UNCUYO
UNIVERSIDAD NACIONAL DE CUYO

FACULTAD DE INGENIERÍA

INTELIGENCIA ARTIFICIAL II
UNIDAD 1: BÚSQUEDA Y OPTIMIZACIÓN

Grupo Nº1

Alumnos:

- ***Eula, Adriano German (L:12336)***
- ***Reinoso, Maximiliano Gabriel (L:11754)***
- ***Sena, Julieta (L:11367)***

Resolución de los ejercicios:

https://colab.research.google.com/drive/1xaFQNJXtnguMP_Xpa-Deb3y2JL975uJA?usp=sharing

Ejercicio 1:

En el este primer problema tenemos que hacer uso del algoritmo de búsqueda A* para encontrar el camino más corto desde una posición en el espacio (posición inicial) de un robot de 6 grados libertad hasta una posición final (meta). Para poder estudiar mejor el desempeño del algoritmo. Hacemos que las posiciones espaciales de inicio y meta sean generadas aleatoriamente y además que entre el camino de una posición a otra se encuentren obstáculos también generados aleatoriamente. Cabe destacar que esto es solo para poder hacer un estudio del algoritmo para este tipo de tareas en la industria, ya que en una situación real un robot de estas características cuenta con una lista de tareas específicas con sus posiciones relativas en el espacio (Ej:Un robot en una línea de ensamble) y los obstáculos ya están cargados por el espacio de trabajo en el que se encuentra. Para la aplicación del algoritmo A* hicimos uso del concepto de clases y objetos en la programación del mismo. Principalmente la clase Nodos que es la que posee todos atributos de los nodos que serían la evaluación de la heurística (h), el costo desde el inicio (g) y la función de desempeño (f). La otra clase es la llamada clase A_estrella que es la que posee los atributos para la resolución del problema de búsqueda. Hacemos usos de listas (Listobs,LNO y LNC) para ir buscando cual es el nodo con menor costo f. En este código vamos buscando el FMC (f de menor costo) en la LNO que es la lista de posibles nodos a explorar y el que tenga menor valor de f lo extraemos y lo ponemos en la lista de nodos explorados (LNC) de esta manera vamos vaciando LNO y mejorando las posibilidades de encontrar la meta. También vamos teniendo en cuenta el desplazamiento total y la cantidad de movimientos que necesita el robot para llegar a la meta.

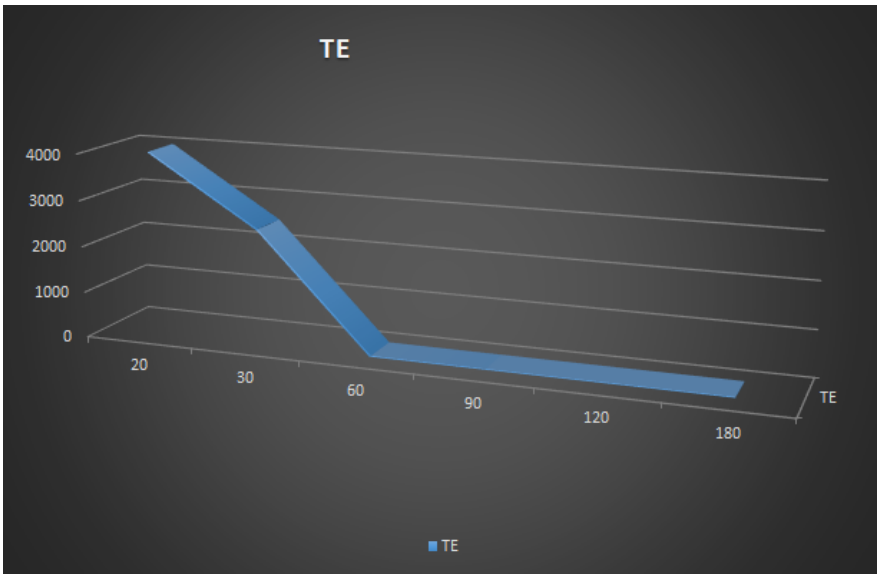
En la resolución de este problema vamos a notar una notable tendencia de que a medida que aumentamos el paso y el porcentaje de obstáculos que aleatoriamente se van a generar en el espacio el algoritmo va a tener mejor rendimiento o en este caso menor tiempo de ejecución. Pero esto no quiere decir que vamos a encontrar el camino desde el punto inicial hacia el final.Todo lo contrario, a medida que nosotros aumentamos el paso, ya estamos acotando las posibilidades de movimiento del robot considerablemente (su máximo es de 180°) y además si aumentamos el porcentaje de obstáculos vamos a notar que el algoritmo si bien va a ejecutarse rápido, la probabilidad de que no encuentre la solución buscada va a aumentar debido a que estamos restringiendo demasiado su entorno como para lograr que cumpla con la tarea.

