Complejidad Computacional – Tarea 6

Cova Pacheco Felipe de Jesús

Actividades: 1. Leer el artículo An Overview of Heuristic Solution Methods (de E. A.Silver). 2. Elaborar un escrito en donde describas a detalle las 9 metaheur ısticaspresentadas en el art ıculo. Adem as deber as indicar y justificar a cu altipo de heur ıstica b asica pertenecen.

Profesor: María de Luz Gasca Soto
Ayudante: José Luis Vázquez Lázaro

Cova Pacheco Felipe de Jesús

Introducción

Una metaheurística es un método heurístico para resolver un tipo de problema computacional general, usando los parámetros dados por el usuario sobre unos procedimientos genéricos y abstractos de una manera que se espera eficiente. Normalmente, estos procedimientos son heurísticos.

Las metaheurísticas generalmente se aplican a problemas que no tienen un algoritmo o heurística específica que dé una solución satisfactoria; o bien cuando no es posible implementar ese método óptimo. La mayoría de las metaheurísticas tienen como objetivo los problemas de optimización combinatoria, pero por supuesto, se pueden aplicar a cualquier problema que se pueda reformular en términos heurísticos, por ejemplo en resolución de ecuaciones booleanas.

Las metaheurísticas no son la panacea y suelen ser menos eficientes que las heurísticas específicas, en varios órdenes de magnitud, en problemas que aceptan este tipo de heurísticas puras.

El objetivo de la optimización combinatoria es encontrar un objeto matemático finito (por ejemplo, un vector de bits o permutación) que maximice (o minimice, dependiendo del problema) una función especificada por el usuario de la metaheurística. A estos objetos se les suele llamar estados, y al conjunto de todos los estados candidatos se le llama espacio de búsqueda. La naturaleza de los estados y del espacio de búsqueda son usualmente específicos del problema.
Tomemos en cuenta las siguientes metaheurísticas presentadas a continuación:

Multilevel refinement

La idea de la técnica de niveles múltiples es reducir la magnitud de una gráfica mediante la combinación de vértices, calcular una partición en esta gráfica reducido y, finalmente, proyectar esta partición en la gráfica original.

En la primera fase, la magnitud de la gráfica se reduce al fusionar los vértices. La fusión de vértices se realiza de forma iterativa: de un gráfica se crea una nueva gráfica más gruesa y de esta nueva gráfica más gruesa se crea una gráfica aún más gruesa. Esto se hace hasta que se alcanza una cierta magnitud pequeña. Así se inducen gráficas con diferentes magnitudes.

En la segunda fase, una partición de la gráfica con la menor magnitud, la gráfica más gruesa, es calculada.

En la tercera y última fase, la partición calculada se proyecta de forma iterativa hacia la gráfica original. En cada iteración se aplica una heurística de refinamiento. La fusión de vértices induce un mapa entre los vértices de una gráfica y los vértices de su gráfica más grueso que se utiliza para la proyección posterior. Es posible que se necesite un rebalanceo para asegurar el tamaño de la partición, ya que los vértices que no pertenecen a la misma partición pueden ser combinados.

La técnica multinivel ha demostrado mejorar significativamente los resultados, tanto en términos de calidad como de tiempo de ejecución. Especialmente cuando se utiliza en heurísticas considerando la gráfica solo localmente, ya que la técnica de niveles múltiples constituye una visión más global de la gráfica.

Dado que la solución va trascendiendo de una manera paso a paso, el método que le caracteriza es el constructivo.

Búsqueda Tabú

La búsqueda tabú es un algoritmo metaheurístico que puede utilizarse para resolver problemas de optimización combinatoria, tales como el problema del viajante (TSP, del inglés Travelling Salesman Problem). La búsqueda tabú utiliza un procedimiento de búsqueda local o por vecindades para moverse iterativamente desde una solución x hacia una solución x' en la vecindad x hasta satisfacer algún criterio de parada. Para poder explorar regiones del espacio de búsqueda que serían dejadas de lado por el procedimiento de búsqueda local, la búsqueda tabú modifica la estructura de vecinos para cada solución a medida que la búsqueda progresa. Las soluciones admitidas para $N^*(x)$, el nuevo vecindario, son determinadas mediante el uso de estructuras de memoria. La búsqueda entonces progresa moviéndose iterativamente de una solución x hacia una solución x' en $N^*(x)$.

Quizás la estructura de memoria más importante usada para determinar las soluciones permitidas a un $N^*(x)$, sea la lista tabú. En su forma más simple, una lista tabú es una memoria de corto plazo que contiene las soluciones que fueron visitadas en el pasado reciente
(menos de n iteraciones atrás, donde n es el número de soluciones previas que van a ser almacenadas (n también es llamado el tenor del tabú)). La búsqueda tabú excluye las soluciones en la lista tabú de $N^*(x)$. Una

variación de la lista tabú prohíbe soluciones que tienen ciertos atributos (i.e., soluciones al problema del agente viajero (TSP) que incluyen aristas no deseadas) o prevenir ciertos movimientos (i.e., un arco que fue agregado a un recorrido del TSP no puede ser eliminado en los siguientes n movimientos). Los atributos seleccionados de las soluciones recientemente visitadas son denominados "tabú-activos." Las posibles soluciones que contengan elementos tabú-activos son "tabú".

