Reconocido Simulado

Equipo:

Felix Yahveh Alanis Candelaria	1947480
Juan Carlos Diaz Gonzalez	1963196
Johann Joseph Velazquez Antonio	1962111

Instrucciones

Programar una de la siguientes metaheurísticas para el problema de ruta más corta o problema de la mochila:

- Algoritmos Genéticos
- GRASP
- Recocido Simulado* (Esta metaheurística no la vimos en clase pero es más sencilla que las anteriores)

Parte 1 (10 puntos)

Programar la metaheurística

Parte 2 Extra (5 puntos)

Hacer una comparativa con algún algoritmo informado o no informado de las tareas anteriores.

Indicaciones Generales (Aplican a casi todas sus tareas)

- Necesito ver gráficas de los resultados de su experimentación (boxplot, barras, etc.) y una pequeña conclusión en base a los resultados que obtuvieron, así como la descripción de las características del equipo donde corrieron los experimentos y las instancias que utilizaron.
- Necesito que agreguen la liga al repositorio con el código que usaron.
- No peguen código en su tarea , en su lugar, escríbanlo como pseudocódigo.
- Si se deciden por el problema de ruta más corta utilicen las instancias del problema de ruta más corta que crearon en la tarea 2 y obtengan el valor óptimo con su algoritmo Dijkstra que también utilizaron en la tarea 2.

- Si eligen el problema de la mochila utilicen las instancias del problema de la mochila.
- La tarea es en equipo, **máximo 3 personas**, en el reporte indiquen por favor los nombres de los integrantes, no es obligatorio usar las herramientas como overleaf o Seaborn, etc. pueden hacer el reporte en Word y las gráficas en Excel o lo que les parezca adecuado, siempre y cuando al final exporten su reporte a un PDF e incluya lo que les pido en la tarea. El código de la tarea debe de ser en Python.

Introduccion

El algoritmo del reconocimiento simulado (Simulated Annealing) se puede describir en los siguientes pasos:

- Inicializar la solución: Comenzar con una solución aleatoria para el problema de optimización.
- Establecer los parámetros: Establecer los parámetros del algoritmo, incluyendo la temperatura inicial, la función de enfriamiento y el número de iteraciones.
- Bucle principal: Repetir el siguiente proceso para un número predeterminado de iteraciones o hasta que se alcance un criterio de parada.
- **Generar una solución vecina**: Generar una solución vecina a partir de la solución actual cambiando algunos de sus componentes de manera aleatoria.
- **Evaluar la solución vecina**: Evaluar la función de costo de la nueva solución.
- **Aceptar o rechazar la solución**: Aceptar la nueva solución si su función de costo es menor que la de la solución actual. Si la función de costo es mayor, se debe calcular la probabilidad de aceptar la nueva solución, que depende de la temperatura y de la diferencia de costos entre la solución actual y la nueva solución.
- **Actualizar la temperatura**: Actualizar la temperatura utilizando una función de enfriamiento.
- **Devolver la mejor solución encontrada**: Al final del bucle principal, se devuelve la mejor solución encontrada durante todas las iteraciones.
- **Inicialización**: Se comienza con una solución aleatoria, que representa una posible combinación de objetos que se pueden colocar en la mochila.
- **Evaluación**: Se calcula el valor total de la solución actual y se compara con el mejor valor obtenido hasta el momento.

- **Generación de vecinos**: Para generar una solución vecina, se puede realizar una de las siguientes operaciones:

- **Agregar un objeto aleatorio que aún no está en la mochila**. Eliminar un objeto aleatorio de la mochila. Intercambiar un objeto aleatorio de la mochila por otro objeto que aún no está en la mochila. Evaluación de vecinos: Se calcula el valor de la solución vecina y se compara con el valor de la solución actual.

- **Aceptación o rechazo de vecinos**: Si el valor de la solución vecina es mayor que el valor de la solución actual, se acepta la solución vecina como la nueva solución actual. Si el valor de la solución vecina es menor que el valor de la solución actual, se calcula la probabilidad de aceptar la solución vecina. La probabilidad de aceptación depende de la temperatura actual y de la diferencia entre los valores de las soluciones.

- **Enfriamiento**: Se disminuye la temperatura actual utilizando una función de enfriamiento.

