Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 1

La mafia de los algoritmos Greedy

1er cuatrimestre - 2025

Nathalia Lucía Encinoza Vilela
106295

Iván Erlich
105989

Chiara López Angelini
111225

1. Resumen

En este trabajo se presenta una solución al problema de verificación de transacciones sospechosas utilizando un algoritmo greedy. El objetivo es determinar si una serie de transacciones realizadas por un sospechoso coinciden con los intervalos de tiempo aproximados de transacciones fraudulentas. Se demuestra la correctitud del algoritmo, se analiza su complejidad temporal, y se verifica su rendimiento a través de pruebas y mediciones.

2. Introducción

El problema planteado consiste en verificar si un conjunto de n transacciones realizadas por un sospechoso coinciden con n transacciones sospechosas de las cuales se conoce un timestamp aproximado t_i y un margen de error e_i. Cada transacción sospechosa se representa como un intervalo $[t_i - e_i, t_i + e_i]$, y se busca determinar si cada uno de los timestamps exactos s_i de las transacciones del sospechoso pueden asignarse a uno de estos intervalos. La particularidad del problema es que estos intervalos pueden solaparse parcial o totalmente, lo que añade complejidad a la hora de asignar correctamente cada transacción del sospechoso a un intervalo.

3. Análisis del problema

3.1. Formalización del problema

Dados:

- n timestamps aproximados t_i con sus respectivos errores e_i, formando intervalos $[t_i - e_i, t_i + e_i]$.
- n timestamps exactos s_i ordenados de menor a mayor.

El objetivo es determinar si existe una asignación única tal que cada s_i pertenezca a algún intervalo $[t_j - e_j, t_j + e_j]$, donde cada intervalo se utiliza exactamente una vez.

3.2. Enfoque Greedy

Para resolver este problema, se diseñó un algoritmo greedy que procesa cada transacción sospechosa secuencialmente y, para cada una:

1. Identifica todos los intervalos sospechosos en los que podría encajar.
2. Selecciona el intervalo con el tiempo de finalización más temprano.
3. Asigna la transacción a ese intervalo y lo elimina de la lista de intervalos disponibles.

El óptimo local que sigue el algoritmo utiliza la estrategia "elegir primero el que termina antes". La idea detrás de esta elección greedy es que al seleccionar el intervalo que termina más temprano, maximizamos la flexibilidad para las transacciones futuras. Es decir, al elegir intervalos que terminan antes, dejamos más "espacio temporal" disponible para asignar las siguientes transacciones sospechosas.

Esta estrategia funciona correctamente para este problema porque:

- Garantiza que si hay una solución válida, el algoritmo la encontrará.
- Permite resolver los casos de superposición de intervalos de manera óptima.
- Es consistente con la demostración de correctitud basada en intercambios.

Este enfoque contribuye a alcanzar el óptimo global (una asignación completa y válida, si existe).

3.3. Algoritmo propuesto

1. Transformar cada timestamp aproximado y error en un intervalo de la forma $(t_i - e_i, t_i, t_i + e_i)$.
2. Ordenar estos intervalos por tiempo de inicio $(t_i - e_i)$.
3. Para cada timestamp del sospechoso s_i:
   a) Identificar todos los intervalos que contienen a s_i.
   b) Si no hay intervalos que contengan a s_i, concluir que el sospechoso no es culpable.
   c) Entre los intervalos candidatos, seleccionar el que tiene el menor tiempo de finalización $(t_i + e_i)$.
   d) Asignar s_i a ese intervalo y eliminarlo de la lista de intervalos disponibles.
4. Si todos los timestamps del sospechoso pueden ser asignados, concluir que el sospechoso es culpable.

```python
def check_suspicius_transactions(n, transactions_with_error,
    suspicious_transactions):
    res = []

    transactions_with_error = [(t[0] - t[1] if t[0] - t[1] > 0 else 0, t[0], t[0] +
     t[1], t) for t in transactions_with_error]

    transactions_with_error.sort(key=lambda x: x[0])

    for i in range(n):
        actual_suspicious_transaction = suspicious_transactions[i]
        for i in range(len(transactions_with_error)):
            actual_transaction = transactions_with_error[i]
            if suspicious_transaction_is_in_range(actual_suspicious_transaction,
    actual_transaction):
                transactions_candidates.append((actual_transaction, i))

        if not transactions_candidates:
            return NOT_THE_SUSPECT
        final_candidate, index = tie_break_candidates(transactions_candidates)
        res.append((actual_suspicious_transaction, final_candidate))
        transactions_with_error = [t for i, t in enumerate(transactions_with_error)
     if i != index]

    return res
```

```python
def tie_break_candidates(transactions_candidates):
    '''
    Given a list of transactions candidates, it returns the one with the smallest
    finish time

    Args:
        - transactions_candidates: list of tuples (transaction, index).
    E.g. [((1, 2, 3, (2, 1)), 0), ((2, 3, 4, (2, 1)), 1), ...]

    Returns:
        - The transaction with the smallest finish time and its index.
    E.g. ((1, 2, 3, (2, 1)), 0)
    '''
    first_to_finish = transactions_candidates[0][0][2]
    first_to_finish_transaction = transactions_candidates[0]

    for i in range(len(transactions_candidates)):
        actual_transaction_finish_time = transactions_candidates[i][0][2]
        if actual_transaction_finish_time < first_to_finish:
            first_to_finish = actual_transaction_finish_time
            first_to_finish_transaction = transactions_candidates[i]

    return first_to_finish_transaction[0][3], first_to_finish_transaction[1]
```

