1 Fuerza Bruta

Se comprueba todas las posibles combinaciones de k empresas y devuelve la primera combinación que cubre todas las tecnologías requeridas por cada empresa en C. Si no se encuentra tal combinación, la función devuelve None.

1.1 Correctitud

La correctitud de este algoritmo se basa en la propiedad de que si alguna combinación de las empresas puede proporcionar todas las tecnologías, la encontrará, porque itera sobre todas las posibles combinaciones de empresas.

1. El algoritmo comienza generando todas las combinaciones posibles de k empresas.
2. Para cada combinación, el algoritmo recoge todas las tecnologías proporcionadas por las empresas en esa combinación y verifica si esas tecnologías son exactamente las tecnologías requeridas S.
3. Si encuentra una combinación que proporciona exactamente S, entonces devuelve esa combinación y termina.
4. Si ninguna combinación satisface el requisito, el algoritmo devuelve None.

Por lo tanto, si existe una solución, el algoritmo la encontrará. Si no existe una solución, el algoritmo también lo determinará correctamente.

1.2 Complejidad temporal

La complejidad temporal de este algoritmo es $O(n^k)$, donde n es el número de empresas y k es el tamaño de los subconjuntos que estamos considerando.

El cálculo de la complejidad se basa en la cantidad de subconjuntos de k empresas que se generan, que es $C(n,k) = \frac{n!}{(n-k)!k!}$ en el peor de los casos. En la práctica, como k es una constante, podemos simplificar esto a $O(n^k)$.

Si para cada subconjunto, unimos todas las tecnologías, lo que puede tener un coste de hasta $O(n)$. Por lo tanto, la complejidad temporal total del algoritmo puede considerarse $O(n^{k+1})$.

2 Algoritmo de Búsqueda Tabú

En este caso utilizamos la Búsqueda Tabú, una metaheurística que es eficaz para explorar el espacio de soluciones y evitar quedarse atrapado en mínimos locales. Para demostrar la correctitud y analizar la complejidad temporal y espacial de este algoritmo, necesitamos considerar varios puntos clave, pero primero exploremos un poco en el código:

- La función `generar_vecindario` crea un conjunto de soluciones vecinas cambiando una compañía en la solución actual por otra compañía que no esté presente en la solución.
- La función `evaluar_solucion` evalúa una solución contando el número total de tecnologías únicas que puede cubrir.
- La función `tabu_search` es la implementación principal de la búsqueda tabú. Inicialmente, crea una solución aleatoria y mantiene una lista tabú para almacenar soluciones que no deben ser consideradas en futuras iteraciones. Luego, en cada iteración:
  - Genera un vecindario de soluciones a partir de la solución actual.
  - Descarta las soluciones que están en la lista tabú.
  - Evalúa las soluciones restantes y elige la mejor solución vecina.
  - Actualiza la solución actual y la mejor solución encontrada hasta el momento.
  - Actualiza la lista tabú, agregando la mejor solución vecina y eliminando la solución más antigua si la lista excede su tamaño máximo.
- El algoritmo se ejecuta durante un número máximo de iteraciones y devuelve la mejor solución encontrada.

2.1 Correctitud

La Búsqueda Tabú se puede considerar correcta porque implementa un procedimiento sistemático de exploración del espacio de soluciones que garantiza que no se quede atrapado en óptimos locales. Esta garantía proviene de su uso de la lista tabú, que evita que el algoritmo retroceda a soluciones recientemente visitadas.

2.2 Complejidad temporal

La complejidad de tiempo de la Búsqueda Tabú depende del número de iteraciones (`max_iter`), el tamaño del vecindario de cada solución, y el tiempo requerido para evaluar una solución y comparar soluciones.