- **Criterio de parada:** Se repiten los pasos 3-6 hasta que se alcance un criterio de parada, como un número máximo de iteraciones o un tiempo límite.

- **Retorno:** Se devuelve la mejor solución encontrada.

En resumen, el algoritmo del reconocimiento simulado para resolver el problema de la mochila consiste en generar soluciones aleatorias, evaluarlas y generar soluciones vecinas mediante operaciones específicas. Luego, se aceptan o rechazan las soluciones vecinas en función de una probabilidad calculada utilizando la temperatura actual y se disminuye la temperatura gradualmente. El algoritmo se repite hasta que se alcance un criterio de parada, y se devuelve la mejor solución encontrada.

```
Algoritmo reconocimiento Simulado
Inicio

importar random
importar math

# Definir los objetos (peso, valor)
objetos = [(269, 10), (95, 55), (4, 10), (60, 47), (32, 5), (23, 4), (72, 50), (80, 8), (62, 61),

# Definir la capacidad de la mochila, la temperatura inicial y el enfriamiento
capacidad_mochila = 300
temperatura_inicial = 100
enfriamiento = 0.999

# Función para calcular el valor total de una solución
función calcular_valor(solucion, objetos)
    valor = 0
    Para i en rango de longitud(solucion)
        Si solucion[i] es igual a 1
```

```
            valor = valor + objetos[i][1]
    Retornar valor

# Función para calcular el peso total de una solución
función calcular_peso(solucion, objetos)
    peso = 0
    Para i en rango de longitud(solucion)
        Si solucion[i] es igual a 1
            peso = peso + objetos[i][0]
    Retornar peso

# Función para generar una solución vecina
función generar_vecino(solucion)
    vecino = lista(solucion)
    i = aleatorio entre 0 y longitud(vecino) - 1
    Si vecino[i] es igual a 0
        vecino[i] = 1
    Sino
        vecino[i] = 0
    Retornar vecino

# Función para aplicar el algoritmo de reconocimiento simulado
función simulated_annealing(objetos, capacidad_mochila, temperatura_inicial, enfriamiento)
    # Inicialización
    solucion_actual = [aleatorio entre 0 y 1 para _ en rango de longitud(objetos)]
    valor_actual = calcular_valor(solucion_actual, objetos)
    mejor_solucion = lista(solucion_actual)
    mejor_valor = valor_actual

    # Bucle principal
    temperatura = temperatura_inicial
    Mientras temperatura sea mayor que 1
        Para i en rango de 100
            # Generar una solución vecina
            vecino = generar_vecino(solucion_actual)

            # Evaluar la solución vecina
            valor_vecino = calcular_valor(vecino, objetos)
            peso_vecino = calcular_peso(vecino, objetos)

            # Aceptar o rechazar la solución vecina
            Si peso_vecino es menor o igual a capacidad_mochila y (valor_vecino es mayor que valo
                solucion_actual = lista(vecino)
                valor_actual = valor_vecino

            # Actualizar la mejor solución encontrada
            Si valor_actual es mayor que mejor_valor y peso_vecino es menor o igual a capacidad_m
                mejor_solucion = lista(solucion_actual)
                mejor_valor = valor_actual
```

```
        # Enfriamiento
        temperatura = temperatura * enfriamiento

    Retornar mejor

Fin Algoritmo
```

