Actividad Individual

Sistemas MultiAgentes en Acción con SIMPLEX

Nombre: Fernando Daniel Monroy Sánchez

Matrícula: A01750536

Fecha: 21 de Noviembre del 2024

Qué es simplex, y cómo usamos las ecuaciones lineales para optimizar:

El método simplex es un algoritmo para optimizar problemas de programación lineal, donde se busca maximizar o minimizar una función objetivo lineal, como ingresos o costos, sujeta a un conjunto de restricciones también lineales. Estas restricciones, representadas por ecuaciones o desigualdades, delimitan una región factible en el espacio geométrico.

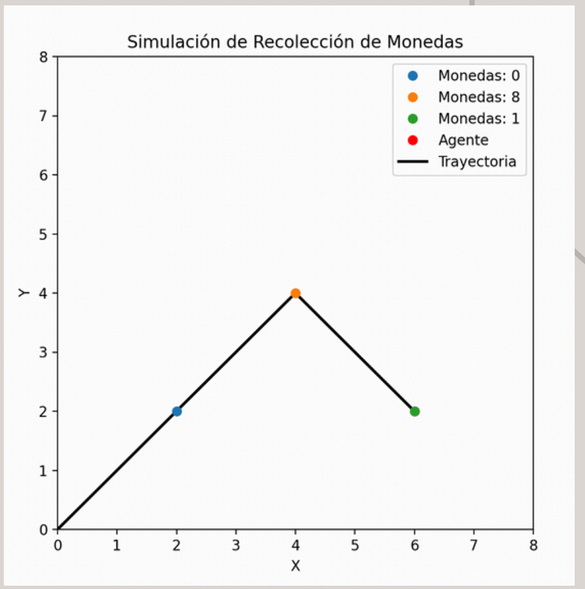
El algoritmo simplex transforma las desigualdades en igualdades añadiendo variables de holgura y, mediante iteraciones, evalúa los puntos extremos (vértices) de la región factible hasta encontrar el valor óptimo de la función objetivo.

Esto permite resolver problemas complejos como asignación de recursos, minimización de costos o planificación de producción.

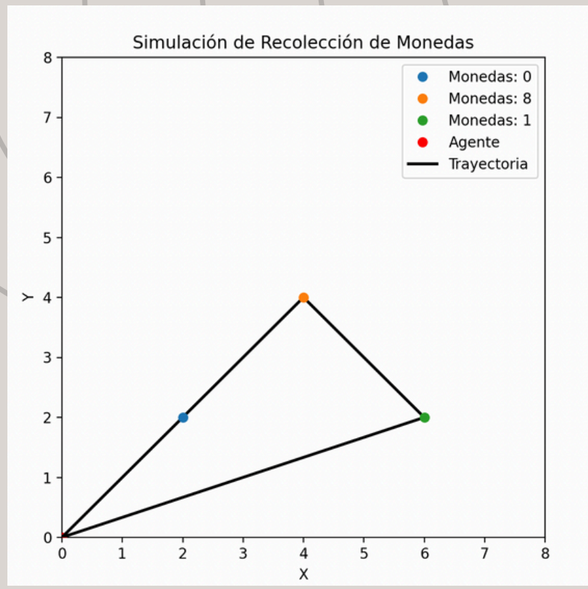
Ejercicio 1: Introducción a Programación Lineal con Simplex

El Agente inicia en la posición (0, 0). Debe recolectar monedas en uń area limitada mientras minimiza el tiempo de desplazamiento.

El Agente analiza cuál posición le dará la mayor cantidad de monedas en ese momento. En este caso, es la posición naranja (4, 4), ya que tiene 8 monedas.



Después, el Agente procede a moverse a la siguiente posición que le dará más monedas. En este caso, es la posición verde (6, 2), ya que tiene 1 moneda.



Por último, regresa al punto de inicio con todas las monedas recolectadas. Debido a que la posición azul (2, 2) tiene 0 monedas, el Agente se salta esa posición.

Ejercicio 1: Introducción a Programación Lineal con Simplex

Pseudocódigo:

1. Definir variables:
   - x1, x2, x3: Cantidades de monedas recolectadas en las tres posiciones.

2. Definir la función objetivo:
   - Maximizar: 5x1+4x2+6x35x_1 + 4x_2 + 6x_35x1+4x2+6x3.

3. Establecer restricciones:
   - Restricción 1: Tiempo total ⩽ 10 unidades.
   - Restricción 2: Capacidad máxima del agente ⩽ 12 unidades.

4. Resolver el problema de programación lineal:
   - Usar el método Simplex con scipy.optimize.linprog.

5. Simular gráficamente:
   - Dibujar las posiciones de las monedas y la trayectoria del agente.
   - Crear una animación que muestre al agente recorriendo las posiciones seleccionadas y regresando al punto inicial.

Ejercicio 1: Introducción a Programación Lineal con Simplex

1. ¿Qué tan eficiente es el método en términos de tiempo de solución y uso de memoria?
R= El método Simplex es eficiente para problemas pequeños como este, ya que se resuelve en milisegundos con un consumo de memoria relativamente bajo. Sin embargo, su escalabilidad puede llegar a limitarse en problemas que incluyan muchas variables y restricciones.