Por otro lado si nos vamos al otro extremo, colocando un porcentaje de obstáculos casi nulo o nulo y un paso muy chico, que se asemeja con la realidad ya que los motores paso a paso llegan a tener un paso de 0.9 y 1.8 grados respectivamente. Lo que logramos es generar una cantidad de nodos posibles a explorar por el algoritmo que hace que el costo de explorar todo el árbol de búsqueda sea muy grande y casi innecesario ya que la tarea de moverse de un punto a otro llevaría demasiado tiempo y costo computacional volviendo así al algoritmo poco óptimo.

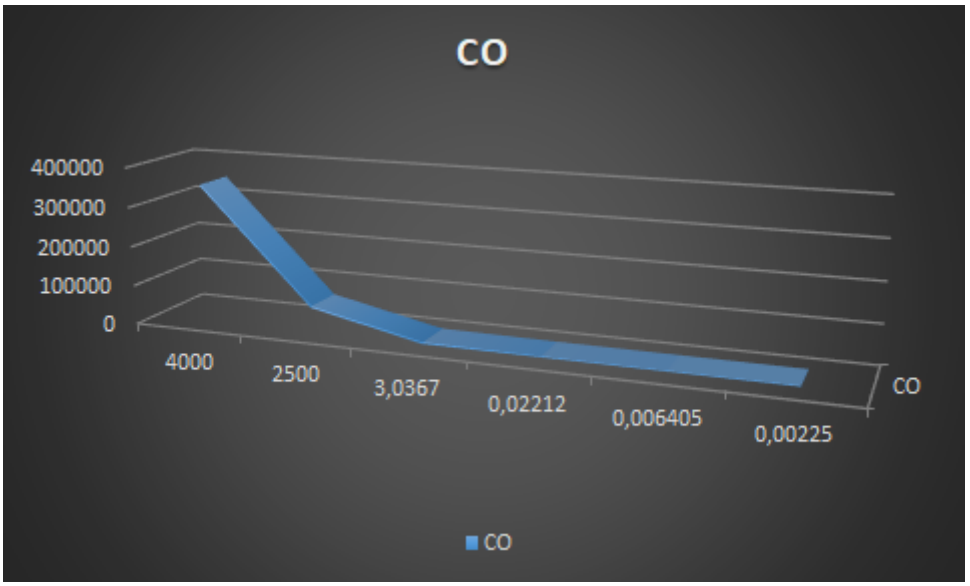
Obstáculos (%)	**25**	**40**	**60**	**90**
Paso 20°	*CO:250000* *Mov:NaN* *NE:NaN* *TE:NaN*	*CO:400000* *Mov:NaN* *NE:NaN* *TE:NaN*	*CO:600000* *Mov:NaN* *NE:NaN* *TE:NaN*	*CO:900000* *Mov:NaN* *NE:NaN* *TE:NaN*
Paso 30°	*CO:29412* *Mov:NaN* *NE:NaN* *TE:NaN*	*CO:47059* *Mov:12* *NE:3711* *TE:1072s*	*CO:70589* *Mov:NaN* *NE:NaN* *TE:NaN*	*CO:105884* *Mov:15* *NE:3749* *TE:362s*
Paso 60°	*CO:1024* *Mov:10* *NE:1180* *TE:6.1s*	*CO:1638* *Mov:10* *NE:1060* *TE:5.15s*	*CO:2457* *Mov:7* *NE:101* *TE:0.2778s*	*CO:3686* *Mov:7* *NE:66* *TE:0.619s*
Paso 90°	*CO:182* *Mov:6* *NE:118* *TE:0.042s*	*CO:291* *Mov:5* *NE:48* *TE:0.0223s*	*CO:437* *Mov:4* *NE:16* *TE:0.0062s*	*CO:656* *Mov:7* *NE:66* *TE:0.019s*
Paso 120°	*CO:16* *Mov:3* *NE:8* *TE:0.0012s*	*CO:25* *Mov:5* *NE:31* *TE:0.0038s*	*CO:38* *Mov:3* *NE:5* *TE:0.00086s*	*CO:57* *Mov:3* *NE:4* *TE:0.000702s*
Paso 180°	*CO:16* *Mov:4* *NE:16* *TE:0.0028s*	*CO:25* *Mov:3* *NE:7* *TE:0.00182s*	*CO:38* *Mov:7* *NE:30* *TE:0.0033s*	*CO:57* *Mov:0* *NE:10* *TE:0.0011*

CO:Cantidad de Obstáculos-Mov:Números de movimientos-NE:Nodos explorados-TE:Tiempo de Ejecución en segundos

En la tabla anterior se puede observar que para el paso de 20° solo tenemos los datos de la cantidad de obstáculos que vamos a generar. Pero no se pudo obtener los datos a analizar ya que la plataforma donde lo corrimos sobrepasaba sus capacidades de cálculo. Por otro lado esto afirma que en la realidad hay que trabajar con un promedio de restricciones no generadas aleatoriamente. A continuación se muestran dos gráficos representativos de la tabla anterior. Podemos observar que las curvas a medida que se achica el paso la dificultad de resolución aumenta exponencialmente y el punto de inflexión es una vez que disminuimos el valor del paso por debajo de los 60°.



