Optimización

Primer Cuatrimestre 2025

Práctica N° 8: Algoritmos aproximados.

Ejercicio 1 El Problema de la Mochila 0/1 consiste en seleccionar un subconjunto de ítems, cada uno con un peso w_i y un valor v_i, para maximizar el valor total dentro de una mochila con capacidad máxima W. La restricción es que cada ítem debe ser tomado completamente o no ser tomado en absoluto. Formalmente, si x_i es una variable binaria tal que $x_i = 1$ si el ítem i es seleccionado y $x_i = 0$ si no lo es, el problema puede formularse como:

$$\begin{aligned} \text{Maximizar:} \quad & \sum_{i=1}^{n} v_i x_i \\ \text{Sujeto a:} \quad & \sum_{i=1}^{n} w_i x_i \leq W \\ & x_i \in \{0,1\} \quad \text{para todo } i = 1, \ldots, n. \end{aligned}$$

La heurística greedy para este problema funciona de la siguiente manera:

- Calcular la relación valor/peso (v_i/w_i) para cada ítem.
- Ordenar los ítems de mayor a menor según esta relación.
- Seleccionar iterativamente los ítems en este orden, incluyéndolos si no exceden la capacidad de la mochila, hasta que no quede espacio o no haya más ítems.

(a) Proponer un conjunto de ítems (peso y valor) y una capacidad W para la mochila donde la heurística greedy no produzca la solución óptima.

(b) Modificar el ejemplo anterior para encontrar ejemplos donde el factor de aproximación sea arbitrariamente malo.

Ejercicio 2 Se requiere asignar un conjunto de tareas con duraciones predefinidas a **dos máquinas** idénticas que operan en paralelo. El objetivo es minimizar el Makespan, definido como el tiempo total que la última máquina en terminar su carga de trabajo toma para completar todas sus tareas asignadas. Es decir, es la duración máxima de trabajo entre las dos máquinas. Cada tarea debe ser asignada completamente a una única máquina.

Formalmente, dado un conjunto de tareas $T = \{t_1, t_2, \ldots, t_n\}$, con t_i representando la duración de la tarea i. Buscamos particionar T en dos subconjuntos T_1 (tareas de la Máquina 1) y T_2 (tareas de la Máquina 2) tal que $T_1 \cup T_2 = T$, $T_1 \cap T_2 = \emptyset$, y se minimice:

$$\text{máx} \left(\sum_{t \in T_1} t, \sum_{t \in T_2} t \right)$$

Para este problema, aplicaremos la heurística greedy LPT, que funciona de la siguiente manera:

1. **Ordenar las tareas:** Las tareas se ordenan en orden **descendente** según su duración.

2. **Asignación iterativa:** Recorriendo las tareas en el orden previamente establecido, cada tarea se asigna a la máquina cuya carga de trabajo acumulada actual sea la **menor**. En caso de empate en la carga de trabajo actual, la tarea se asigna a la Máquina 1.

Encuentre un ejemplo de duración de las tareas donde la heurística greedy no sea óptima. ¿Cuán lejos está del óptimo?

Ejercicio 3 El Problema del Viajante de Comercio (TSP) se existen N ciudades que forma un grafo completo (todas conectadas entre sí). Los costos (o distancias) de viaje entre cada par de ciudades son no negativos. El objetivo del problema es encontrar una ruta que, partiendo de una ciudad designada, visite cada una de las otras $N-1$ ciudades exactamente una vez y luego regrese a la ciudad de origen, de manera que el costo total de la ruta sea minimizado.

Considere una heurística greedy para el TSP con distancias simétricas, basada en ir hacia la ciudad no visitada más cercana. Construya un contraejemplo de cómo la heurística puede no conducir al óptimo global. (Sugerencia: alcanza con considerar 4 ciudades). Modifique su ejemplo para que el factor de aproximación sea arbitrariamente malo.

Ejercicio 4 Se desea construir una heurística de búsqueda local para el TSP. Se parte de una solución inicial válida, por ejemplo, la generada por la heurística greedy o una ruta aleatoria. Para ello, se consiera la siguiente operación conocida como 2-opt, que implica elegir dos puntos de corte en la ruta (aristas), invertir el tramo de ciudades entre esos dos puntos, y luego reconectar la ruta.

El algoritmo de búsqueda local procede a examinar todas las posibles operaciones 2-opt sobre la ruta actual. Si alguna de estas operaciones 2-opt resulta en una ruta con un costo total menor, se elige la que produce la mayor reducción de costo. La ruta actual se actualiza con esta nueva ruta mejorada.

Aplique el algoritmo de búsqueda local a los ejemplos construidos en el ejercicio anterior. Detalle cada paso de manera exhaustiva.

Ejercicio 5 El Problema de las N-Reinas es un problema clásico de satisfacción de restricciones en el ajedrez, que consiste en colocar N reinas en un tablero de ajedrez de $N \times N$ casillas de manera que ninguna reina ataque a otra. En este ejercicio, lo reformularemos como un problema de optimización para aplicar una heurística de búsqueda local: el objetivo será minimizar el número de pares de reinas que se atacan mutuamente.

Diseñe un algoritmo de búsqueda local para resolver este problema. Su diseño debe incluir:

- **Representación de la Solución:** ¿Cómo se representaría una configuración de N reinas en el tablero bajo la hipótesis de üna reina por columna¿ (Sugerencia: Piense en un vector o lista).

- **Función Objetivo:** ¿Cómo se calcula el çosto"de una configuración (es decir, el número de pares de reinas que se atacan)? Recuerde que dos reinas se atacan si están en la misma fila o en la misma diagonal.

- **Definición de Vecindad:** Proponga un movimiento simple para generar soluciones vecinas. (Sugerencia: Mover una única reina a otra casilla dentro de su misma columna).

- **Lógica General del Algoritmo de Búsqueda Local:** Describa el proceso iterativo que seguiría el algoritmo, desde una solución inicial (ej., aleatoria) hasta alcanzar un óptimo local (minimizar los ataques).

Ejercicio 6 Dado un grafo no dirigido $G = (V, E)$, el problema del corte máximo (Max Cut) consiste en particionar sus vértices en dos subconjuntos disjuntos. El objetivo es maximizar el número de aristas que tienen un extremo en cada uno de los dos subconjuntos (es decir, las aristas que cruzan el corte).

Diseñe un algoritmo de búsqueda local para resolver este problema. Su diseño debe incluir:

- **Representación de la Solución:** ¿Cómo se representaría una partición de los vértices en dos subconjuntos V_1 y V_2?
- **Función Objetivo:** ¿Cómo se calcula el "valor"(el número de aristas en el corte) de una partición? (Recuerde que se busca maximizar este valor).
- **Definición de Vecindad:** Proponga un movimiento simple para generar soluciones vecinas. (Sugerencia: Considere mover un único vértice de un subconjunto (V_1 o V_2) al otro subconjunto).
- **Lógica General del Algoritmo de Búsqueda Local:** Describa cómo el algoritmo procedería para encontrar un óptimo local (maximizar el corte), desde una solución inicial (ej., asignación aleatoria de vértices a V_1 o V_2).