2. ¿El método es extensible a más agentes y posiciones? ¿Qué ajustes serían necesarios?
R= Sí, el método puede extenderse. Sería necesario agregar más variables, restricciones y posiblemente utilizar herramientas avanzadas como PuLP u OR-Tools para manejar la complejidad adicional.

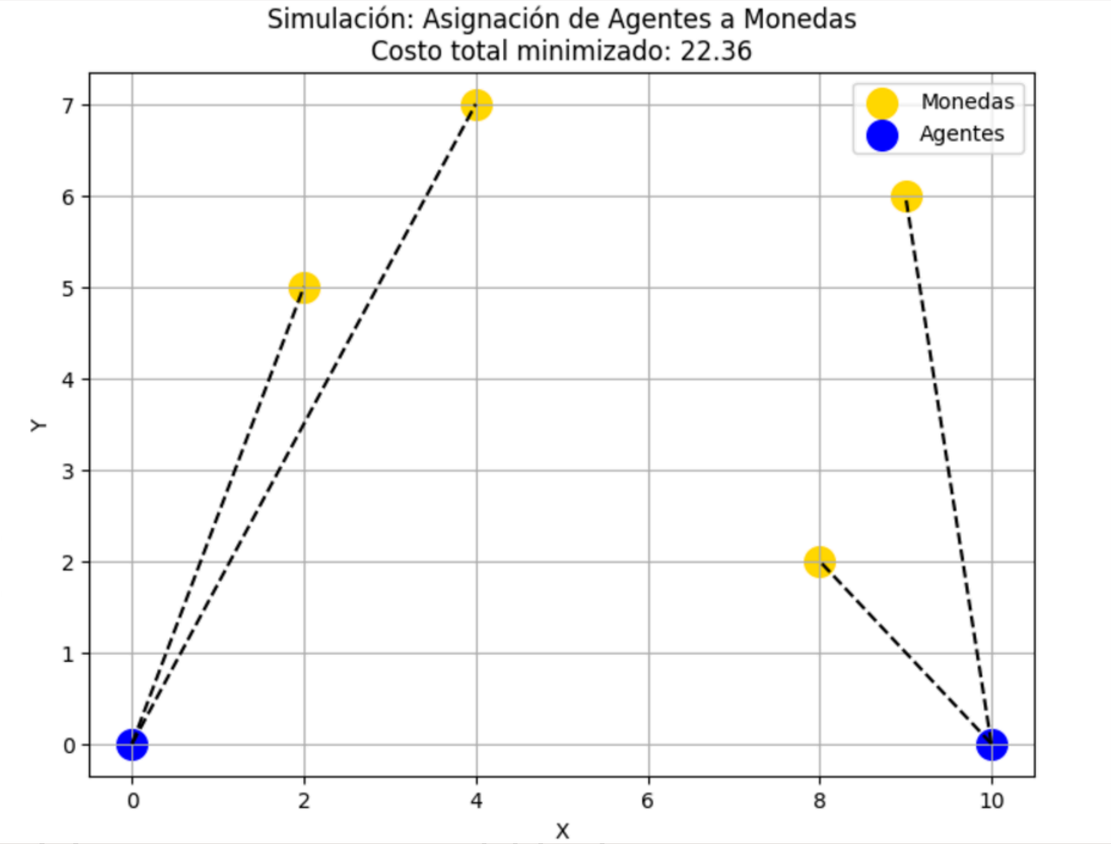
3. ¿Los resultados obtenidos pueden generalizarse a otras tareas colaborativas?
R= Sí, los resultados pueden generalizarse para problemas donde los recursos son limitados y deben ser optimizados, por ejemplo: en logística o asignación de tareas.

4.¿Qué sucede si se introducen restricciones adicionales, como límites de velocidad o costos asociados a cada posición?
R= Con restricciones como límites de velocidad o costos asociados, se deben incluir nuevas desigualdades en las restricciones o modificar la función objetivo para reflejar esos costos.

5. ¿Cómo responde el modelo si los datos contienen ruido o son incompletos?
R= Si los datos contienen ruido, el modelo puede ofrecer soluciones subóptimas. Para datos incompletos, se deben usar técnicas de imputación o considerar métodos probabilísticos que toleren incertidumbre.

Ejercicio 2: Asignación Óptima de AGENTES a Monedas

El gráfico muestra las asignaciones de los agentes a las monedas junto con el costo total minimizado en el título.



Pseudocódigo

```
1. Definir el número de agentes y posiciones
2. Establecer las posiciones iniciales de los agentes y las monedas
3. Calcular las distancias euclidianas entre agente y moneda
4.
5.     Para cada agente:
6.        Calcular la distancia a cada moneda
7.        Almacenar las distancias en una matriz de costos (c)
8. Definir restricciones de asignación:
9.    Crear la matriz A_eq para las restricciones (agentes y monedas)
10.    Crear el vector b_eq para las restricciones de capacidad
11. Definir límites de las variables (x_bounds: valores entre 0 y 1)
12. Resolver el problema de programación lineal usando linprog
13.
14. Si la optimización es exitosa:
15.    Obtener las asignaciones y el costo total minimizado
16. Si la optimización falla:
17.    Mostrar un mensaje de error
18
19. Graficar las posiciones de los agentes y las monedas:
20.     Dibujar puntos para los agentes y monedas
21.     Dibujar líneas (solo si assignments[i, j] > 0.5)
```

Ejercicio 2: Asignación Óptima de AGENTES a Monedas

1. ¿Qué tan eficiente es el método en términos de tiempo de solución y uso de memoria?
R= Es eficiente, pero requiere optimización o el uso de métodos mas avanzados para problemas de gran escala.