Las listas tabú que contienen atributos pueden ser más efectivas para algunos dominios, pese a que presentan un nuevo problema. Cuando sólo un atributo es marcado como tabú, esto por lo general resulta en que más de una solución es marcada como tabú. Algunas de estas soluciones, que ahora deben ser evitadas, podrían ser de excelente calidad y no serían visitadas. Para mitigar este problema, se introducen los "criterios de aspiración": estos pueden modificar el estado de tabú de una solución, por lo tanto incluyendo la antes excluida solución en el conjunto de soluciones permitidas. Un criterio de aspiración muy utilizado es admitir soluciones que son mejores que la mejor solución conocida al momento.

Es un método de mejora local, ya que de igual manera, trabaja con los vecinos.

Búsqueda Beam

El proceso de búsqueda de una solución óptima supone calcular las distancias (el costo) acumuladas a lo largo de los múltiples posibles caminos que nacen en cada punto de decisión local. Todos esos caminos que parten de cualquier punto de decisión, constituyen un haz de caminos.
Beam Search es una versión modificada o restringida del algorítmo de búsqueda Best-First (El primero, el mejor). Se denomina restringido porque la cantidad de memoria disponible para guardar los nodos posibles está limitada

La idea básica es podar o recortar esos caminos del haz y la forma más sencilla de hacerlo es aplicar un umbral de poda o recorte a esos caminos. El umbral es un umbral de distancias, y los caminos que superen el umbral, serán recortados.

Beam search utiliza de forma similar la búsqueda en amplitud (anchura) para construir su árbol de búsqueda. En cada nivel del árbol, genera todos los sucesores de los estados en el nivel actual, la clasificación por orden de costo en la heurística aumenta. Sin embargo, sólo se almacena un número

predeterminado de mejores estados en cada nivel (llamado el ancho del haz). Solo esos estados se expanden. Cuanto mayor sea el ancho del haz, menor el número de estados que se podan. Búsqueda Beam no es óptima, (es decir, no hay garantía de que va a encontrar la mejor solución).
Devuelve la primera solución encontrada.

Es una heurística que va reduciendo el espacio de solución, y esto por el sustento de las restricciones.

Recocido Simulado

El objetivo general de este tipo de algoritmo es encontrar una buena aproximación al valor óptimo de una función en un espacio de búsqueda grande. A este valor óptimo se lo denomina "óptimo global".
El nombre e inspiración viene del proceso de recocido del acero y cerámicas, una técnica que consiste en calentar y luego enfriar lentamente el material para variar sus propiedades físicas. El calor causa que los átomos aumenten su energía y que puedan así desplazarse de sus posiciones iniciales (un mínimo local de energía); el enfriamiento lento les da mayores probabilidades de recristalizar en configuraciones con menor energía que la inicial (mínimo global).

En cada iteración, el método de recocido simulado evalúa algunos vecinos del estado actual s y probabilísticamente decide entre efectuar una transición a un nuevo estado s' o quedarse en el estado s. En el ejemplo de recocido de metales descrito arriba, el estado s e podría definir en función de la posición de todos los átomos del material en el momento actual; el desplazamiento de un átomo se consideraría como un estado vecino del primero en este ejemplo. Típicamente la comparación entre estados vecinos se repite hasta que se encuentre un estado óptimo que minimice la energía del sistema o hasta que se cumpla cierto tiempo computacional u otras condiciones.

El vecindario de un estado s está compuesto por todos los estados a los que se pueda llegar a partir de s mediante un cambio en la conformación del sistema. Los estados vecinos son generados mediante métodos de Montecarlo.
El método de evaluación de estados vecinos es fundamental para encontrar una solución óptima global al problema dado. Los algoritmos heurísticos, basados en buscar siempre un estado vecino mejor (con energía más baja) que el actual se detienen en el momento que encuentran un mínimo local de energía. El problema con este método es

que no puede asegurar que la solución encontrada sea un óptimo global, pues el espacio de búsqueda explorado no abarca todas las posibles variaciones del sistema.

Es un método de mejora local, ya que al igual que la búsqueda tabú, coincide en que hace una mejora local y continúa haciendo mejoras completas sobre los vecinos.

Búsqueda de vecindad variable

La idea básica es ir cambiando en forma sistemática la vecindad al momento de realizar la búsqueda.
Sea N_k un conjunto finito de vecindades predefinidas y $N_k(x)$ el conjunto de soluciones en la k-ésima vecindad de x.
La mayoría de los algoritmos de búsqueda local usan una sola vecindad, y las vecindades a veces se pueden inducir a partir de una o más métricas que se tengan en el espacio de solución.
VNS se basa en tres hechos simples:

1. Un mínimo local con respecto a una estructura de vecindad no lo es necesariamente con respecto a otra.
2. Un mínimo global es un mínimo local con respecto a todas las posibles estructuras de vecindades.
3. En muchos problemas el mínimo local con respecto a una o varias estructuras de vecindad están relativamente cerca.