Tiempo de ejecución promedio en función del paso del robot



Cantidad de obstáculos promedio en función del tiempo de ejecución promedio para cada paso

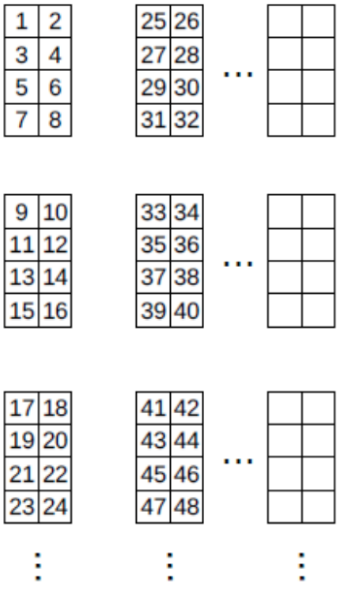
En conclusión cuando queremos analizar el desempeño de un algoritmo y llevarlo a una situación real. Debemos considerar que a veces en esta clase de problemas además de considerar la eficacia del algoritmo, también hay que tener en cuenta si el costo computacional y de ejecución vale la pena solo para asegurar que exista el movimiento (caso del robot). En este caso particular nos dimos cuenta que a veces es conveniente agregar un número de restricciones (obstáculos) para desechar ciertos nodos a explorar en el árbol de búsqueda y de esta manera optimizar el desempeño del algoritmo y por ende del robot. Por ejemplo en una planta industrial se podría delimitar un cierto espacio de trabajo y por más que la forma física del robot le

permita expandirse fuera de esa delimitación no lo haga a razón de mejorar su desempeño en conjunto con el algoritmo de búsqueda

Ejercicio 2:

En este ejercicio haremos uso del algoritmo A* para la resolución al problema de encontrar el camino más corto entre 2 puntos dados. Dicho algoritmo es completo (si existe una solución, la encuentra) y ordenado (la primera solución encontrada es la mejor) si su heurística es admisible. Esto significa principalmente que no debe sobreestimar el costo de llegar al objetivo. Para nuestro caso particular, la heurística utilizada es la distancia de Manhattan entre el inicio y la meta.

Por otro lado, el espacio de búsqueda utilizado viene determinado con la siguiente forma:



El algoritmo utilizado posee principalmente dos clases (clase Nodo y clase A*). La clase Nodo se encarga de evaluar los atributos de cada nodo (en este caso, su costo desde el punto de inicio g(n), su heurística h(n) y su función de evaluación f(n)). La clase A* es la que posee los métodos de resolución iterativa del problema. Los parámetros del algoritmo son: posición inicial, posición final, cantidad de filas,

cantidad de columnas, posición de los estantes. Además, esta clase trabaja con 3 listas: Lista Abierta (nodos por expandir), Lista Estantes (nodos que no pueden ser expandidos, en este caso las posiciones ocupadas por las estanterías), y Lista Cerrada (nodos ya expandidos).

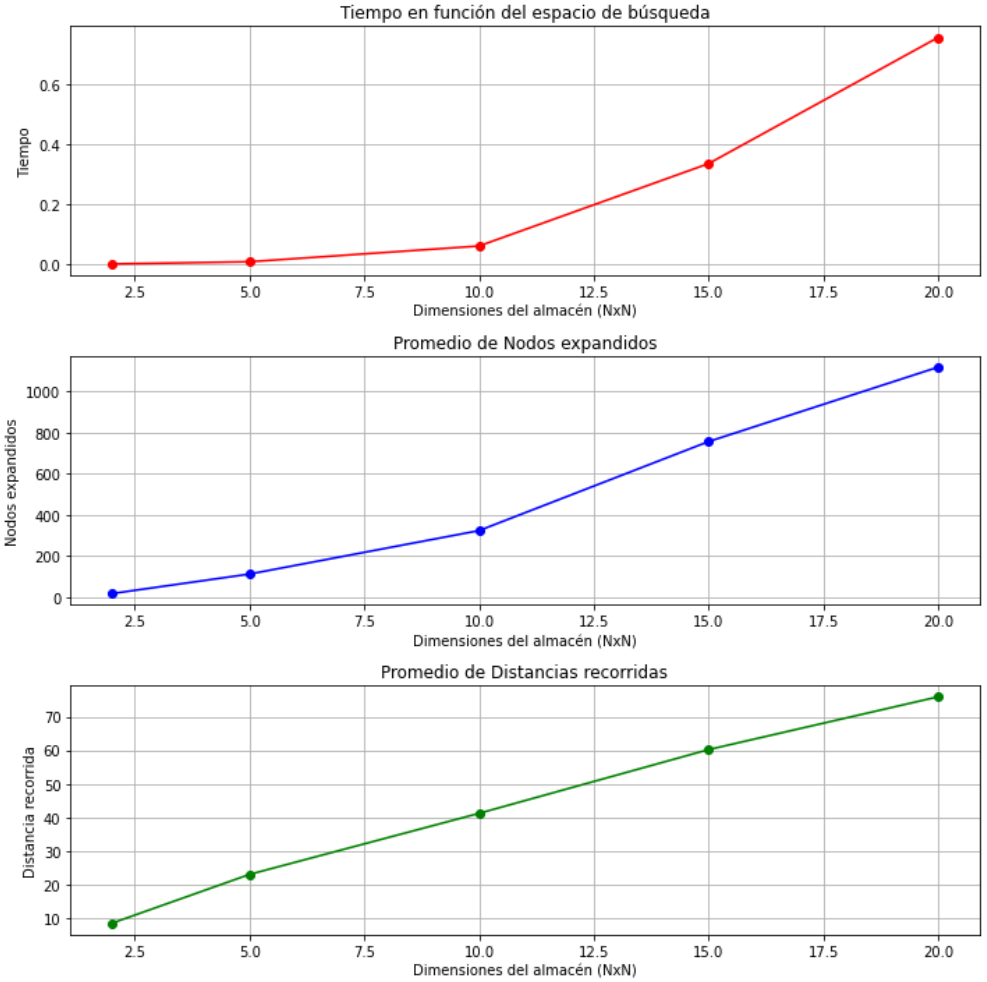
El proceso de búsqueda consiste en tomar el nodo con menor f(n) (en la primera iteración es el INICIO) y expandirlo (encontrar sus nodos vecinos). Dicho nodo pasa a estar en la lista cerrada, y luego de corroborar que sus vecinos son nodos válidos (fuera de Lista Estantes y dentro del espacio de búsqueda) se agregan a la Lista Abierta como posibles nodos a expandir.