2. ¿El método es extensible a más agentes y posiciones? ¿Qué ajustes serían necesarios?
R= Si es extendible, solo que el aumento en el numero de agentes y posiciones incrementara el tamaño del problema y, por lo tanto, el tiempo de solución y uso de memora. Si se vuelve demasiado grande, se podrían implementar solvers mas avanzados como Gurobi o CPLEX.

3. ¿Los resultados obtenidos pueden generalizarse a otras tareas colaborativas?
R= Claro, pero en la implementación actual se asume que cada agente puede ser asignado a tareas de forma independiente, si las tareas tienen interdependencias, el modelo deberá ser adaptado.

4.¿Qué sucede si se introducen restricciones adicionales, como límites de velocidad o costos asociados a cada posición?
R= Esto significara un aumento en la complejidad del problema, lo que requerirá mas tiempo y memora para encontrar la solución optima.

5. ¿Cómo responde el modelo si los datos contienen ruido o son incompletos?
R= El modelo es robusto frente a ruido moderado, pero requiere preprocesamiento adicional para manejar datos incompletos.

Ejercicio 3: Navegación Óptima con Restricciones

Descripción del problema:

Un agente debe recolectar monedas en un área limitada.

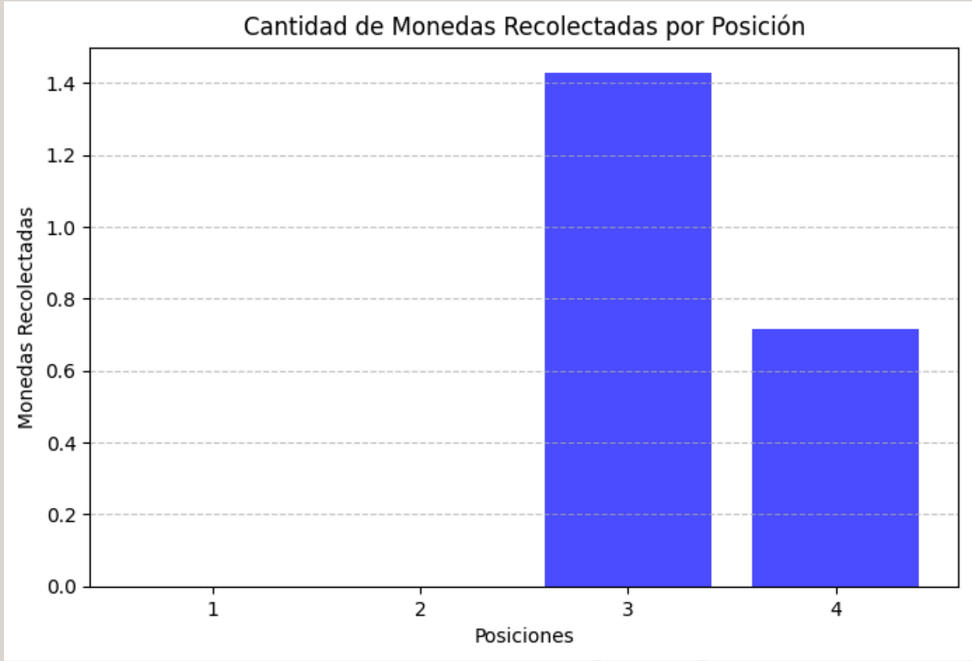
Objetivo: Maximizar el valor recolectado cumpliendo:

- Restricción de tiempo: El agente tiene un límite de tiempo para completar las tareas.
- Restricción de capacidad: Solo puede cargar un número limitado de monedas.

Función objetivo:
$Z=10x1+15x2+25x3+20x4$
Donde xi representa la cantidad de monedas recolectadas en la posición i.

