Informe: Solución Óptima al Problema de Selección de Intervalos de Máxima Cardinalidad

Nicolás Paz Reyes, Martín Moloeznik y Luca Bogado

19 de octubre de 2025

1 Descripción Matemática del Problema

El problema consiste en, dado un conjunto finito C de n intervalos cerrados sobre la recta real, encontrar un subconjunto de C de máxima cardinalidad cuyos intervalos sean disjuntos.

Formalmente:

- Un conjunto $C = \{(a_i, b_i) \mid a_i, b_i \in \mathbb{R}, a_i \leq b_i\}$.
- Dos intervalos $I_1 = (a_1, b_1)$ e $I_2 = (a_2, b_2)$ son disjuntos (compatibles) si no se superponen. Asumiendo la convención de intervalos semiabiertos (a, b), la compatibilidad se define como: $b_1 \leq a_2$ o $b_2 \leq a_1$.
- Debemos encontrar un subconjunto $C' \subseteq C$ tal que:
  1. **Validez:** $\forall I, J \in C', (I \text{ y } J \text{ son disjuntos})$.
  2. **Optimalidad:** $|C'|$ es máximo. Es decir, para cualquier otro subconjunto $S \subseteq C$ que cumpla la condición de validez, se tiene que $|S| \leq |C'|$.

2 Estrategia Greedy y Solución Propuesta

Para resolver este problema de optimización, proponemos una estrategia *greedy* que toma decisiones localmente óptimas en cada paso, con la expectativa de alcanzar una solución globalmente óptima.

La Estrategia: "Elegir el que Termina Antes"

Nuestra estrategia se basa en la intuición de que, para maximizar el número de intervalos seleccionados, debemos liberar el "recurso" (los subintervalos) lo antes posible. El criterio de selección será siempre el intervalo que se encuentre más a la izquierda, por ende podremos ir analizando todas las regiones del espacio en donde potencialmente puedan haber subconjuntos de intervalos consecutivos disjuntos posibles.

El algoritmo procede de la siguiente manera:

1. **Ordenar:** Ordenar el conjunto C de n intervalos en orden ascendente, según su **tiempo de finalización** (b_i).
2. **Seleccionar:** Inicializar la solución G con el primer intervalo de C (aquel con el b_i mínimo global). Sea $g_{b_{menor}}$ este intervalo.
3. **Iterar y Filtrar:** Recorrer C_o (a partir del intervalo *siguiente* a $g_{\text{b\_menor}}$) hasta encontrar el *primer* intervalo $I = (a, b)$ que sea compatible con $g_{\text{b\_menor}}$ (es decir, $a > b_{\text{menor}}$).
4. **Volver o terminar:**
   - **Si se encontró** dicho intervalo I: Añadir I a G, actualizar $g_{b_{menor}} = I$, y **volver al Paso 3** (continuando la búsqueda desde el intervalo *siguiente* a I).
   - **Si no quedan intervalos** por revisar (o no se encuentran más intervalos compatibles), **Terminar**. G es la solución final.

Esta estrategia garantiza que, en cada paso, la elección *greedy* es la que elije la máxima cantidad de intervalos disjuntos.

3 Demostración de Optimalidad

A continuación, demostramos formalmente que la estrategia *greedy* descripta ("Elegir el que Termina Antes") produce siempre una solución de cardinalidad máxima.

Teorema: El algoritmo *greedy* (G) encuentra una solución óptima.

Definiciones:

- Sea $G = \{g_1, g_2, \ldots, g_k\}$ la solución encontrada por nuestro algoritmo *greedy*, con k intervalos. Los intervalos están ordenados por su tiempo de finalización, de modo que $b_{g1} \leq b_{g2} \leq \cdots \leq b_{gk}$. (Donde $g_i = (a_{gi}, b_{gi})$).
- Sea $O = \{o_1, o_2, \ldots, o_m\}$ una solución **óptima** cualquiera, con m intervalos, también ordenados por su tiempo de finalización ($b_{o1} \leq b_{o2} \leq \cdots \leq b_{om}$).

Objetivo: Demostrar que $k = m$.

Por la definición de optimalidad, G no puede ser más grande que O, por lo tanto, sabemos que $m \geq k$. La prueba se centrará en demostrar que $m = k$.

3.1 Paso a Paso de la Demostración

1. Caso Base: Soluciones Idénticas Si la solución *greedy* G y la solución óptima O son idénticas, entonces $k = m$ y ya hemos demostrado que G es óptima. Para el resto de la demostración, es decir, los demás casos, asumimos que $G \neq O$.

2. Encontrar la Primera Diferencia Si las soluciones no son idénticas, debe existir un primer intervalo en el que difieren. Sea i el primer índice tal que $g_i \neq o_i$. Esto implica que los intervalos anteriores sí coinciden: $g_1 = o_1, g_2 = o_2, \ldots, g_{i-1} = o_{i-1}$.