Luego se selecciona de dicha lista aquel con el menor f(n), pero antes de expandirlo se verifica que no sea el nodo meta (para el h(n)=0), si es el caso, el problema se termina. La función f(n) se calcula para cada nodo, y sabemos que g(n) aumenta en 1 a medida que nos alejamos del inicio, y h(n) va decreciendo (nos acercamos a la meta). Para esto se emplean varios métodos dentro de la clase A*, que realizan tareas de verificación, creación de vecinos, evaluación de los nodos, etc.

Para evaluar el desempeño del algoritmo, se fijó el punto inicial en el origen de coordenadas, posición (0,0), y la posición final se obtuvo al azar. Se registró el tiempo promedio de resolución, los nodos expandidos en promedio y la distancia promedio de la solución. Se realizaron 50 repeticiones en los tamaños de espacio de [2,5,10,15,20].

Los resultados obtenidos son:

Tamaño del almacén	*2x2*	*5x5*	*10x10*	*15x15*	*20x20*
Tiempo promedio de búsqueda	*0.00058*	*0.0083*	*0.049*	*0.312*	*0.783*
Nodos expandidos en promedio	*24*	*113*	*270*	*807*	*1118*
Distancia recorrida en promedio	*9.72*	*23.02*	*35.98*	*65.32*	*75.3*



Como podemos ver en la primera figura de la gráfica anterior, vemos una clara tendencia exponencial en cuanto al aumento del tiempo de búsqueda con respecto al espacio de búsqueda.

Luego en la segunda figura, también se puede ver una leve tendencia exponencial en el aumento de los nodos expandidos respecto del tamaño del almacén.

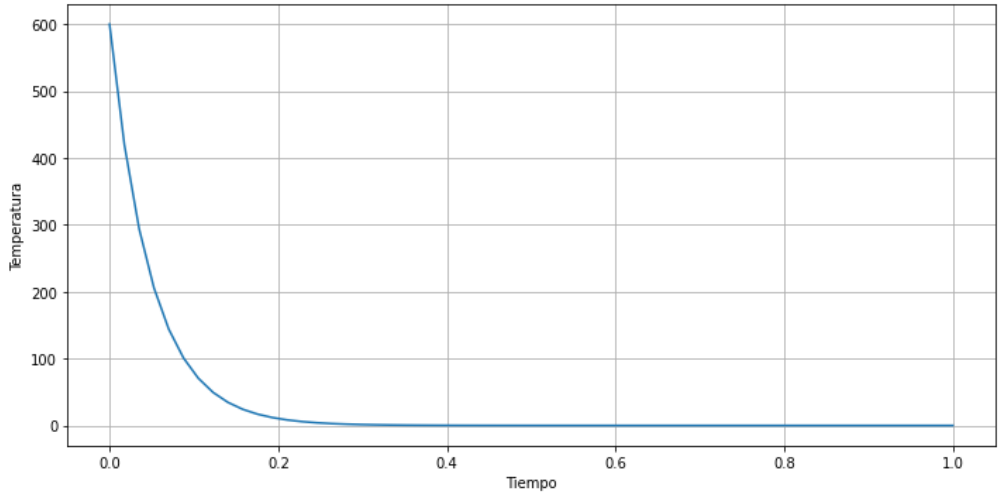
Por último, en cuanto a la distancia recorrida, vemos una tendencia lineal respecto del tamaño del almacén, esto se debe a que en cuanto a la relación de proporciones, dicha relación dimensional es casi constante.

Ejercicio 3:

En este ejercicio haremos uso del Algoritmo de Temple Simulado, así como también de la implementación anterior del algoritmo A*. Se posee el mismo entorno de búsqueda que el ejercicio anterior.