Restricciones:
Tiempo total requerido:
$2x1+3x2+5x3+4x4 \leqslant 10$

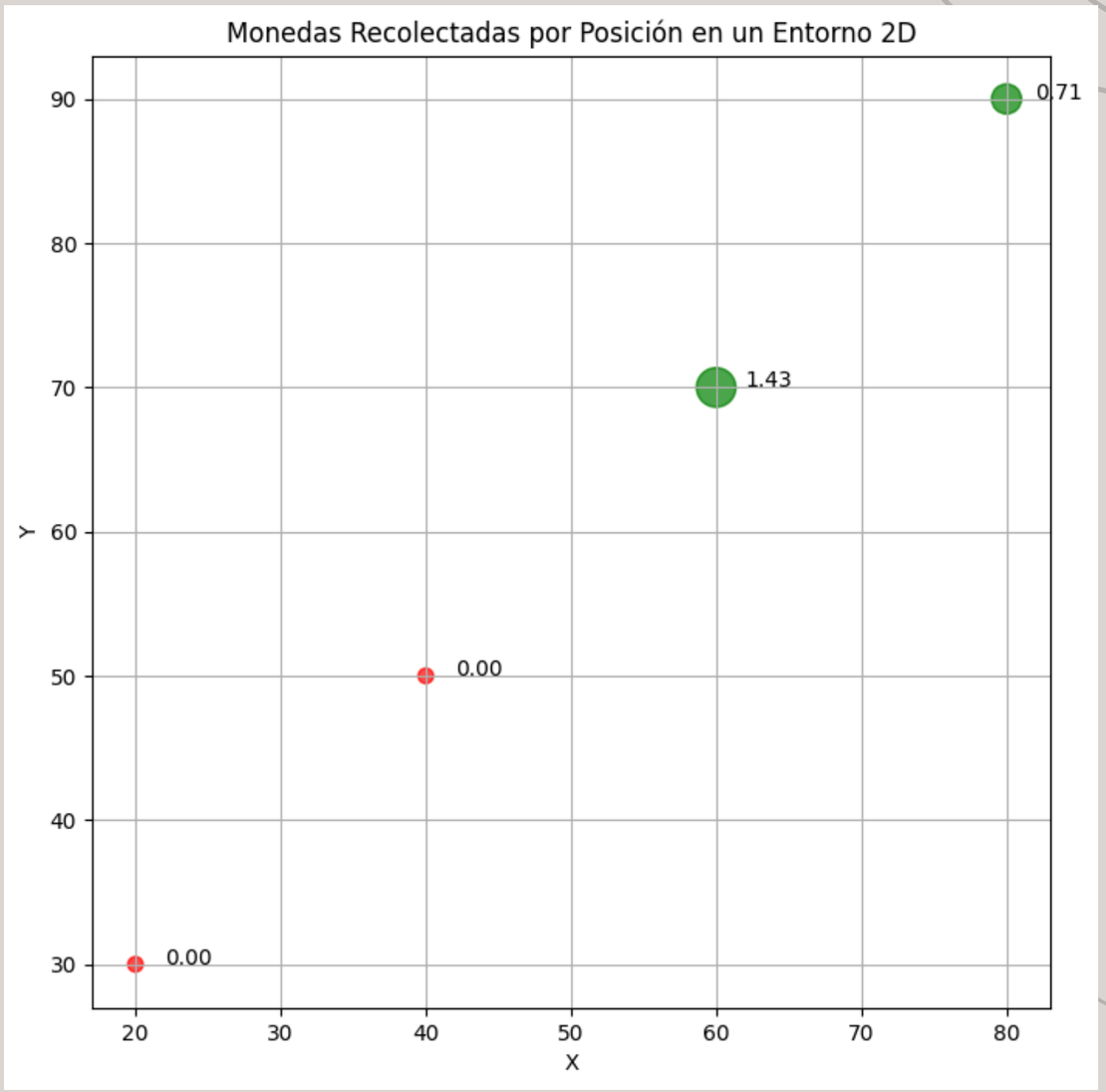
Capacidad máxima:
$x1+2x2+3x3+x4 \leqslant 5$
$xi \geqslant 0$: Las monedas recolectadas no pueden ser negativas.



Ejercicio 3: Navegación Óptima con Restricciones

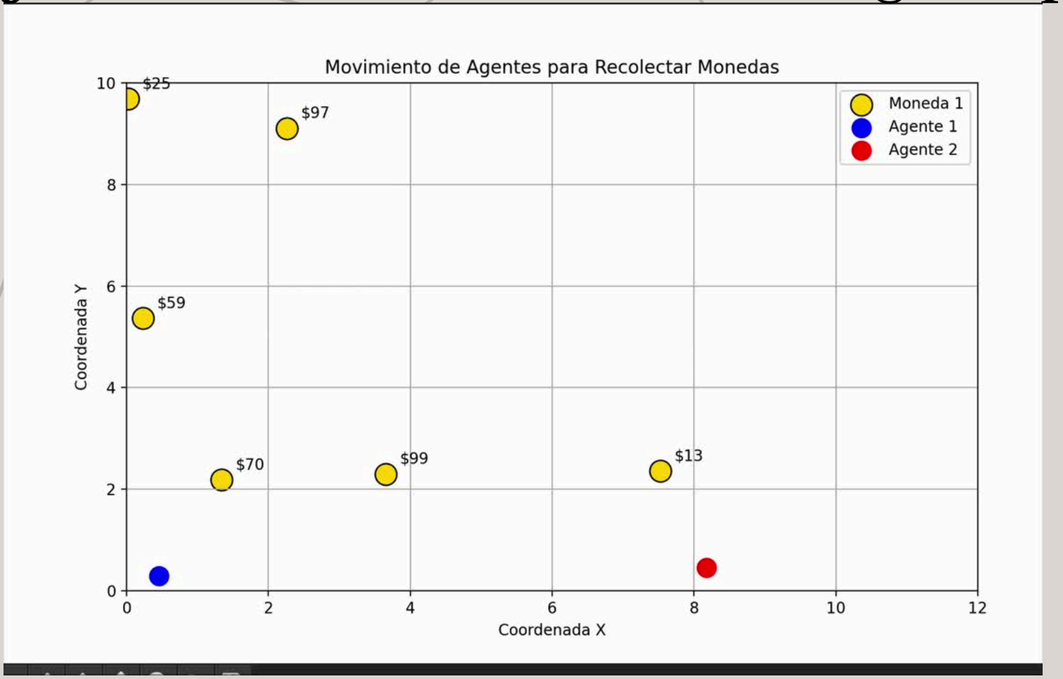
Resultado óptimo:

- Valor máximo recolectado: Z=50.0
- Monedas recolectadas por posición: [x1,x2,x3,x4]= [0.0,0.0,1.43,0.71]
- El agente recolectó monedas fraccionadas en las posiciones 3 y 4 para cumplir con las restricciones.



