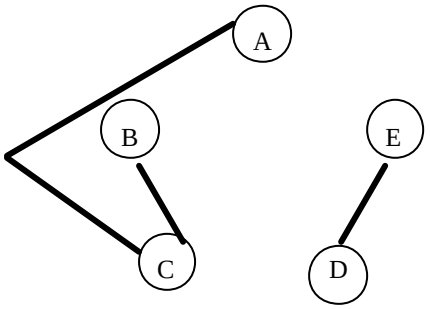
Hay $N \cdot (N - 1) / 2$ pares de ejes. La complejidad de cualquier k-intercambio es del orden n^k.

Vamos a aplicar 2-intercambio en nuestro problema del viajante. Ya que tenemos una solución en la cual el tour óptimo tiene un costo de 16, como la del vecino más próximo, podemos tratar de llegar a la solución de 15, que era la de inserción más cercana.

Recordemos la solución



Podemos quitar dos ejes, por ejemplo B - E y A - D, con lo cual eliminamos un costo de 1 + 8. Así quedará la siguiente solución:



pero nos quedó el tour incompleto. Hay que unir B con A o con D (no lo unimos con E porque volveríamos a la solución anterior). Si lo unimos con A cerramos el tour, así que lo tenemos que unir con D, con lo cual no nos queda otro remedio que unir A con E. Los ejes que agregamos tienen un costo de 6 + 2. El tour resultante tiene un costo total de 15 y es el mismo que queda cuando aplicamos inserción más cercana.

Si podemos encontrar un par de ejes que podamos reemplazar con otro par de ejes que tengan menor costo (sumados) que la suma de los costos del par de ejes que hemos quitado, se puede seguir mejorando.

Simmulated Annealing.

En esta técnica de mejoramiento, se aceptan intercambios que pueden NO mejorar la solución actual siempre que se cumpla con cierto parámetro de aceptación, regido por una función de probabilidad, que va decreciendo en función del tiempo (para que no siga probando infinitamente) imitando el proceso físico de "simulated annealing" (se usa para el enfriamiento de cristales). Es de fácil implementación pero de convergencia lenta y extremadamente dependiente de la forma en la cual decrece el parámetro de aceptación.

Las heurísticas y los problemas de secuenciamiento.

Los problemas de secuenciamiento se relacionan con el desarrollo de un orden exacto (o secuencia) de procesamiento de trabajos. En el secuenciamiento se calculan los tiempos de cada tarea para ver el tiempo total de la tarea global a realizar.

Uno de los métodos de secuenciamiento más antiguos es la gráfica de Gantt, creada por Henry Gantt en 1917. Es un cuadro en el cual el tiempo se encuentra en la abscisa y algún recurso escaso (maquinarias, gente en horas máquina, etc) se carga en la ordenada.

En la literatura se pone mucha atención en los algoritmos óptimos para el secuenciamiento de tareas. Lo que hacen estos algoritmos, normalmente, es optimizar una o más medidas de rendimiento de los programas, por ejemplo se puede optimizar el tiempo de fabricación. Sin embargo, usan una serie de premisas muy restrictivas como por ejemplo: tiempos de procesamiento constantes, para que no haya intereferencia de unos trabajos sobre otros, ni éstos se translapen. Un problema en particular complicado, es el de la programación de máquinas m x n, donde m es el número de máquinas y n el número de trabajos.

El problema de programación de máquinas m x n se ha resuelto para m = 1, 2, 3 y para algunos otros valores arbitrarios de n. No se han desarrollado todavía algoritmos eficientes para $m \geq 4$ porque hay muchas secuencias posibles. Este problema es más simple cuando m = 1 y se tiene cualquier valor de n. Esto sucede porque el tiempo de fabricación y la utilización son fijos. Sin embargo puede minimizar la suma de los tiempos de entrega o tiempos muertos de los trabajos. Esto se logra programando en primer lugar las tareas que tienen tiempo de procesamiento más corto. Así se reduce el tiempo que otros trabajos tienen que esperar para su procesamiento.

