Tarea: Planificación de Rutas de Drones con Zonas de Exclusión

Teoría de la Computación

Noviembre 2025

Objetivo

El objetivo de esta tarea es demostrar que un problema de planificación logística pertenece a la clase NP-completo, mediante una reducción polinomial desde un problema clásico conocido como NP-completo. En particular, se analizará un problema inventado.

Problemas a estudiar

A: SAT

Definido formalmente en la hoja de ejercicios.

B: Planificación de Rutas de Drones con Zonas de Exclusión

Una empresa de vigilancia aérea utiliza drones autónomos para inspeccionar distintos puntos de control distribuidos en una zona industrial. Cada punto debe visitarse, pero algunas áreas son incompatibles entre sí por razones de interferencia electromagnética o seguridad: si un dron sobrevuela una zona determinada, no puede sobrevolar ninguna zona que esté marcada como incompatible con ella.

El dron tiene una autonomía de vuelo limitada (en minutos o distancia total) y cada punto tiene un valor de prioridad que indica su importancia para la misión.

El objetivo es determinar si existe una ruta factible que cumpla las siguientes condiciones:

- Comienza y termina en la base de operaciones.
- La distancia total recorrida no excede un límite máximo (M).
- La suma total de prioridades de los puntos visitados alcanza o supera un valor mínimo (V).
- No se visitan puntos que se encuentren dentro de zonas mutuamente excluyentes.

Entrada formal: Se proporciona:

- Un conjunto de puntos de control ($P = p_1, p_2, \ldots, p_n$).
- Una matriz de distancias simétrica ($D : P \times P \to N$), con ($D(p_i, p_i) = 0$).
- Una función de distancias hacia la base ($r : P \to N$)
- Un conjunto de pares de exclusión ($E \subseteq P \times P$), donde si ($(p_i, p_j) \in E$), entonces no pueden visitarse ambos en la misma ruta.

- Una función de prioridad ($b : P \to N$).
- Un límite de distancia total ($M \in N$).
- Una prioridad mínima requerida ($V \in N$).

La pregunta de decisión es:

¿Existe una ruta que cumpla todas las condiciones anteriores?

Parte 1: Verificadores y Reducción en Haskell

1.1 Representación de dominios y soluciones

Definir en Haskell los siguientes tipos:

```
type DomA = undefined
type SolA = undefined

type DomB = undefined
type SolB = undefined
```

1.2 Verificadores en tiempo polinomial

Implementar las siguientes funciones en Haskell:

```
verifyA :: (DomA, SolA) -> Bool
verifyB :: (DomB, SolB) -> Bool
```

Justificar formalmente que ambas funciones pueden evaluarse en tiempo polinomial respecto al tamaño de la entrada.

1.3 Resolución en tiempo exponencial

Implementar funciones que devuelvan una solución si existe, utilizando un algoritmo de exploración:

```
solveA :: DomA -> SolA
solveB :: DomB -> SolB
```

Estas funciones no deben ser eficientes; su objetivo es ilustrar que la solución puede encontrarse por fuerza bruta y ser verificada en tiempo polinomial.

1.4 Reducción polinomial entre problemas

Demostrar que B es NP-Conpleto. Definir una estrategia para la reducción de instancias de A a instancias de B:

```
reduceAToB :: DomA -> DomB
```

No es necesario implementarla, pero sí describirla formalmente. Debe justificarse que es efectivamente una reducción polinomial según la definición vista en el curso.

1.5 Ejemplos de instancias de prueba

A continuación se muestran dos instancias simples del problema SAT :

- **Ejemplo 1 (satisfacible):**

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_3)$$

- **Ejemplo 2 (insatisfacible):**

$$(x_1) \wedge (x_2 \vee x_3) \wedge (\neg x_1)$$

Muestre con qué instancia de B se corresponden

Parte 2: Verificadores y Reducción en otros modelos

2.1 Verificador de SAT en Imp

Implementar en Imp puro el siguiente programa para la verificación de SAT:

$\underline{def}\quad VERIFY_{SAT}\quad (clausulas,\quad valuacion)\quad \underline{return}\quad res$

Que devuelve un booleano correspondiente a la verificación de la instancia *clausulas* sobre la valuación *valuacion*. Calcular su complejidad temporal y probar que es polinomial

2.2 Verificador de B en χ

Implementar en χ puro una función que verifique el problema B. Explicar dicha función y probar que su complejidad temporal es polinomial.

2.3 Codificación de la Reducción en Máquina de Turing

Definir un alfabeto (Σ) y una codificación de las instancias de `DomA` y `DomB` como cadenas sobre (Σ). Especificar una Máquina de Turing que, al recibir como entrada una codificación de una instancia de `A`, produzca como salida la codificación de una instancia de `B` tal que:

- Si la instancia de `A` es satisfacible, entonces la instancia de `B` tiene una solución.
- Si la instancia de `A` no es satisfacible, entonces la instancia de `B` no tiene una solución.

Entrega

El contenido de la entrega consistirá en:

- Un archivo Haskell de nombre `Solucion.hs` con el código solicitado
- Un documento PDF de nombre `Informe.pdf` con los siguientes requerimientos:
  - Documento realizado en LaTeX(sugerimos usar Overleaf), siguiendo el template de IEEE (https://www.overleaf.com/latex/templates/ieee-conference-template/grfzhhncsfqn)
  - Todas las decisiones tomadas deberán ser debidamente documentados
  - El informe debe ser autocontenido y autoexplicativo.

- Se deberá citar fuentes externas conforme a lo establecido en los reglamentos de la universidad (Citas IEEE)
- En caso de usar IA/IAG se deberá citarlo incluyendo los links a los chats, o incluyendo las imágenes que considere relevantes. En caso de detectar que no se cumple este punto, se podrá anular la entrega y dar aviso a coordinación.

La fecha de entrega límite será la noche anterior al parcial (7/12 23:55) por Aulas.

Se permitirán entregas de **hasta dos estudiantes del mismo dictado**, siempre que ambos suban su entrega a aulas.

Esta entrega tendrá una defensa que se realizará junto al parcial