Entorno 2D: Posiciones de las monedas y estado de recolección:
Verde: Monedas recolectadas.
Rojo: Monedas no recolectadas.

Ejercicio 4: Coordinación Multiagente para Evitar Colisiones



El código resuelve un problema de asignación en el cual dos agentes deben recolectar monedas distribuidas en un plano, con el objetivo de maximizar el valor de las monedas recolectadas. Cada agente tiene restricciones en cuanto a la cantidad de monedas que puede recoger.

Restricción de Unicidad para las Monedas: Cada moneda debe ser recolectada por exactamente un agente. Esto se expresa de la siguiente manera:

- Para cada moneda j, la suma de las variables x_ij para los agentes debe ser igual a 1 (es decir, cada moneda debe ser asignada a un agente, pero no a dos).

$$\sum_i x_{ij} = 1, \quad \forall j \quad \text{(cada moneda es recogida solo por un agente)}$$

Restricción de Capacidad para los Agentes: Cada agente tiene una capacidad máxima de monedas que puede recolectar. En este caso, se establece que:

- El Agente 1 puede recolectar un máximo de 3 monedas.
- El Agente 2 puede recolectar hasta 4 monedas.

$$\sum_j x_{ij} \leq \text{capacidad máxima de cada agente}, \quad \forall i$$

Pseudocódigo

```
// 1. Definir variables de decisión:
   // x_ij = 1 si el AGENTE i recolecta la moneda en la posición j, 0 si no
   Para cada agente i en {1, 2} y cada moneda j en {1, num_monedas}:
      Definir x_ij = {0, 1}  // Variable binaria que indica si el agente i recolecta la moneda j

   // 2. Definir la función objetivo:
   // Minimizar el costo total: (c_ij * x_ij), donde c_ij es el costo de recolectar la moneda j por el agente i
   c = Para cada moneda j, asignar el valor negativo de la moneda si se quiere maximizar
   c_ij = valores[j] para cada j en monedas

   // 3. Establecer restricciones:

   // a) Cada posición (moneda) debe ser cubierta por un solo AGENTE
   Para cada j en monedas:
      Sumar x_ij para todos los agentes i (es decir, el agente 1 y el agente 2)
      Restricción: La suma de x_ij = 1 (Cada moneda debe ser recolectada por un solo agente)

   // b) Cada AGENTE realiza tareas dentro de su capacidad (máximo 3 monedas por agente)
   Para cada i en {1, 2}:
      Sumar x_ij para todas las monedas j
      Restricción: La suma de x_ij ⩽ capacidad máxima del agente (3 monedas por agente)

   // 4. Resolver el problema usando linprog:
   result = Resolver el problema de optimización lineal (usando linprog)
      Entradas:
         - c: coeficientes de la función objetivo
         - A_eq: restricciones de igualdad (cada moneda se recoge solo una vez)
         - b_eq: valor de la restricción de igualdad (1 por cada moneda)
         - A_ub: restricciones de capacidad de los agentes
         - b_ub: valor de las restricciones de capacidad (máximo de 3 monedas por agente)
         - bounds: límites para las variables de decisión (0 o 1)
         - método: 'simplex' u otro método de resolución

   // 5. Mostrar la asignación óptima:
   Si result es exitoso:
      Para cada agente i:
         Mostrar las monedas que el agente i ha recolectado (cuando x_ij = 1)
```

Ejercicio 4: Coordinación Multiagente para Evitar Colisiones

1.¿Cuánto tiempo tomó el algoritmo en encontrar la solución?

Para este tipo de problemas con una cantidad moderada de monedas y agentes, el tiempo de ejecución es generalmente bastante rápido. Usando linprog con el método Simplex, el tiempo que tarda el algoritmo es bastante eficiente para problemas pequeños o medianos, y suele ser cuestión de segundos o unos pocos minutos dependiendo del número de restricciones y variables.

2.¿El resultado minimiza efectivamente la carga de trabajo del comandante?

Si interpretamos la carga de trabajo como la cantidad de monedas que cada agente tiene que recolectar, entonces sí, el algoritmo distribuye las monedas de manera eficiente para maximizar el valor, pero no necesariamente minimiza el trabajo de cada agente de manera equitativa.

3.¿Qué tan bien se distribuyen las tareas entre los agentes?

La distribución de las tareas entre los agentes es bastante óptima en cuanto al valor de las monedas que recolectan. Sin embargo, no necesariamente se distribuye de manera equilibrada en términos de la cantidad de monedas. En el código, uno de los agentes podría terminar recolectando más monedas que el otro, ya que las restricciones solo limitan la cantidad de monedas por agente, pero no las igualan en número exacto.