Parte 2

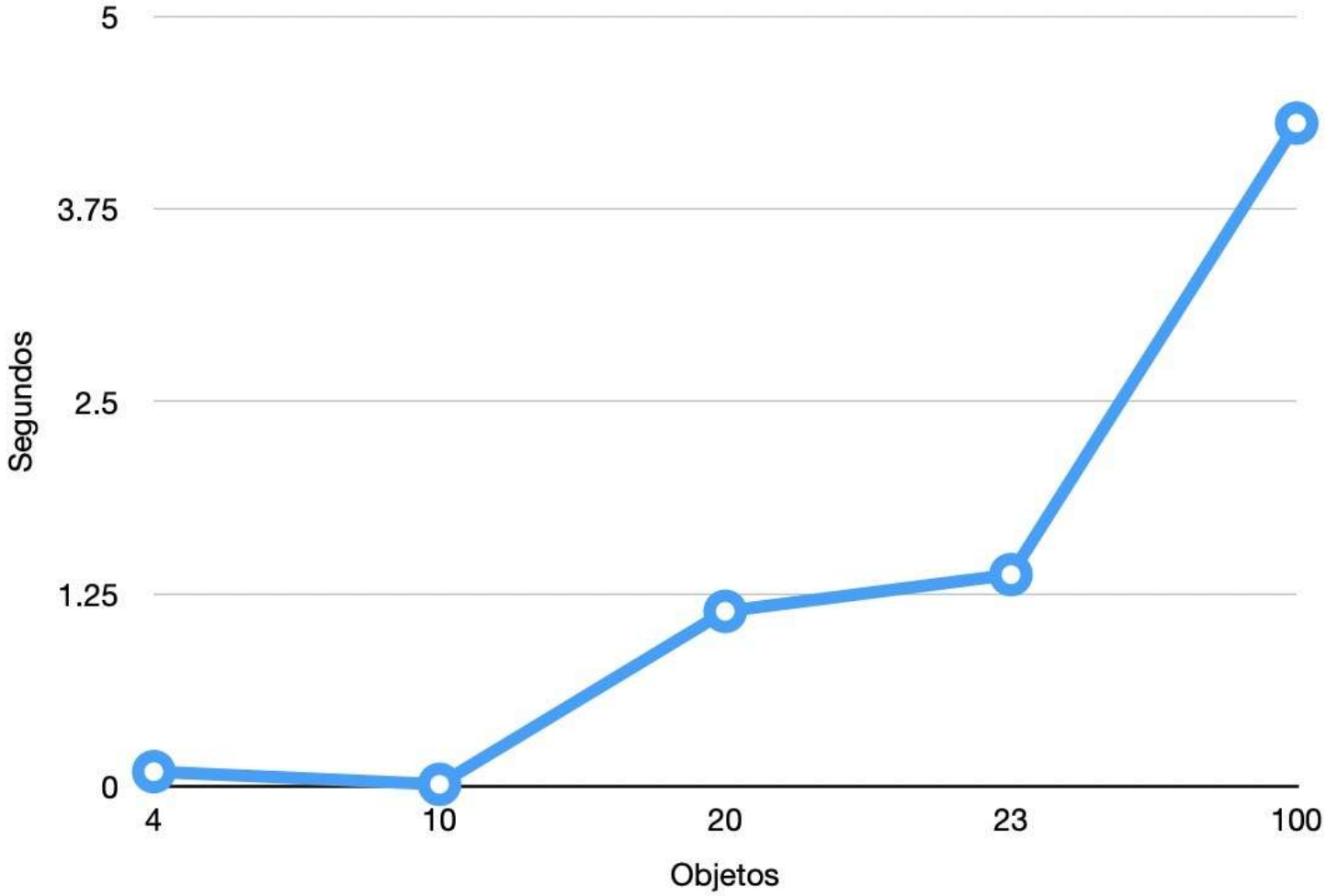
Los resultados de este algoritmo se compararon con el algoritmo informado A*, a continuación se expone el respectivo pseudocódigo, para más profundidad se puede acudir al repositorio corresponidiente a la Tarea 5.

Se utilizaron las mismas instancias en ambos análisis y se realizaron un número de 3 pruebas por cada instancia para poder mantener una mayor precisión y resultados, las isntancias utilizadas fueron:

