Optimización de Recursos mediante Programación Lineal

Laura Rodríguez Lardiés

Septiembre 2023

Resumen

En este documento resolveremos un problema de optimización mediante la programación lineal. Hay también un repositorio en Github donde viene todo explicado e incluye el código completo, para visitarlo haga click **aquí**.

Índice

1. Introducción

El problema consiste en obtener el ejército más poderoso posible teniendo en cuenta que hay recursos limitados. Debido a las restricciones que tenemos, para resolver el problema es conveniente usar la programación lineal para maximizar el poder del ejército. Los datos del problema son los siguientes:

Recursos	Espadachines	Arqueros	Jinetes
Comida (1200 en total)	60	80	140
Madera (800 en total)	20	10	0
Oro (600 en total)	0	40	100
Poder	70	95	230

2. Ecuaciones del problema

Antes de comenzar a maximizar o minimizar cualquier cosa, debemos definir la función que queremos optimizar, conocida como función objetivo, y las ecuaciones que restringen a la función objetivo.

2.1. Función Objetivo

Como ya hemos visto en la introducción, queremos maximizar el poder de nuestro ejército. Además, al emplear programación lineal, la función debe ser lineal, es decir, debe tener la forma $ax + by + cz + d$. Por lo tanto, nuestra ecuación objetivo tendrá cada tipo de unidad del ejército multiplicado por la cantidad de poder que aporta al ejército. En otras palabras, las variables x, y, z serán representadas por los tipos de unidades del ejército (espadachines, arqueros y jinetes), mientras que los coeficientes que las acompañan corresponden con el valor de poder de cada unidad del ejército.

$$espadachines \times 70 + arqueros \times 95 + jinetes \times 230$$

Como podemos ver, la ecuación objetivo nos muestra el poder de nuestro ejército, que es lo que queremos maximizar.

2.2. Restricciones

Ahora definamos las restricciones. Contamos con tres recursos, por lo tanto, crearemos tres inecuaciones de restricción diferentes, que también serán lineales. Seguiremos el mismo procedimiento que para la función objetivo, sin embargo, ahora igualaremos la inecuación a la cantidad de recurso que tenemos, es decir, restringimos (acotamos por arriba).

2.2.1. Comida

Como máximo nos podemos gastar 1200 de comida, y todas las variables (unidades del ejército) gastan de este recurso.

$$espadachines \times 60 + arqueros \times 80 + jinetes \times 140 \leq 1200$$

2.2.2. Madera

Como máximo nos podemos gastar 800 de madera, pero esta vez vemos que en los datos se muestra que los jinetes no hacen uso de este recurso. Por lo tanto, el coeficiente que acompaña a esta variable (jinetes) es 0, lo que hace que la variable jinetes no se muestre en la inecuación.

$$espadachines \times 20 + arqueros \times 10 \leq 800$$

La variable de jinetes no queda reflejada en la inecuación ya que el hecho de tener jinetes en neustro ejército no afecta en absoluto en el consumo de la madera.

2.2.3. Oro

Como máximo nos podemos gastar 600 de oro y vemos que, de nuevo, hay una variable que no nos hace gastar dicho recurso, en este caso los espadachines. Por lo tanto, la variable de espadachines no se ve reflejada en la inecuación.

$$arqueros \times 40 + jinetes \times 100 \leq 600$$

Como podemos ver, esta inecuación nos dice que el hecho de tener espadachines en nuestro ejército no nos cuesta nada de dinero.

3. Código

Una vez definidas las restricciones y la función objetivo, podemos comenzar a desarrollar nuestro algoritmo de programación lineal con el objetivo de optimizar nuestro ejército. Más concretamente, lo que buscamos es maximizarlo.

El primer paso es decidir qué biblioteca usar para la programación lineal. En nuestro caso, hemos usado Google OR-Tools.

```
# Import OR-Tools para programacion lineal.
from ortools.linear_solver import pywraplp
```

OR-Tools viene con su propio solucionador de programación lineal, llamado GLOP, que es el que vamos a usar en este proyecto.

```
# Creamos un solucionador usando el backend de GLOP (Paquete de optimizacion lineal
    de Google).
solucionador = pywraplp.Solver('Maximiza el poder del ejercito', pywraplp.Solver.
    GLOP_LINEAR_PROGRAMMING)
```

El próximo paso es definir las variables del problema, que en nuestro caso son las unidades del ejército. Al ser unidades, estas variables han de ser siempre números enteros, por lo tanto emplearemos la variable IntVar de OR-Tools.

```
# Variables que queremos optimizar (variables enteras)
espadachines = solucionador.IntVar(0, solucionador.infinity(), 'espadachines')
arqueros = solucionador.IntVar(0, solucionador.infinity(), 'arqueros')
jinetes = solucionador.IntVar(0, solucionador.infinity(), 'jinetes')
```