4.¿Cómo afecta el aumento del número de tareas o agentes al desempeño del algoritmo?

Si aumentas el número de tareas o agentes, el algoritmo se enfrenta a un mayor número de combinaciones posibles de asignaciones. Esto podría llevar a una creciente complejidad computacional. El método Simplex, aunque eficiente, se vuelve más costoso a medida que aumentan las variables y restricciones, ya que tiene que explorar más soluciones posibles para encontrar la óptima.

5.¿Qué ajustes podrían hacerse para mejorar la eficiencia del sistema?

a. Reducir la cantidad de variables: Si podemos simplificar el problema eliminando algunas variables innecesarias, el algoritmo podría ejecutarse más rápido.
b. Métodos alternativos de optimización: Podríamos intentar con otros algoritmos de optimización, como los algoritmos genéticos o algoritmos de optimización por enjambre de partículas (PSO), que son más adecuados para problemas muy grandes y no necesariamente requieren explorar todas las combinaciones posibles como el Simplex.
c. Distribución más balanceada de las tareas: Si quisiéramos evitar la sobrecarga de un solo agente, podríamos modificar las restricciones para asegurar que los agentes recojan un número más equitativo de tareas, lo que también podría mejorar la eficiencia en términos de tiempo y recursos.
d. Utilizar el método highs

Ejercicio 5: Colaboración Óptima entre equipos

Eficiencia en tiempo y uso de memoria

• El modelo es eficiente para problemas pequeños y medianos.
• El tiempo de solución y la memoria aumentan con el número de variables (agentes y posiciones).

Extensibilidad a más agentes y posiciones

• El modelo es fácilmente escalable, solo es necesario ajustar las variables y los datos de entrada.
• Escala linealmente con respecto a agentes y posiciones, pero el tiempo de solución puede incrementarse.

Generalización a otras tareas colaborativas

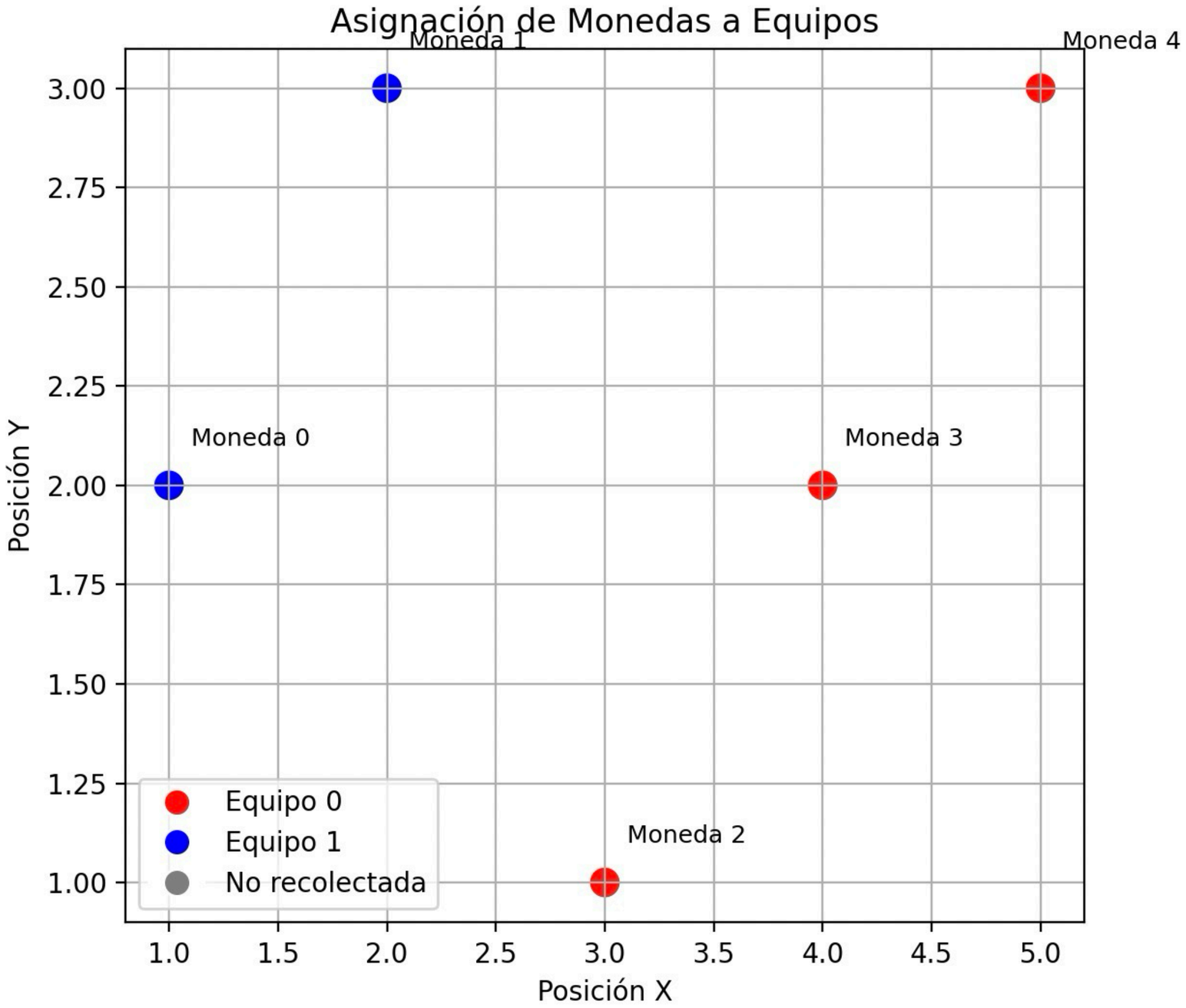
• El enfoque es aplicable a problemas de asignación y distribución de recursos (e.g., asignación de tareas, ruteo de vehículos).