3.4. Demostración

Para demostrar que el algoritmo propuesto determina correctamente si los timestamps del sospechoso corresponden a los intervalos sospechosos, debemos probar que:

1. Si existe una asignación válida, el algoritmo la encuentra.
2. Si el algoritmo encuentra una asignación, esta es válida.
3. Si no existe una asignación válida, el algoritmo lo detecta correctamente.

3.4.1. Caso 1: si existe una asignación válida, el algoritmo la encuentra

Supongamos que existe una asignación válida $A = (s_1, I_{j1}), (s_2, I_{j2}), ..., (s_n, I_{jn})$ donde cada s_i está asignado a un intervalo $I_{ji} = [t_{ji} - e_{ji}, t_{ji} + e_{ji}]$.

Consideremos la asignación A' que construye nuestro algoritmo greedy. Probaremos por inducción que A' también es una asignación válida.

Base: para la primera transacción sospechosa s_1, el algoritmo identifica todos los intervalos que la contienen y selecciona el que tiene el menor tiempo de finalización. Como existe una asignación válida A, al menos un intervalo debe contener a s_1. Por lo tanto, el algoritmo asignará s_1 a algún intervalo, aunque podría no ser el mismo que en A.

Paso inductivo: supongamos que el algoritmo ha asignado correctamente las primeras $k-1$ transacciones sospechosas. Para la transacción s_k, hay dos posibilidades:

1. El intervalo I_{jk} (el que le corresponde a s_k en la asignación A) aún está disponible. En este caso, s_k se asignará a algún intervalo disponible (posiblemente I_{jk} u otro con tiempo de finalización menor).

2. El intervalo I_{jk} ya ha sido asignado a una transacción anterior s_m (con $m < k$). Esto significa que en nuestro algoritmo, cuando procesamos s_m, el intervalo I_{jk} era un candidato válido y tenía el menor tiempo de finalización entre todos los candidatos disponibles en ese momento. En la asignación A, s_m está asignado a I_{jm}, lo que implica que I_{jm} también era un candidato válido para s_m en nuestro algoritmo. Dado que nuestro algoritmo eligió I_{jk} en lugar de I_{jm}, sabemos que $t_{jk} + e_{jk} \leq t_{jm} + e_{jm}$. En la asignación A, s_k está asignado a I_{jk} y s_m a I_{jm}. Podemos crear una nueva asignación válida A'' intercambiando estas asignaciones: s_m se asigna a I_{jk} (como lo hace nuestro algoritmo) y s_k se asigna a I_{jm}. Para que A'' sea válida, necesitamos verificar que s_k está dentro de I_{jm}, lo cual es cierto porque $s_k \leq s_m$ (ya que procesamos las transacciones en orden) y s_m está dentro de I_{jk}, y I_{jk} termina antes o al mismo tiempo que I_{jm}.

Por lo tanto, si existe una asignación válida, nuestro algoritmo encontrará una (posiblemente diferente pero igualmente válida).