- Generar el vecindario: Esto implica generar todas las soluciones que difieren de la solución actual al reemplazar una empresa por otra. Dado que esto se hace para cada empresa en la solución actual y para cada empresa en C, la complejidad de tiempo de esta etapa es $O(k \cdot |C|)$, donde $|C|$ es el número total de empresas, y k es el tamaño de la solución.
- Evaluar el vecindario: Para cada solución en el vecindario, calculamos su valor llamando a `evaluar_solucion`, que tiene una complejidad de tiempo de $O(k)$, ya que implica unir los conjuntos de tecnologías de las empresas en la solución. Esto se hace para cada solución en el vecindario, por lo que la complejidad de tiempo total para evaluar el vecindario es $O(k^2 \cdot |C|)$.

- Elegir la mejor solución del vecindario: Esto implica ordenar las soluciones en el vecindario por su valor, lo cual tiene una complejidad de tiempo de $O(k \cdot |C| \log(k \cdot |C|))$ utilizando un algoritmo de ordenación.

Estos pasos se repiten en cada iteración del algoritmo, por lo que la complejidad de tiempo total es $O(\texttt{max\_iter} \cdot k \cdot |C|(k + \log(k \cdot |C|)))$.

3 Reducción a un problema NP (SAT)

Para demostrar que el problema es NP-completo, realizaremos una reducción del problema SAT al problema.

3.1 Definición del problema SAT

Dada una expresión booleana en forma normal conjuntiva (conjuntos de cláusulas, donde cada cláusula es una disyunción de literales), el problema SAT pregunta si existe alguna asignación de valores de verdad a las variables que haga que toda la expresión sea verdadera.

3.2 Definición del problema

Dada una lista de tecnologías S y una lista de empresas, cada una con un conjunto de tecnologías requeridas C, y un número k, ¿existe alguna selección de k empresas tal que la unión de las tecnologías requeridas por las empresas seleccionadas contenga todas las tecnologías requeridas por cualquier otra empresa en la lista?

3.3 Prueba

Primero, demostraremos que el problema está en NP. Dado una muestra (es decir, una selección de k empresas), podemos verificar en tiempo polinómico si la unión de las tecnologías requeridas por las empresas seleccionadas contiene todas las tecnologías requeridas por cualquier otra empresa en la lista. Por lo tanto, el problema está en NP.

Para demostrar que el problema es NP-completo, reduciremos el problema SAT al problema inicial.

3.4 Reducción de SAT al problema

Dada una instancia del problema SAT con n variables y m cláusulas, creamos una instancia del problema de la siguiente manera:

1. Para cada variable x_i en SAT, creamos dos tecnologías en el problema: t_i y t'_i (representando la variable y su negación, respectivamente).

2. Para cada cláusula c_j en SAT, creamos una empresa en el problema que requiere las tecnologías correspondientes a los literales en c_j.

3. Establecemos k igual a m (el número de cláusulas en SAT).

Esta reducción se puede realizar en tiempo polinómico.

3.5 Correctitud de la reducción

Supongamos que existe una asignación de valores de verdad que satisface la fórmula SAT. Entonces, podemos seleccionar un conjunto de k empresas en el problema de manera que la unión de sus tecnologías requeridas cubra todas las tecnologías requeridas por todas las empresas en C. Cada variable verdadera en la asignación SAT corresponde a seleccionar la tecnología correspondiente en el problema, y cada variable falsa en la asignación SAT corresponde a seleccionar la tecnología complementaria. Como la asignación satisface todas las cláusulas en SAT, hemos cubierto todas las tecnologías requeridas por todas las empresas en C.

A la inversa, supongamos que podemos seleccionar un conjunto de k empresas en el problema tal que la unión de sus tecnologías requeridas cubra todas las tecnologías requeridas por todas las empresas en C. Entonces, podemos construir una asignación de valores de verdad que satisface la fórmula SAT seleccionando cada variable para ser verdadera si su tecnología correspondiente fue seleccionada en el problema, y falsa de lo contrario.

Por lo tanto, el problema es NP-completo.