3. La Propiedad Clave de Greedy En el paso i, nuestro algoritmo *greedy* eligió el intervalo g_i. Lo hizo del conjunto de todos los intervalos disponibles que no se superponían con g_{i-1}. El intervalo o_i (de la solución óptima) también pertenece a ese conjunto de candidatos válidos, ya que o_i es compatible con o_{i-1}, y $o_{i-1} = g_{i-1}$. Por la propia regla de selección del algoritmo (elegir siempre el que finaliza antes, es decir, el "b menor"), es imposible que o_i termine antes que g_i. Formalmente:

$$b_{gi} \leq b_{oi}$$

Esta es la propiedad fundamental de nuestra estrategia, que usaremos a continuación.

4. Demostración por Argumento de Intercambio Ahora, creemos una nueva solución O' a partir de la óptima O, reemplazando o_i por g_i:

$$O' = \{o_1, \ldots, o_{i-1}, g_i, o_{i+1}, \ldots, o_m\}$$

Verificación de Validez de O': Debemos demostrar que O' sigue siendo un conjunto de intervalos disjuntos.

1. **Compatibilidad con anteriores ($j < i$):** g_i es compatible con g_{i-1}, y como $g_{i-1} = o_{i-1}$, g_i es compatible con o_{i-1}.
2. **Compatibilidad con posteriores ($j > i$):** Debemos probar que g_i es compatible con o_{i+1}.
   - Sabemos que O es válida, por lo que $a_{oi+1} \geq b_{oi}$.
   - Por nuestra **Propiedad Clave (Paso 3)**, sabemos que $b_{gi} \leq b_{oi}$.
   - Combinando ambas: $a_{oi+1} \geq b_{oi} \geq b_{gi}$.
   - Esto demuestra que g_i es compatible con o_{i+1}.

Optimalidad de O': O' es una solución válida y tiene tamaño m. Por lo tanto, O' también es una solución óptima.

Conclusión del Intercambio: Hemos demostrado que si O difiere de G, podemos transformarla en G paso a paso sin alterar su optimalidad. Esto implica que $|G| = |O|$, es decir, $k = m$.

5. Demostración por Contradicción (El caso k + 1) Para solidificar la conclusión $k = m$, demostramos que $m > k$ conduce a una contradicción.

Supongamos para llegar al absurdo, que la solución óptima es estrictamente más grande que la solución greedy: $m > k$.

Esto trae como **consecuencia** que si $m > k$ es válido, debe existir un intervalo o_{k+1} en la solución óptima O. Como O es válida, o_{k+1} debe ser compatible con o_k, es decir, $a_{ok+1} \geq b_{ok}$.

La Contradicción: Aplicamos nuestra **Propiedad Clave (Paso 3)**. Esta propiedad se mantiene para todo $i \leq k$. Por lo tanto, para $i = k$, tenemos $b_{gk} \leq b_{ok}$.
Combinando los pasos:

$$a_{ok+1} \geq b_{ok} \geq b_{gk}$$

Esto implica que $a_{ok+1} \geq b_{gk}$. Esta desigualdad significa que o_{k+1} **era un candidato compatible** con g_k (el último intervalo que el *greedy* seleccionó).

Sin embargo, el algoritmo *greedy* se detuvo después de g_k. Por definición (Paso 4 de nuestra estrategia), esto solo ocurre cuando no quedan más intervalos compatibles.
La existencia de o_{k+1} (un intervalo compatible que el *greedy* no seleccionó) **contradice la condición de finalización del algoritmo**.

4 Conclusión Final

La suposición ($m > k$) es falsa. Dado que $m \geq k$ y $m \not> k$, la única posibilidad restante es que $m = k$. Hemos demostrado mediante el argumento de intercambio que la solución greedy es óptima. Además, hemos reforzado esta conclusión probando por contradicción que ninguna solución puede ser estrictamente más grande. Por lo tanto, la estrategia greedy de elegir el intervalo que finaliza primero produce siempre una solución de máxima cardinalidad. **Es óptima**.

5 Análisis del Programa y Complejidad

El algoritmo *greedy* descripto (Pasos 1-4) se implementó en Haskell de la siguiente forma.

```haskell
type Intervalo = (Int, Int)

compararIntervalo :: Intervalo -> Intervalo -> Ordering
compararIntervalo (_, b1) (_, b2) = compare b1 b2

seleccionarIntervalos :: [Intervalo] -> [Intervalo]
seleccionarIntervalos [] = []
seleccionarIntervalos intervalos = reverse (go (tail intervalosOrdenados) [head intervalosOrdenados])
where
 -- Ordenar por tiempo de finalización
 intervalosOrdenados = sortBy compararIntervalo intervalos

 -- Función recursiva que implementa la selección
 go :: [Intervalo] -> [Intervalo] -> [Intervalo]
 go [] acumulador = acumulador
 go (actual:resto) (ultimoSeleccionado:acc) =
 let inicioActual = fst actual
     finUltimo = snd ultimoSeleccionado
 in
 -- Criterio Greedy (inicioActual >= finUltimo)
   if inicioActual >= finUltimo
     then go resto (actual : ultimoSeleccionado : acc)
     else go resto (ultimoSeleccionado : acc)
```

Figure 1: Nuestra implementación del algoritmo en Haskell.