El problema de programación de máquinas con m = 2 es ligeramente más complicado. Si suponemos un orden en el cual los trabajos deben pasar por las dos máquinas (por ejemplo, primero por la máquina 1 y luego por la máquina 2), se simplifica bastante el problema. Un algorimo de resolución fue desarrollado por primera vez por S. M. Johnson y se denomina "regla de la mano izquierda - mano derecha". En este método primero se hace una lista en la cual los trabajos se colocan como filas y en cada una de esas filas se coloca a la izquierda el tiempo de procesamiento de ese trabajo en la máquina 1 y a la derecha el tiempo de procesamiento de ese trabajo en la máquina 2. Luego se revisa la lista de tiempos de los trabajos buscando el tiempo de procesamiento más corto. Si ese tiempo está en la columna de la derecha se coloca el trabajo al cual pertenece a la derecha (en orden) y se lo tacha de la tabla; si está a la izquierda se coloca el trabajo a la izquierda y se lo tacha de la tabla.

Las reglas de secuenciamiento son de gran interés teórico, sin embargo, no se han aplicado mucho en la práctica porque los problemas reales incluyen una gran cantidad de variables en tiempos de procesamiento, objetivos múltiples y otros factores que los complican.

Tipos de problemas de secuenciamiento.

a) El problema de secuenciamiento con máquinas distintas.

En este problema hay una serie de trabajos a ser procesados llamados $J_1 \ldots J_k$.

Cada uno de estos trabajos tiene m operaciones que llamaremos $J_{i1} \ldots J_{im}$. Cada operación debe hacerse en una determinada máquina. Hay m máquinas, una para cada operación, con lo cual la operación i de cada uno de los trabajos debe hacerse en la máquina i.

Cada operación i del trabajo j tiene un tiempo de procesamiento p_{ij}.

Dentro de cada trabajo hay un orden en el cual pueden efectuarse las operaciones. Ese orden puede ser el mismo para todos los trabajos o puede ser propio de cada trabajo.

El objetivo de este tipo de problemas es decidir el secuenciamiento de los trabajos en las máquinas (es decir, en qué orden deben pasar los trabajos por cada una de las máquinas) de forma tal de minimizar el tiempo total.

- Si la secuencia que deben seguir las operaciones de cada trabajo es la misma para todos los trabajos, el problema se llama **Flow shop scheduling**.
- Si para cada trabajo hay una secuencia propia de operaciones, no necesariamente igual a la de los otros trabajos, el problema se llama **Job-shop scheduling**.

Ambos problemas son NP-Hard.

Además, a ambos problemas se pueden agregar tiempos de espera para cada trabajo, r_i (release time) y tiempos mínimos para la entrega, d_i (due date).

Las secuencias pueden clasificarse como secuencias activas o no. Una secuencia es activa si ninguna operación puede ser empezada antes sin retrasar las demás tareas. Se puede demostrar que siempre existe una secuencia óptima que sea activa.

b) El problema de secuenciamiento de una sola máquina.

Ya habíamos dicho que este problema era el más sencillo de los problemas de secuenciamiento.

Se puede resumir como sigue:

Cada trabajo J_i tiene:

- Un tiempo de procesamiento p_i
- Un tiempo de espera r_i (puede pensarse como lo que tardó el trabajo en llegar hasta ese punto).

El objetivo es encontrar el tiempo mínimo en el cual pueden completarse todos los trabajos.

El problema se complica si existe un tiempo mínimo durante el cual debe quedarse el trabajo en el sistema antes de poder ser terminado de procesar.

c) El problema de secuenciamiento con costo de set-up.

Para ver una aplicación de este tipo de problemas, podemos suponer un medio de producción (una máquina sola) en el cual se deben procesar varios trabajos. En este tipo de problemas no es lo mismo hacer las tareas en cualquier orden. El tiempo de ejecución individual de las tareas no se modifica. Lo que pasa es que la producción tiene que detenerse después de ejecutar un trabajo para preparar el siguiente. Este "tiempo muerto" se llama tiempo de set-up o de instalación. Ese tiempo depende de cuál es el último trabajo que se realizó y cuál es el siguiente. Si ambos trabajos son semejantes, el tiempo será inferior que si los trabajos son muy distintos. Evidentemente, se darán cuenta de que no se trata de cualquier tipo de trabajos. Puede ser una máquina para preparar helados. Siguiendo este ejemplo, si luego de procesar helados de crema americana se procesan helados de crema vainilla, el tiempo de set-up será mucho menor que el de procesar helados de crema vainilla luego de los helados de chocolate (el tiempo de set-up sería el de limpieza y acondicionamiento de la máquina).