Los parámetros del algoritmo son Temperatura inicial, Temperatura final, los objetos a analizar y el factor de enfriamiento.

Como medida de enfriamiento se decidió tomar el valor de la temperatura y multiplicarlo por el factor de enfriamiento. Es decir que este factor siempre tiene que ser menor a 1. A continuación se observa la curva de enfriamiento para 600 grados de temperatura inicial.



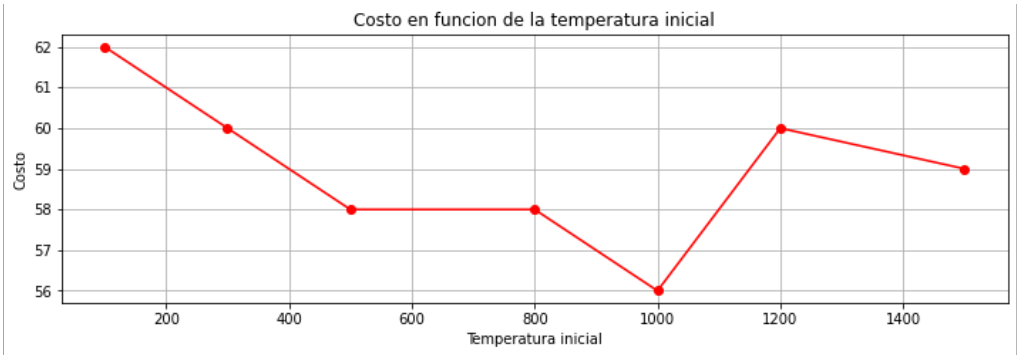
Además de la clase A* del ejercicio anterior, se encuentra el algoritmo de temple, que consta de una clase Temple. Esta recibe como parámetros de entrada los valores ya mencionados. A su vez la clase Temple hace uso de la clase Carrito que representa al operario o robot que tendrá que recolectar los objetos en el orden dado y este mismo nos informaría cual ha sido el costo del camino, cada cuadro del mapa vale 1 a fines simplificar el valor del costo. También se incorporó la clase Caché, la cual guarda la distancia entre distintas posiciones del almacén a fines de agilizar los cálculos de las distancias y de esta forma hacer más eficiente la búsqueda del mejor orden de recolección. Esto permitió disminuir notablemente los tiempos de ejecución. Si bien no se realizaron pruebas previas a su incorporación al algoritmo, se pudo percibir la reducción de los tiempos.

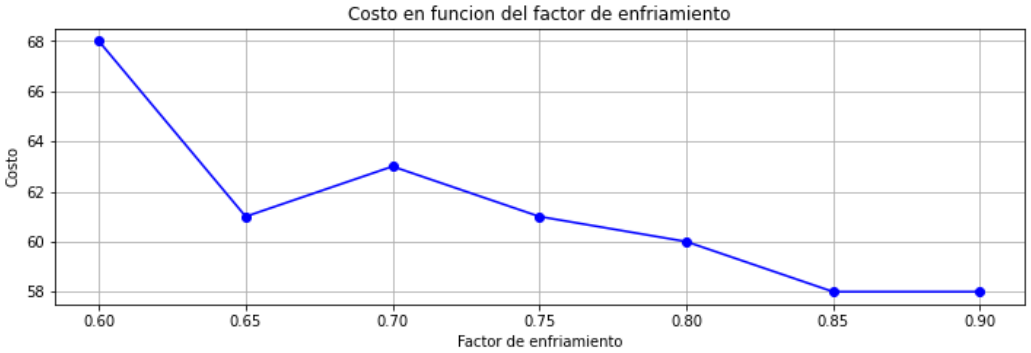
El algoritmo para cada iteración realiza una permutación simple de la lista de objetos y se calcula el costo de recolectarlos en dicho orden, se compara ese valor con uno previamente guardado, si el nuevo costo es menor pasa a ser el orden de objetos guardados, si el costo no es menor se calcula un valor de probabilidad para la selección en función de la temperatura T, a mayor T son mayores las probabilidades de elegir un orden de recolección peor.

Para evaluar el desempeño del sistema haremos 2 evaluaciones:

- Cómo varía el costo del camino en función de la Temp. Inicial a factor de enfriamiento constante.
- Cómo varía el costo del camino en función del factor de enfriamiento a Temp. Inicial constante.