▶ **Instancias:**

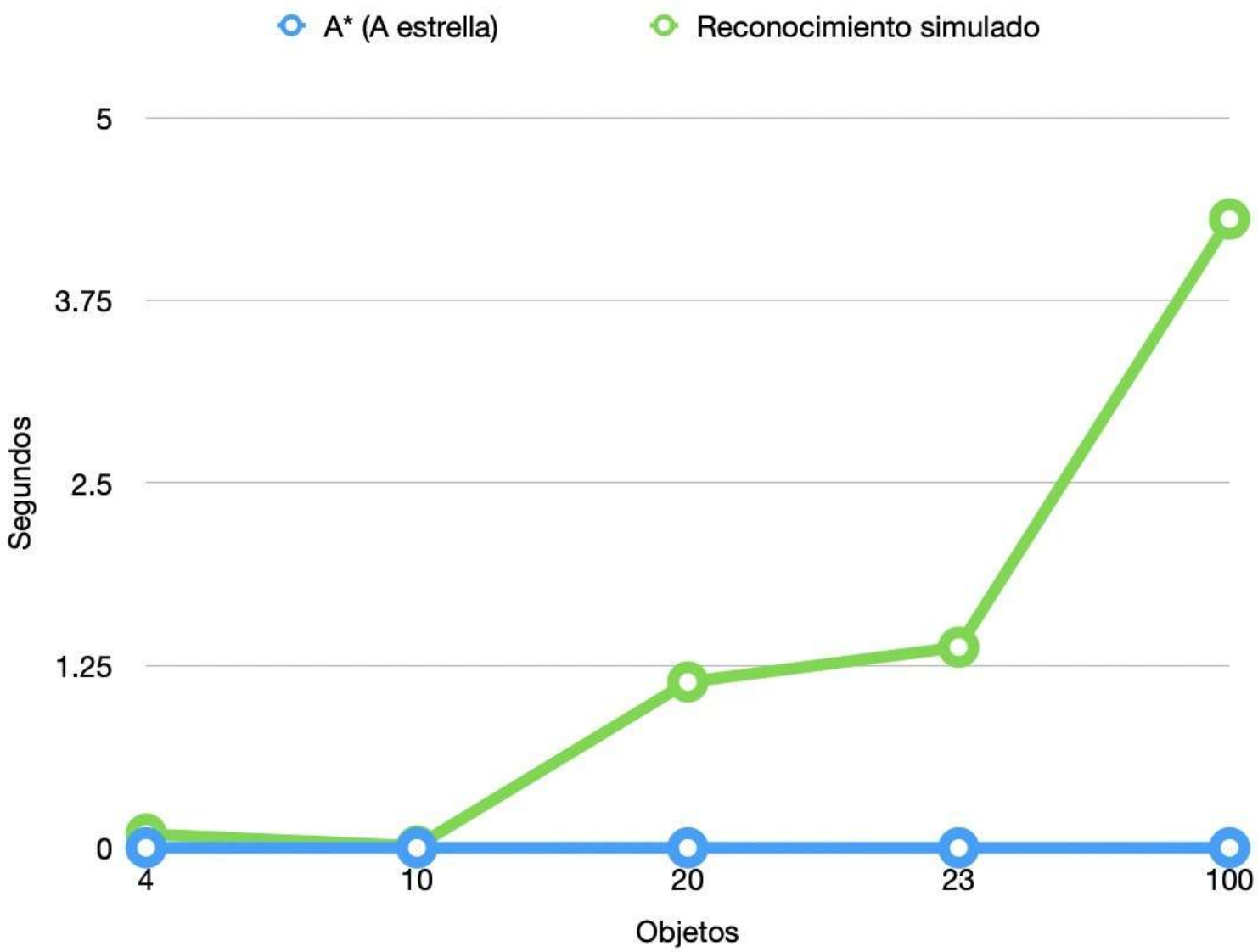
```
Algoritmo a_estrella

    Incluir "grafo" como gr
    Incluir "math"
    Incluir "heapq"

    Funcion distancia(nodo_a, nodo_b)
        x1 = longitud(nodo_a)
        y1 = longitud(nodo_a)
        x2 = longitud(nodo_b)
        y2 = longitud(nodo_b)
        Retornar raiz_cuadrada((x1 - x2) ^ 2 + (y1 - y2) ^ 2)
    FinFuncion

    Funcion a_estrella(nodo_inicial, nodo_final)
        abiertos = [(0, nodo_inicial)]
        cerrados = conjunto()
        padres = diccionario()
        costos = diccionario()
        ruta = diccionario()

        costos[nodo_inicial] = 0
        ruta[nodo_inicial] = [nodo_inicial]
```

```
        Mientras que abiertos no sea vacío Hacer
            costo_actual, nodo_actual = heapq.eliminar_menor(abiertos)

            Si nodo_actual = nodo_final Entonces
                Retornar ruta[nodo_actual], costos[nodo_actual]
            FinSi

            cerrados.insertar(nodo_actual)

            Para cada nodo_vecino, peso en gr.graph[nodo_actual] Hacer
                Si nodo_vecino está en cerrados Entonces
                    Continuar
                FinSi

                nuevo_costo = costos[nodo_actual] + peso
                Si nodo_vecino no está en costos o nuevo_costo < costos[nodo_vecino] Entonces
                    costos[nodo_vecino] = nuevo_costo
                    heuristica = distancia(gr.graph[nodo_vecino], gr.graph[nodo_final])
                    heapq.insertar(abiertos, (nuevo_costo + heuristica, nodo_vecino))
                    padres[nodo_vecino] = nodo_actual
                    ruta[nodo_vecino] = concatenar(ruta[nodo_actual], [nodo_vecino])
                FinSi
            FinPara
        FinMientras

        Retornar nulo
    FinFuncion

    grafica = {}

    grafo = gr.Grafo(grafica)

    Para cada g en grafica Hacer
        Para cada x en grafica Hacer
            Si g = x Entonces
                Continuar
            FinSi

            camino = a_estrella(g, x)
            Escribir("El camino mas corto de ", g, " a ", x, " es: ", camino[0], " con costo: ",
        FinPara
    FinPara

FinAlgoritmo
```