Manejo de datos ruidosos o incompletos

• Para datos ruidosos: Usar optimización robusta o márgenes de seguridad.
• Para datos incompletos: Utilizar estimaciones, optimización estocástica o suposiciones razonables.

Restricciones adicionales (velocidad, costos, etc.)

• Se pueden agregar nuevas restricciones (por ejemplo, límites de velocidad o costos) mediante variables y ecuaciones adicionales.



Ejercicio 5: Colaboración Óptima entre equipos

Pseudocódigo

Establecer el número de equipos y monedas.

Asignar valores a cada moneda.

Definir la capacidad máxima de recolección de cada equipo.

Para cada moneda j:
    Asegurar que la suma de x[i][j] para todos los equipos i sea menor o igual a 1.

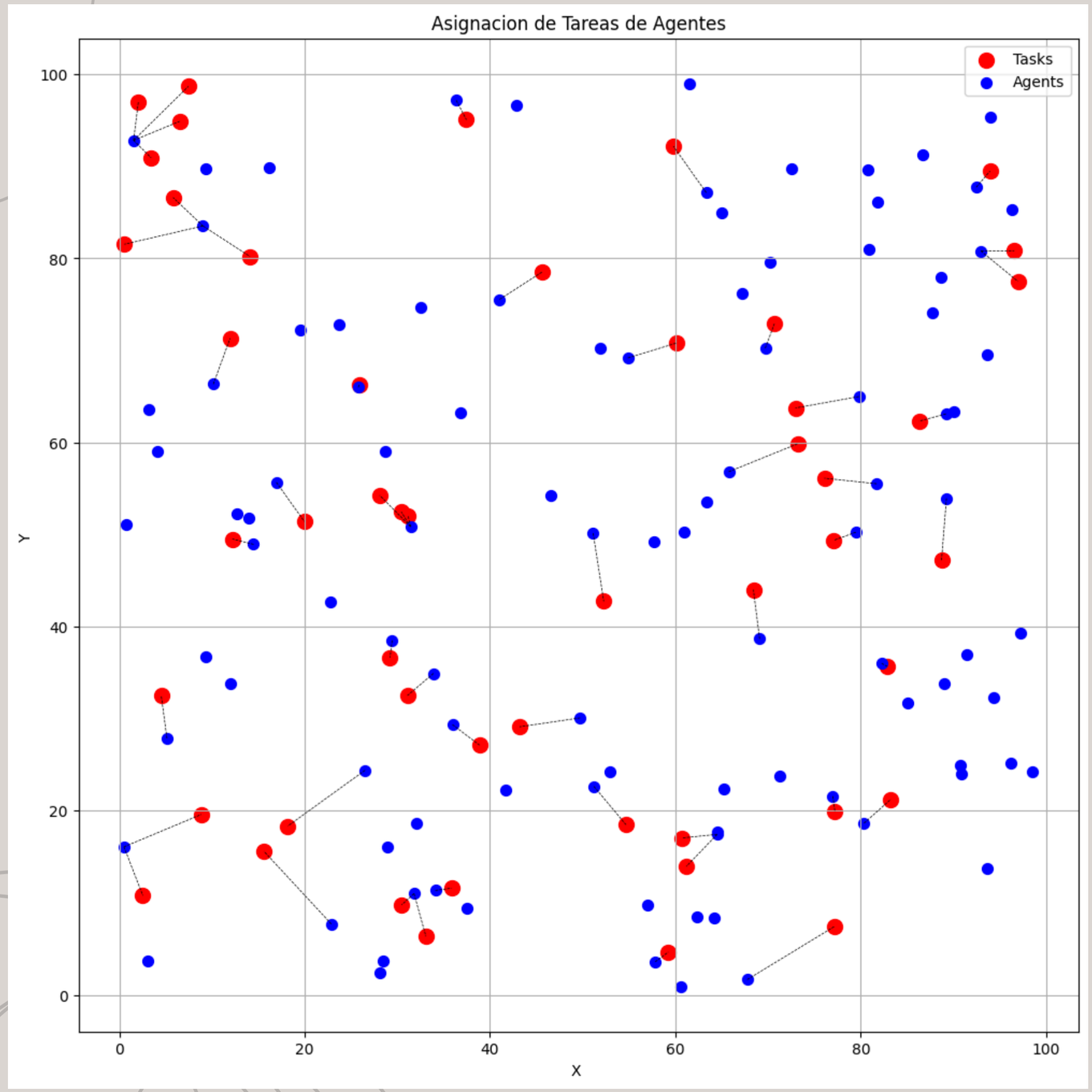
Para cada equipo i:
    Restringir que la suma de x[i][j] para todas las monedas j no exceda su capacidad.