Para esto se realizaron 100 ciclos con cada conjunto de parámetros con una lista de 10 objetos y se obtuvieron las siguientes figuras:



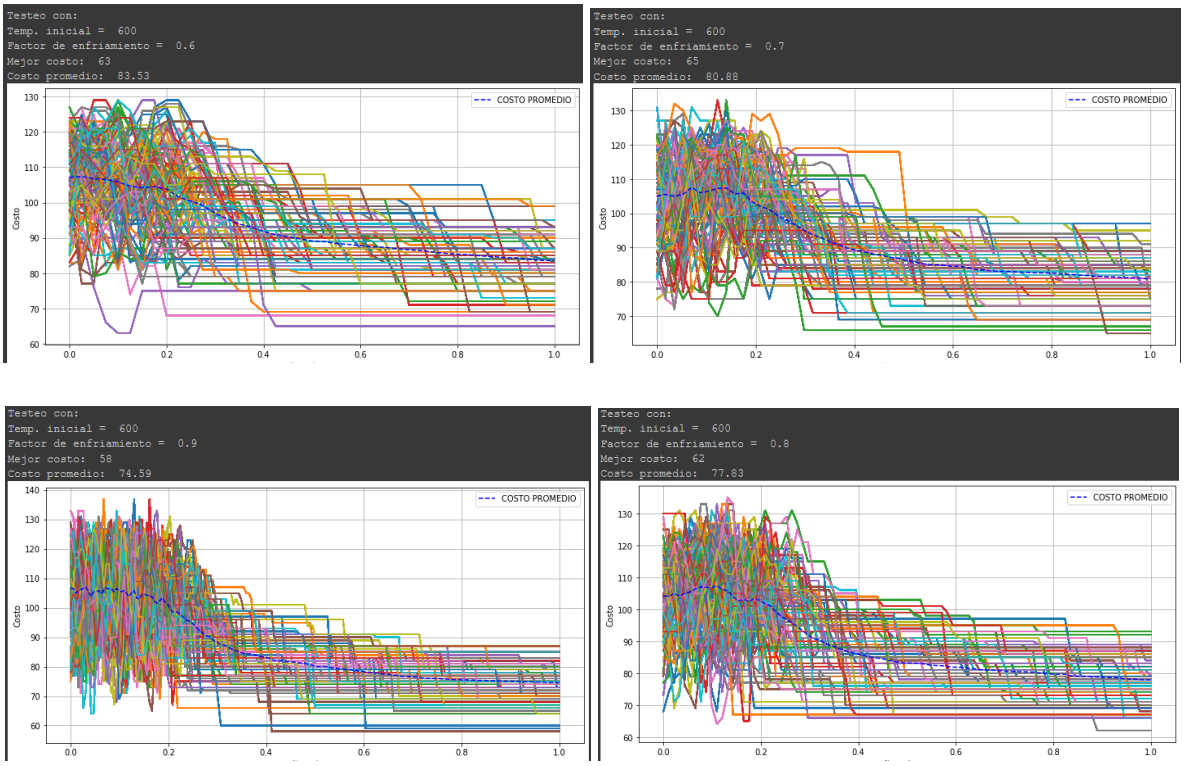


En primer lugar, podemos ver en la primer figura que a medida que aumentamos la temperatura, hay una leve tendencia en la disminución del costo del camino obtenido, si bien la relación no es directa, ya que depende de la estocasticidad del algoritmo, al partir de una temperatura inicial mayor el sistema se mueve de manera más aleatoria en en el inicio de la evolución provocando que pueda llegar a un mejor valor de mínimo local.

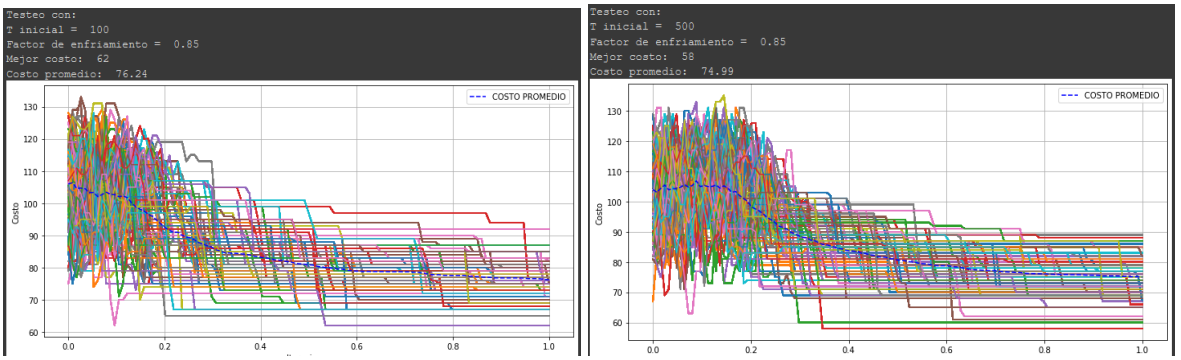
Luego, en la segunda figura, podemos ver que el aumento del valor del factor de enfriamiento, es decir, un decaimiento más lento de la temperatura, nos favorece en la búsqueda del valor mínimo local, esto se debe a que el algoritmo mantiene la energía suficiente para elegir soluciones peores que pueden luego desembocar en un mejor mínimo local por más tiempo.