3.4.2. Caso 2: si el algoritmo encuentra una asignación, esta es válida

Por construcción, cada transacción sospechosa s_i solo se asigna a un intervalo I_j si $s_i \in I_j$, y cada intervalo se utiliza a lo sumo una vez. Por lo tanto, cualquier asignación que encuentre el algoritmo es válida.

3.4.3. Caso 3: si no existe una asignación válida, el algoritmo lo detecta correctamente

Si en algún momento el algoritmo no encuentra ningún intervalo disponible que contenga a la transacción sospechosa actual s_i, entonces devuelve "No es el sospechoso correcto". Esto es correcto porque si no hay manera de asignar s_i a un intervalo disponible, no puede existir una asignación válida para todas las transacciones.

Por lo tanto, hemos demostrado que el algoritmo es correcto en todos los casos.

3.5. Complejidad

Analizaremos la complejidad temporal de las diferentes partes del algoritmo:

1. Creación de intervalos: $O(n)$.

2. Ordenamiento de intervalos por tiempo de inicio: $O(n \log n)$.

3. Para cada transacción sospechosa (bucle externo): $O(n)$.

   a) Búsqueda de intervalos candidatos (bucle interno): $O(n)$.

b) Selección del mejor candidato (`tie_break_candidates`): $O(n)$.

c) Eliminación del intervalo seleccionado: $O(n)$.

Luego, la complejidad total es: $O(n) + O(n \log n) + O(n) \times (O(n) + O(n) + O(n)) = O(n) + O(n \log n) + O(n^2) = O(n^2)$

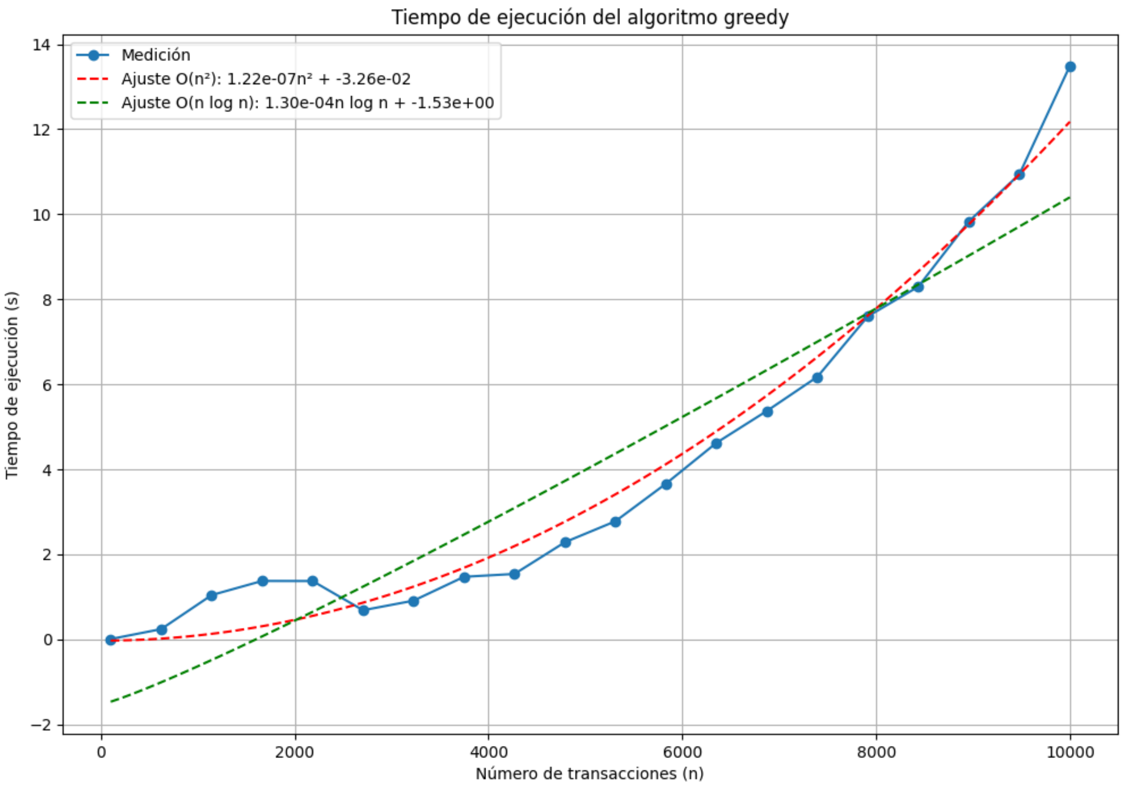
4. Mediciones

4.1. Comportamiento del algoritmo

Para verificar la complejidad temporal del algoritmo, realizamos una serie de mediciones con diferentes tamaños de entrada. Generamos conjuntos de datos aleatorios y medimos el tiempo de ejecución del algoritmo para cada uno.