Ahora agregamos al solucionador las restricciones que hemos definido anteriormente. Este paso es sencillo ya que existe la función Add() que directamente agrega lo que le digas al solucionador.

```
# Las restricciones de cada variable se agregan directamente a la instancia del
    solucionador.
solucionador.Add(espadachines*60 + arqueros*80 + jinetes*140 <= 1200) # Comida.
solucionador.Add(espadachines*20 + arqueros*10 <= 800)                # Madera.
solucionador.Add(arqueros*40 + jinetes*100 <= 600)                    # Oro.
```

Una vez el solucionador tenga las variables definidas y las restricciones agregadas, es hora de definir y maximizar la función objetivo.

```
# Definimos funcion objetivo : espadachines*70 + arqueros*95 + jinetes*230.
# Maximizamos dicha funcion.
solucionador.Maximize(espadachines*70 + arqueros*95 + jinetes*230)
```

Una vez teniendo todo en el solucionador, podemos comenzar a optimizar. Simplemente usamos la función Solve() que nos indica un estado, mediante el cual podemos conocer si la solución obtenida mediante nuestro algoritmo es la solución óptima.

```
estado = solucionador.Solve # Resolvemos el problema.
```

Nuestro último paso es mostrar en pantalla el resultado en caso de que este sea óptimo. De lo contrario, imprimiremos un mensaje que haga saber al usuario que la solución obtenida no es la óptima.

```
# Vemos si el estado de la solucion es optimo.
    if estado == pywraplp.Solver.OPTIMAL:
        print('================= Solucion =================')
        print(f'El problema se ha resuelto en {solucionador.wall_time():.2f}
    milisegundos y en {solucionador.iterations()} iteraciones.')
        print()
        print(f'Valor del poder optimo = {solucionador.Objective().Value()} Poder')
        print('Ejercito:')
        print(f' - Espadachines = {espadachines.solution_value()}')
```

```
        print(f' - Arqueros = {arqueros.solution_value()}')
        print(f' - Jinetes = {jinetes.solution_value()}')

    else: # En el caso en el que el programa no haya encontrado una solucion optima.
        print('El solucionador no ha podido encontrar una solucion optima.')
```

Al ejecutar el código, se nos muestra en pantalla el resultado de la optimización

```
================= Solucion =================
El problema se ha resuelto en 10.00 milisegundos y en 2 iteraciones.

Valor del poder optimo = 1800.0 Poder
Ejercito:
 - Espadachines = 6.0000000000000036
 - Arqueros = 0.0
 - Jinetes = 5.999999999999999
```

4. Conclusiones

Analizando los resultados, podemos ver que nuestro algoritmo ha apostado por gastar todo el dinero en conseguir jinetes ya que dan 230 de poder, que tine una diferencia de 135 de poder con los arqueros, la segunda unidad más poderosa del ejército. Sin embargo, los arqueros y los espadachines tan solo tienen una diferencia de 25 de poder, que comparada con la diferencia entre arqueros y jinetes, es muy pequeña. Por esta razón nuestro algoritmo ha gastado todo nuestro oro en jinetes (6 en total). Una vez obtenidos todos los jinetes posibles, vemos cuales son los recursos que nos dejan (360 comida, 800 madera, 0 oro). Con estos recursos sólo nos da para otros 6 espadachines, lo que significa que nos sobran 680 unidades de madera.

Viendo este análisis del resultado, podemos decir que la mejor unidad es la del jinete, ya que a pesar de costar mucho oro, nos aporta mucho poder y no gasta en madera. Por otro lado, la peor unidad es la de los arqueros, ya que la diferencia de poder con los espadachines no es muy grande y además, gastan más recursos que los espadachines.

Considero que hemos acertado con este algoritmo ya que otros métodos, como el de 'adivinar y comprobar' habría llevado muchos intentos hasta encontrar con la solución óptima. Hay tantas posibles soluciones, que dar con la solución óptima al primer intento adivinando es prácticamente imposible, lo que también hace que el tiempo de optimización sea muy elevado. Sin embargo, mediante la programación lineal, tan sólo nos ha llevado 2 iteraciones y 10 milisegundos encontrar la solución óptima, cosa que sería muy difícil de conseguir usando el método de 'adivinar y comprobar'. Por otro lado, el problema al que se enfrentan los algoritmos de aprendizaje automático es que por mucho que los entrenes, puede que la máquina piense que ya ha optimizado el problema, sin embargo la realidad es que se ha quedado atascado en un máximo o mínimo local, cuando lo que busamos es que sea global. Además, para que la máquina aprenda, tiene que cometer errores primero, lo que significa que el número de iteraciones hasta resolver el problema de optimización será muy elevado.