5
3.75
2.5
1.25
0
Segundos
4
10
20
23
100
Objetos

Objetos	Reconocimiento simulado	Promedios
4	0.0960898399353027	
4	0.0962727281188965	0.0960898399861654
4	0.0959069519042969	
10	0.013680458068847	
10	0.0135331153869629	0.013680378595988
10	0.0138275623321533	
20	1.13751935958862	
20	1.14948463439941	1.13736073176066
20	1.12507820129395	
23	1.37566256523132	
23	1.391780853271 48	1.37562282880147
23	1.35942506790161	
100	4.31449270248413	
100	4.24632835388184	4.30782620112101
100	4.36265754699707	

> A*


A* (A estrella)
Reconocimiento simulado
5
3.75
2.5
1.25
0
Segundos
4
10
20
23
100
Objetos

Objetos	A* (A estrella)	Promedios
4	0.0010027885437011 7	
4	0.0010154247283935 5	0.0010060071945190 4
4	0.0009998083114624 02	
10	0.0014290809631347 7	
10	0.0014568567276001	0.0014427105585734 1
10	0.0014421939849853 5	
20	0.0010008811950683 6	
20	0.0010077953338623	0.0010071115493774 4
20	0.0010126581192016 6	
23	0.0015039443969726 6	
23	0.0015103816986084	0.0015039443969726 6
23	0.0014975070953369 1	
100	0.0019240379333496 1	
100	0.0019364356994628 9	0.0019238789876302 1
100	0.0019111633300781 3	

> Reconocido simulado

El algoritmo de reconocimiento simulado tiene un tiempo de ejecución que aumenta exponencialmente con la cantidad de objetos, por lo que cuando el número de objetos es muy grande, el tiempo de ejecución puede ser prohibitivo. Además, aunque el algoritmo puede ser útil para encontrar soluciones aproximadas a problemas de optimización combinatoria en general, se han desarrollado métodos más especializados y eficientes para problemas de la mochila con un gran número de objetos, como programación lineal entera y algoritmos genéticos.

La razón por la cual el algoritmo de reconocimiento simulado se vuelve exponencialmente ineficiente para problemas con una gran cantidad de objetos es debido a la cantidad de soluciones posibles que deben ser exploradas. Conforme la cantidad de objetos aumenta, la cantidad de soluciones posibles crece de forma exponencial, lo que implica que se deben explorar una gran cantidad de soluciones para encontrar la mejor. El algoritmo de reconocimiento simulado explora un número limitado de soluciones, y aunque es capaz de encontrar una buena solución en muchos casos, para problemas grandes la solución encontrada puede estar lejos de ser la mejor. Por lo tanto, para problemas grandes, el algoritmo de reconocimiento simulado puede ser ineficiente y puede ser necesario buscar otros enfoques más adecuados.

> Se realizaron 3 iteraciones en cada configuración de grafo, y se graficó el promedio utilizado. Todas las pruebas fueron echas bajo el mismo ambiente del IDE Visual Studio Code. Las características del equipo en que se corrió fueron las siguientes: SO: Windows 11 Home Single Language x64
> Procesador: AMD Ryzen 5 5600H with Radeon Graphics 3.30 GHz RAM: 16,0 GB