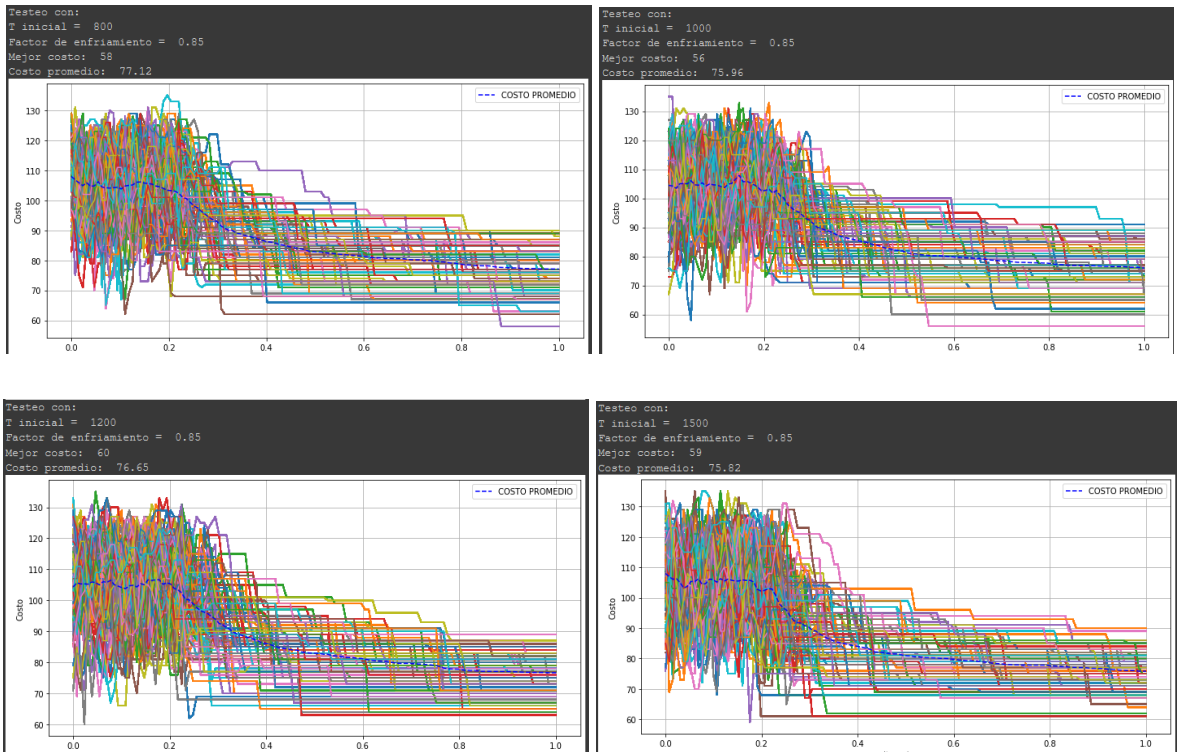
A continuación, mostramos otras gráficas relevantes del análisis realizado:

- En las siguientes gráficas observamos que a medida que aumentamos el valor del factor de enfriamiento, es decir, decaimiento lento de la temperatura, la estabilización de la solución se produce de manera más temprana en la evolución de las iteraciones. Esto quiere decir que a medida que aumentamos el factor de enfriamiento, más rápido llegamos a una mejor solución del problema. Esto en otras palabras significa que se explora más el espacio de búsqueda.



- En las siguientes figuras mostramos como para un mismo factor de enfriamiento, variamos la temperatura inicial y realizamos el cálculo del costo.





Para temperaturas bajas, las soluciones peores son rechazadas rápidamente en la evolución del algoritmo, en cambio, para temperaturas más altas, estas se mantienen presentes por más tiempo y son las que luego pueden presentar una mejor solución del problema, aunque esto conlleva a una cantidad más significativa de iteraciones del algoritmo.

Finalmente, una vez realizadas las pruebas de rendimiento procedemos a su implementación haciendo uso de las listas de órdenes provistas por la cátedra, para las cuales se obtuvo la optimización del orden de recolección correspondiente de forma exitosa. En el mismo se realizan 10 iteraciones de cálculo por cada orden de productos y de las cuales se elige la mejor solución obtenida de todas las ejecuciones, de esta forma nos aseguramos de que a pesar de ser un algoritmo de búsqueda local, la solución entregada sea la mejor y lo más cercana a la óptima posible.

Ejercicio 4:

Para realizar este ejercicio, se utilizaron los algoritmos A* y Temple Simulado, con sus mismos parámetros de inicio. Inicialmente al algoritmo se le ingresan como parámetros iniciales el tamaño de la población y una lista de 2 valores que corresponden a las filas y columnas de las estanterías, con los cuales se genera el entorno de trabajo, en este caso definidos por un valor de 4x4 en el tamaño del almacén. La lista de órdenes con la que se calculará el valor de fitness del algoritmo es

la misma utilizada en el ejercicio anterior para la prueba de desempeño del algoritmo de Temple Simulado.

De forma general, el algoritmo procede a buscar en la Caché la distancia entre 2 puntos dados, en caso de no estar ahí guardada, se utiliza la clase A* para ubicar el camino más corto hasta el objetivo y luego guardar dicha distancia en la Caché para su uso posterior. Luego, utiliza el de Temple Simulado para para optimizar la búsqueda de todos los productos y finalmente mediante el algoritmo Genético, lograr optimizar la ubicación de los productos en los estantes del almacén en base a la forma óptima de buscarlos, es decir cambiando la ubicación de los productos en el almacén para lograr un mejor fitness total de recolección. La medida del fitness utilizada para el algoritmo es la suma de los costos de recolección óptimos de cada orden de productos, ya que esto nos indica el costo total para un conjunto de órdenes de productos dadas.