Desde el punto de vista combinatorio, con n trabajos hay n! combinaciones posibles. Si adoptamos el método de resolución por enumeración, calculando para cada secuencia de trabajo posible el tiempo total, encontraríamos el verdadero óptimo (la sucesión de trabajos con menor tiempo total) pero este método no es práctico. Esa es la razón por la cual, a menudo, en estos problemas se aplican reglas heurísticas.

Es un problema del viajante de comercio asimétrico (no es lo mismo c_{ij} que c_{ji}) en el cual las ciudades son los trabajos y los costos de ir de una ciudad a la otra son los tiempos de set-up. Por eso es que se aplican las mismas heurísticas que para los problemas del viajante.

Reglas de despacho.

En la práctica, los programas son difíciles de mantener, sino imposibles, debido a que las condiciones cambian con frecuencia, las máquinas se descomponen, se enferma un operador, etc. En este caso se utilizan reglas de despacho.

Una regla de despacho especifica qué trabajo debe seleccionarse para ser realizado después, entre una cola de trabajos. A diferencia de los programas, las reglas de despacho no pueden estar sin actualizarse y sirven para responder la pregunta inmediata del trabajador "¿qué hago ahora?".

Para poder indicar las reglas se necesita desarrollar criterios para el rendimiento del taller. Existen tres tipos de criterios: eficiencia de las máquinas y de la mano de obra, inventario de productos en proceso y servicio a clientes. Aunque sólo se trata de tres criterios, existen diversas formas de medirlos.

Algunas mediciones comunes, si tomamos el ejemplo de programación de tareas en un taller que tiene una serie de máquinas para realizar trabajos que le permitan cumplir los pedidos, son: número de pedidos terminados, porcentaje de pedidos terminados tarde, número promedio de pedidos que están esperando, tiempo de espera promedio de los pedidos, porcentaje de mano de obra utilizada, porcentaje de la capacidad de máquinas utilizada. Estas son algunas mediciones usuales pero de todos modos siempre los resultados dependen de las características del problema.

Heurísticas para el planeamiento de producción (secuenciamiento).

Para el problema de secuenciamiento con máquinas distintas podemos ver un algoritmo para la generación de secuencias activas.

t = tiempo

S_t = tareas que pueden se secuenciadas en el tiempo t.

S = operaciones que ya han sido secuenciadas.

Paso 1.

t = 0

Paso 2.

Se determina la máquina disponible en el menor tiempo $\leq$ t (si hay más de una elegir cualquiera) y se la llama m*

Paso 3.

Si hay alguna operación en S_t, para cada operación de S_t que puede hacerse en esa máquina se calcula un índice de prioridad. Se elige la operación que tenga *el mejor índice.*

Paso 4.

Se secuencia la operación j agregándola a S en la máquina m*.

Paso 5.

t = t + 1

Volver a Paso 2.

Para determinar el índice de elección de una tarea se usan las reglas de despacho que son técnicas heurísticas. Veamos algunos ejemplos de heurísticas de despacho:

SPT (Shortest processing time): Se elige la operación con menor tiempo de procesamiento. Esta regla se basa en la idea de que cuando se termina con rapidez una operación de un trabajo, otras máquinas más adelante recibirán el trabajo, dando como resultado una alta velocidad de flujo y una alta utilización.

FCFS (First come, first served): Esta regla se basa en el criterio familiar de "justicia", según el cual., la tarea que llega primero al centro de trabajo se procesa primero.

MWKR (Most work remaining): Se elige la operación asociada al trabajo al cual le queda el mayor tiempo de procesamiento, sumando todas las operaciones de ese trabajo que restan.

LWKR (Lowest work remaining): Se elige la operación asociada con el trabajo al cual le queda el menor tiempo de procesamiento.

MOPNR (Most operation remaining): Se elige la operación que tiene el mayor número de operaciones que le siguen.

En la práctica, la que mejor funciona es MWKR, pero no hay ninguna que siempre dé una solución mejor que todas las demás.

Bibliografía:

🕮 Administración de Operaciones - Roger Schröeder
🕮 Aplicaciones de Investigación Operativa. Casos reales. - Sebastián Ceria.
🕮 Heuristics - Judea Pearl
🕮 Fast Algorithms for Geometric Traveling Salesman problems - J. J. Bentley.
🕮 Investigación de Operaciones en la Ciencia Administrativa - Eppen - Gould.
🕮 Inteligencia Artificial - Elaine Rich y Kevin Knight.
🕮 Inteligencia Artificial – Patrick Henry Winston.