Optimización del Horario de Estudio para Exámenes Finales con Energía Limitada

Mayo 2025

1. Descripción del Problema

En el entorno académico universitario, los estudiantes suelen enfrentar semanas de alta carga académica durante los exámenes finales. En estos periodos, deben decidir cómo distribuir su tiempo de estudio entre varias materias para obtener el mejor rendimiento posible.

Cada materia requiere una cierta cantidad de horas para ser comprendida adecuadamente y tiene un impacto distinto en el promedio final. Sin embargo, el tiempo y la energía del estudiante son recursos limitados.

Este reto consiste en determinar cuántas horas dedicar a cada materia en un solo día, bajo una restricción de tiempo máximo disponible (por ejemplo, 6 horas), con el objetivo de maximizar el "beneficio académico" (porcentaje esperado de mejora en la calificación).

2. Relevancia y Criterio de Novedad

Aunque existen soluciones clásicas al problema de la mochila, esta formulación adapta la estructura al contexto real del estudiante, considerando horas de estudio y mejora esperada. No se ha encontrado una solución publicada que aborde este caso específico de planificación diaria de estudio con programación dinámica, lo que lo convierte en un enfoque novedoso. Además, permite adaptar el modelo a diferentes perfiles según niveles de energía, dificultad de las materias, o carga cognitiva.

3. Equivalencia con Problema Clásico de PD

Este problema es una versión del clásico problema de la mochila 0/1. Cada materia equivale a un *item*: tiene un valor (mejora en el rendimiento académico) y un peso (horas necesarias de estudio). El límite de horas es la capacidad de la mochila. La programación dinámica permite resolver este problema dividiendo la decisión en estudiar o no una materia dada cierta capacidad restante, optimizando desde los subproblemas menores hasta alcanzar la solución global.

4. Bosquejo de Estados y Tabla de Memoización

- **Estados**:
  - i: Índice de la materia (0 a $n-1$)
  - t: Horas restantes disponibles (0 a T)
- **Función de recurrencia**:

$$dp[i][t] = \max(dp[i-1][t],\ dp[i-1][t - horas_i] + valor_i), \quad \text{si } horas_i \leq t$$

- **Caso base**:

$$dp[0][t] = 0 \quad \forall t \in [0, T]$$

- **Tabla de memoización**: Matriz de tamaño $(n+1) \times (T+1)$ donde se almacenan los máximos beneficios posibles para cada subproblema.

La elección de los estados se basa en identificar qué decisiones debe tomar el estudiante: por cada materia, decidir si estudiarla o no dado el tiempo disponible. Esta estructura conduce naturalmente a un enfoque de PD al observar que cada elección genera dos subproblemas que pueden resolverse recursivamente. Además, la implementación en tabla permite evitar recalcular soluciones a subproblemas repetidos, optimizando la eficiencia.

4.1 Solución Recursiva sin Programación Dinámica

Antes de aplicar programación dinámica, el problema puede plantearse recursivamente. La función evalúa dos decisiones por materia: incluirla (si hay tiempo suficiente) o excluirla. La estructura es:

```
def f(i, t):
    if i == 0 or t == 0:
        return 0
    if horas[i-1] > t:
        return f(i-1, t)
    return max(f(i-1, t), f(i-1, t - horas[i-1]) + valor[i-1])
```

Este enfoque genera subproblemas repetidos. La programación dinámica los resuelve de forma eficiente almacenando los resultados ya calculados.

5. Documentación y Análisis

Se implementará una solución en Python que permita observar cómo cambia el rendimiento según la distribución de tiempo. Se grabará un video explicando paso a paso la codificación y la lógica detrás de la programación dinámica aplicada.

El análisis incluirá:

- Complejidad temporal: $O(n \cdot T)$
- Complejidad espacial: $O(n \cdot T)$
- Comparación contra una solución *greedy* para mostrar la ventaja del método óptimo.

Este modelo puede ampliarse incluyendo múltiples días, niveles de fatiga, o materias obligatorias.