La última observación es empírica e implica que un mínimo local muchas veces nos de información acerca del óptimo global.

Debido a que se va eligiendo el siguiente elemento para obtener el mejor beneficio, cumple con la expectativa de ser un método constructivo.

Búsqueda local guiada

Es una extensión de los algoritmos de búsqueda local como Hill Climbing y es similar en estrategia al algoritmo de búsqueda de tabú y al algoritmo de búsqueda local iterado.

La estrategia para el algoritmo de búsqueda local guiada es utilizar penalizaciones para alentar a una técnica de búsqueda local a escapar de los óptimos locales y descubrir los óptimos globales. Se ejecuta un algoritmo de búsqueda local hasta que se atasca en un óptimo local. Las

características de los óptimos locales se evalúan y penalizan, cuyos resultados se utilizan en una función de costo aumentado empleada por el procedimiento de búsqueda local. La búsqueda local se repite varias veces utilizando el último óptimo local descubierto y la función de costo aumentado que guía la exploración fuera de las soluciones con características presentes en el óptimo local descubierto.

Es un método constructivo pues de igual manera que el método descrito anteriormente, tan pronto éste se renueva, el óptimo mejora.

Ant colony search

Este tipo de metaheurística simula el comportamiento de una colonia de hormigas de la siguiente manera: En nuestro mundo natural, las hormigas (inicialmente) vagan de manera aleatoria, al azar, y una vez encontrada comida regresan a su colonia dejando un rastro de feromonas. Si otras hormigas encuentran dicho rastro, es probable que estas no sigan caminando aleatoriamente, puede que estas sigan el rastro de feromonas, regresando y reforzándolo si estas encuentran comida finalmente.

Sin embargo, al paso del tiempo el rastro de feromonas comienza a evaporarse, reduciéndose así su fuerza de atracción. Cuanto más tiempo le tome a una hormiga viajar por el camino y regresar de vuelta otra vez, más tiempo tienen las feromonas para evaporarse. Un camino corto, en comparación, es marchado más frecuentemente, y por lo tanto la densidad de feromonas se hace más grande en caminos cortos que en los largos. La evaporación de feromonas también tiene la ventaja de evitar convergencias a óptimos locales. Si no hubiese evaporación en absoluto, los caminos elegidos por la primera hormiga tenderían a ser excesivamente atractivos para las siguientes hormigas. En este caso, el espacio de búsqueda de soluciones sería limitado.

Por tanto, cuando una hormiga encuentra un buen camino entre la colonia y la fuente de comida, hay más posibilidades de que otras hormigas sigan este camino y con una retroalimentación positiva se conduce finalmente a todas las hormigas a un solo camino. La idea del algoritmo colonia de hormigas es imitar este comportamiento con "hormigas simulada" caminando a través de un grafo que representa el problema en cuestión.

Debido a que cada hormiga comienza por su propia cuenta, pero a fin, éstas terminan uniendose por un mismo camino, a mi conclusión, cumple con ser un método de particionamiento.

Cova Pacheco Felipe de Jesús

Algorítmos Evolutivos

Estos algoritmos se modelan computacionalmente simulando la selección natural y el entrecruzamiento de las especies por medio de la recombinación genética y la mutación. En ellos se mantienen un conjunto de entidades que representan posibles soluciones, las cuales se mezclan y compiten entre sí, de manera que las más aptas son capaces de prevalecer a lo largo del tiempo, evolucionando hacia mejores soluciones cada vez.
En la naturaleza lo único que hay que optimizar es la supervivencia, y eso significa a su vez maximizar diversos factores y minimizar otros. Un algoritmo evolutivo, sin embargo, se usará habitualmente para optimizar sólo una función, no diversas funciones relacionadas entre sí simultáneamente. Se han aplicado con éxito en problemas de optimización, búsqueda, aprendizaje y simulación de sistemas dinámicos, en problemas con espacios de búsqueda extensos y no lineales donde otros métodos no son capaces de encontrar soluciones en un tiempo razonable.
Esta gran familia de algoritmos está compuesta por distintos métodos de búsqueda entre los que se pueden mencionar, la Programación Genética, que consiste en la evolución automática de se programa usando ideas basadas en la selección natural, se busca que poblaciones de programas evolucionen transmitiendo su herencia de manera que se adapten mejor al medio, los individuos tienen mayores probabilidades de reproducirse, la medida de la calidad del individuo dependerá del tipo de problema.
La Programación Evolutiva que se basa en la optimización de funciones y es una abstracción de la evolución al nivel de las especies, por lo que no se requiere el uso de un operador de recombinación (diferentes especies no se pueden cruzar entre sí). Los algoritmos evolutivos presentan una estructura que puede aplicarse en distintos problemas, favoreciendo así grandemente las tareas de diseño e implementación. Como único requisito del usuario que desee aplicar esta técnica para resolver un problema concreto es saber programar en cualquier lenguaje de propósito general en el que codificaría el algoritmo evolutivo.

En este método cabe destacar que cada que la evaluación se ve actualizada, se puede decir que se van descartando soluciones, por lo tanto es un método que reduce espacio.