Al finalizar obtenemos una población con la misma cantidad de individuos que al inicio en caso de ser un valor par inicial, de lo contrario posee un individuo más. Esta población está optimizada en función al fitness de cada individuo, evolucionando los mejores individuos así como también manteniendo los mejores de la generación anterior, de esta forma logramos no perder posibles buenos candidatos y que la generación evoluciona de manera óptima. Como último paso, se procede a la selección del menor valor de fitness, que simboliza el menor costo de recolección de la lista de órdenes, y su individuo correspondiente, que en este caso es el almacén óptimo.

La mejor solución encontrada con un valor de fitness de 12712 fue:

47	88	51	65	109	3	7	120
116	83	93	77	72	105	22	20
123	66	1	107	14	100	110	24
43	59	68	15	17	57	19	92
50	71	63	13	104	103	9	39
29	81	38	46	75	53	58	54
26	33	5	126	96	117	8	30
25	56	45	79	111	36	115	49
94	106	51	35	44	84	32	41
48	61	90	124	108	99	27	74
113	12	67	70	118	86	114	80
18	42	97	37	6	4	122	34
121	87	125	127	60	91	82	55
69	76	89	78	98	119	85	112
62	2	11	102	31	21	28	128
64	10	16	23	101	73	40	95

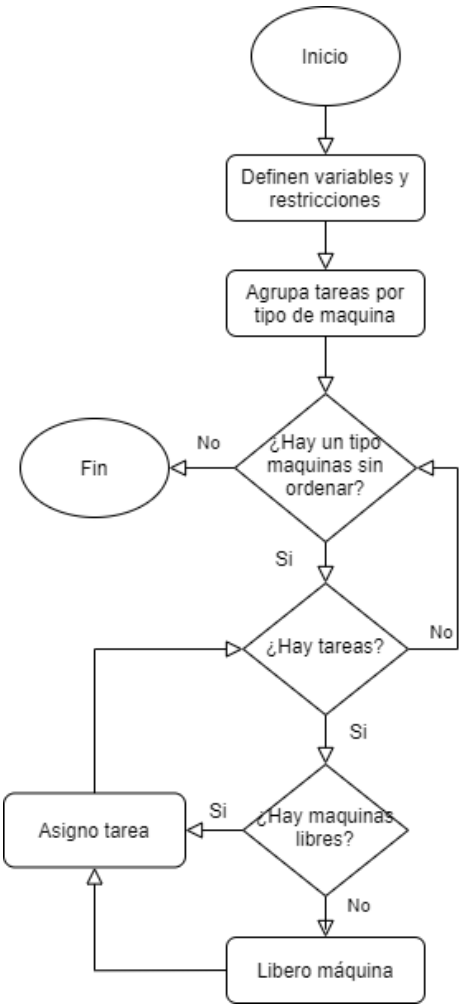
Ejercicio 5:

Para empezar, planteamos ciertas consideraciones a tener en cuenta. Primero, asumimos que hay 3 tipos de máquinas diferentes para realizar tareas específicas, por otro lado la cantidad de máquinas es un parámetro variable que se ingresa por consola, aunque la selección de la cantidad de máquinas que hay de cada tipo se hace de modo aleatoria (se utilizó una semilla a la hora de programar para contar con al menos una máquina de cada tipo).

Con respecto a las tareas, se hizo una suposición de que cada una demora entre 10 y 60 minutos en llevarse a cabo. Al igual que con la cantidad de máquinas disponible, el número de tareas a realizar se ingresa por consola, tanto el tipo de máquina que requiere como el tiempo de uso de la misma son valores completamente aleatorios.

Para la resolución del problema se planteó un algoritmo voraz de búsqueda local. Fue implementado con programación con objetos, consta con dos etapas de ordenamiento (dos métodos diferentes) para facilitar el cambio del código en caso de que en un futuro se quisiera cambiar los datos de inicialización por datos concretos, de esta forma no se generarían grandes complicaciones.

Como punto de partida definimos las variables con sus respectivas restricciones (tiempo y tipo de máquina). La primera etapa consiste en agrupar dichas tareas en listas dependiendo del tipo de máquina que requieren y en contar la cantidad de máquinas que se tiene de cada tipo. Luego, en la segunda etapa de ordenamiento se buscan máquinas libres y se asignan dichas tareas. En caso de no contar con máquinas desocupadas se liberan "con el paso del tiempo". El criterio de selección que se utilizó se basa en la duración de cada tarea, se prioriza asignar la tarea que requiere un tiempo similar al tiempo que necesita una máquina ocupada para liberarse, de esta manera todas las máquinas tendrán periodos de trabajo similares.



Este proceso se repite hasta que no queden tareas por asignar, luego se procede a ordenar otro tipo de máquina de la misma manera y así sucesivamente hasta haber acabado con la totalidad de las tareas.