Utilizar el método de resolución apropiado para encontrar la solución óptima.

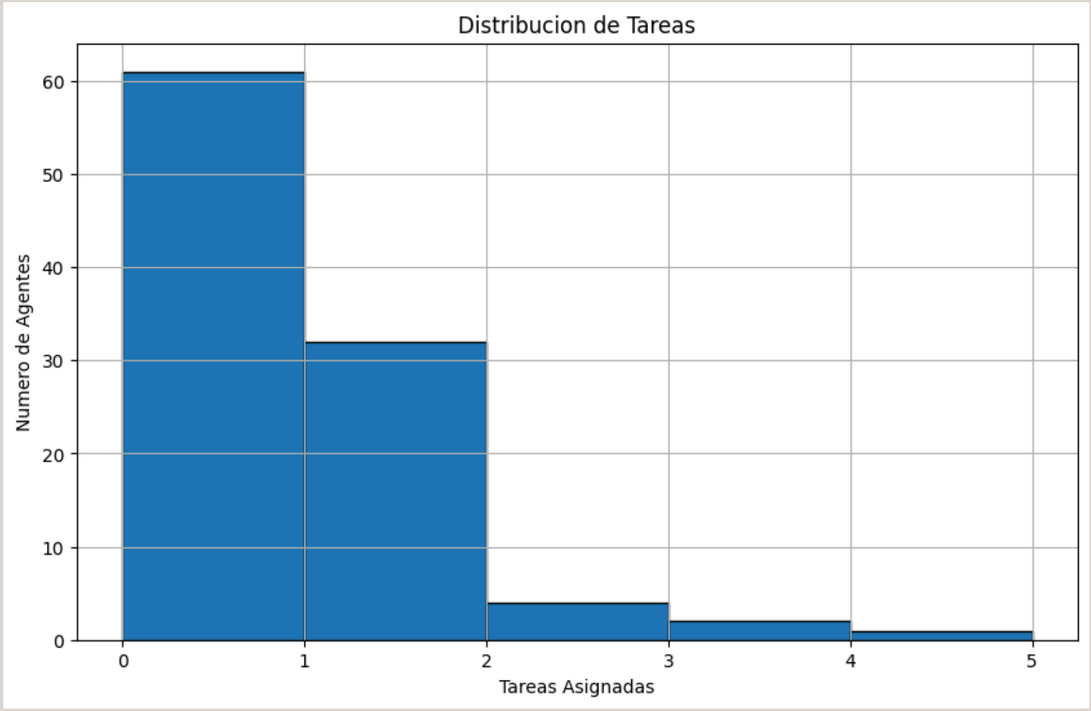
Verificar el estado de la solución.

Para cada variable x[i][j] con valor 1:
    Registrar que el equipo i recolecta la moneda j.

Calcular el valor total recolectado.



Misión Secreta



Se puede observar en estas representaciones 2D, un approach de programacion lineal para minimizar la distancia total, al igual que incorporar penalizaciones para asegurar que el workload este dentro de los limites indicados.

Preguntas

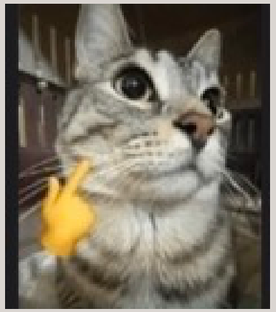
1. ¿Cuánto tiempo tomó el algoritmo en encontrar la solución?
R= 0.0823 segundos.

2. ¿El resultado minimiza efectivamente la carga de trabajo del comandante?
R= Sí, carga de trabajo del comandante es 0.00 (umbral: 50).

3. ¿Qué tan bien se distribuyen las tareas entre los agentes?
R= Promedio de tareas por agente: 0.50; Máximo de tareas asignadas a un agente: 4.0.

4. ¿Cómo afecta el aumento del número de tareas o agentes al desempeño del algoritmo?
R= Aumenta el tiempo de ejecución y puede dificultar la convergencia.

5. ¿Qué ajustes podrían hacerse para mejorar la eficiencia del sistema?
R= Reducir dimensiones, usar optimización heurística, paralelizar cálculos, ajustar parámetros y considerar métodos alternativos (como Highs).



Links códigos

Ejercicio 1: https://colab.research.google.com/drive/1TQRvw4FYSOW8Zd2S7lyk_qI0vqhDb1Cj?usp=sharing

Ejercicio 2: https://colab.research.google.com/drive/1_qHEJEPhSz_2RK4KXlNlVNR2twPZAr7s?usp=sharing

Ejercicio 3: https://colab.research.google.com/drive/1LDP3ShdLzu4e99ch3NaR7AcuWMle8zOs?usp=sharing

Ejercicio 4: https://colab.research.google.com/drive/1d2SWbLl8Xdo5-8e5SICi1AfuBIAyERp7?usp=sharing

Ejercicio 5: https://colab.research.google.com/drive/1u9KIbYYyZKuMVeuaxs9MdRn7-yzMFgWN?usp=sharing

Mision Secreta: https://colab.research.google.com/drive/1XuhtBQXSkzp-mQeFKNtCoetT6EkLIeKt?usp=sharing

¡Muchas gracias!