La siguiente imagen muestra el tiempo de ejecución del algoritmo greedy en función del número de transacciones (n). Se pueden observar las siguientes características:

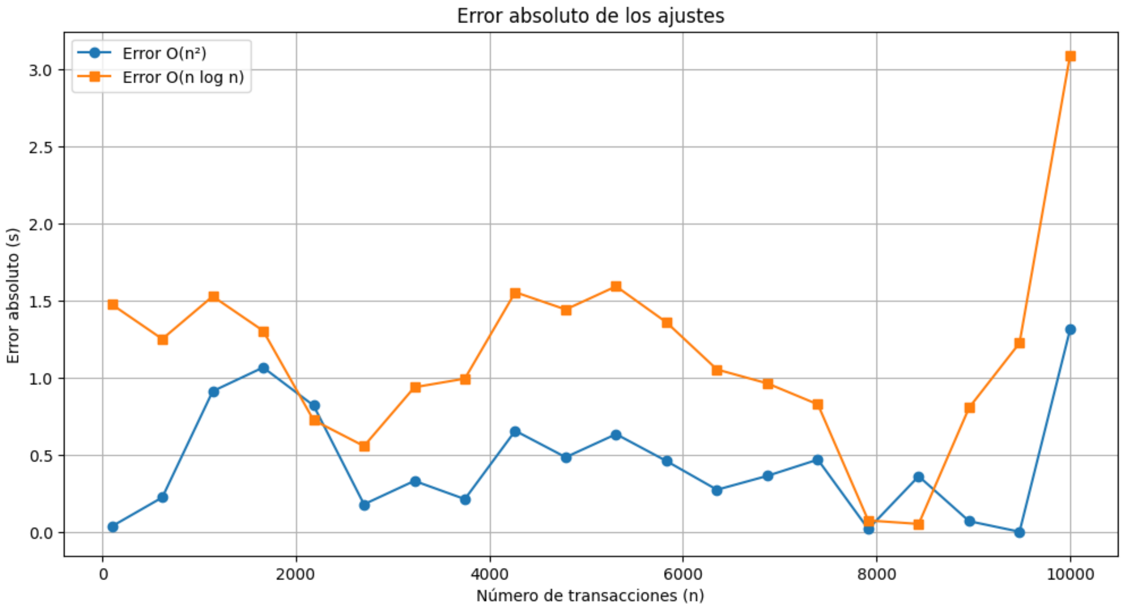
- Los tiempos de ejecución siguen una tendencia creciente no lineal.
- Para tamaños pequeños de entrada (aproximadamente hasta n = 3000), el crecimiento es más lento, pero luego aumenta de forma más pronunciada para entradas mayores, llegando a aproximadamente 13.5 segundos para n = 10000.



4.2. Análisis de errores

La siguiente imagen muestra el error absoluto entre los tiempos medidos y los tiempos predichos por ambos modelos teóricos:

- Error del ajuste $O(n^2)$: se mantiene generalmente por debajo de 1 segundo, con excepción del último punto donde alcanza aproximadamente 1.3 segundos.
- Error del ajuste O(n log n): muestra mayor variabilidad, oscilando entre 0.5 y 1.6 segundos en la mayoría de los puntos, pero aumenta significativamente hasta aproximadamente 3.1 segundos para n = 10000.



4.3. Conclusión

El ajuste $O(n^2)$ parece representar mejor el comportamiento real del algoritmo greedy, especialmente para tamaños de entrada grandes, donde el error medio es menor.

5. Conclusiones

En este trabajo se presentó una solución al problema de verificación de transacciones sospechosas utilizando un enfoque greedy basado en la selección del intervalo con menor tiempo de finalización. Se demostró la correctitud del algoritmo y se verificó su complejidad temporal de $O(n^2)$.

La solución cumple con los requisitos del problema y permite determinar de manera precisa si un conjunto de transacciones coincide con los intervalos sospechosos, ayudando así a identificar al culpable de los robos.