UNCUYO
UNIVERSIDAD NACIONAL DE CUYO

FACULTAD DE INGENIERÍA

Trabajo Práctico N°1:

Búsqueda y Optimización

Inteligencia Artificial II

Profesor:

Dr. Ing. Martin Marchetta

Grupo N°1

Estudiantes:

Juan Ignacio Luna

Mario Fernando Bustillo

Rodrigo Aleo

Facultad de Ingeniería, Universidad Nacional de Cuyo

Ciclo Lectivo 2023

Tabla de contenidos

Introducción

El presente trabajo práctico tiene como objetivo la optimización de la recogida de paquetes en un almacén mediante el uso de tres algoritmos de Inteligencia Artificial: A estrella (A*), Temple Simulado y Algoritmo Genético. En este proyecto, se aborda el problema de la ubicación de los productos en el almacén y se busca maximizar la eficiencia en la recolección de paquetes por parte de los trabajadores. Para ello, se ha planteado un conjunto de requerimientos que incluyen minimizar el tiempo de recorrido y minimizar el número de pasos necesarios para recoger los paquetes. Los tres algoritmos serán utilizados en conjunto para lograr una implementación final que logre obtener los mejores resultados. La implementación se ha realizado utilizando el lenguaje programación python. 3.0 y Jupyter notebook.

Algoritmo A Estrella (A*)

Características del algoritmo

- Es un algoritmo de búsqueda global
- Tenemos un consumo de memoria que va creciendo a medida que avanza la búsqueda.
- Cada nodo es un estado y se van generando dichos estados a medida que se van recorriendo los nodos.
- Se convierte exponencial en la profundidad, es decir si en la ramificación está muy abajo la solución.
- Utilizamos una función f(n) = g(n) + h(n). [Esto, por cada nodo nos da una estimación del costo total para llegar a la meta cuando pasamos por cada nodo] o g(n) es el costo de la ruta desde el nodo raíz hasta el nodo n. o h(n) costo estimado desde el nodo n al nodo objetivo. (Heurística)

Problemática a resolver

El ejercicio consiste en aplicar el algoritmo de A* para encontrar la ruta más corta entre dos puntos, dadas las coordenadas de estas posiciones, dentro de un almacén con dimensiones preestablecidas.

Implementación

Se comienza estableciendo el espacio de trabajo. Se define una matriz de nodos, donde algunos de estos nodos representan casillas donde se almacenan productos y, el resto representa casillas de pasillos por donde puede moverse el agente..

Para el caso, se consideraron únicamente 48 espacios de almacén dentro una matriz de nodos de 10x20 nodos, esto con la finalidad de reducir el tiempo de ejecución computacional.

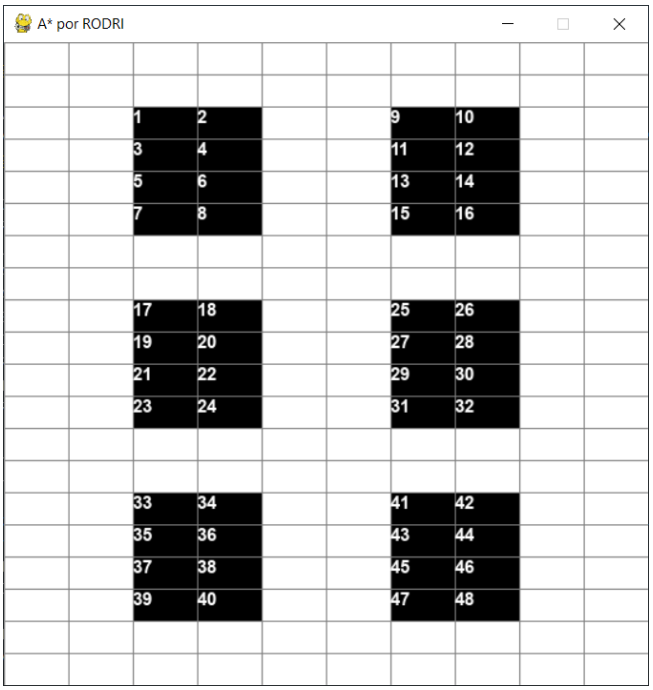


Figura 1: Espacio de búsqueda para el algoritmo A estrella.

Después de establecer el espacio de búsqueda, se procede a crear una interfaz gráfica utilizando la librería Pygame. En esta interfaz, el usuario puede seleccionar un punto de inicio en cualquier casilla blanca (pasillo) y un punto objetivo en cualquier casilla negra (estante) dentro del espacio de almacén.

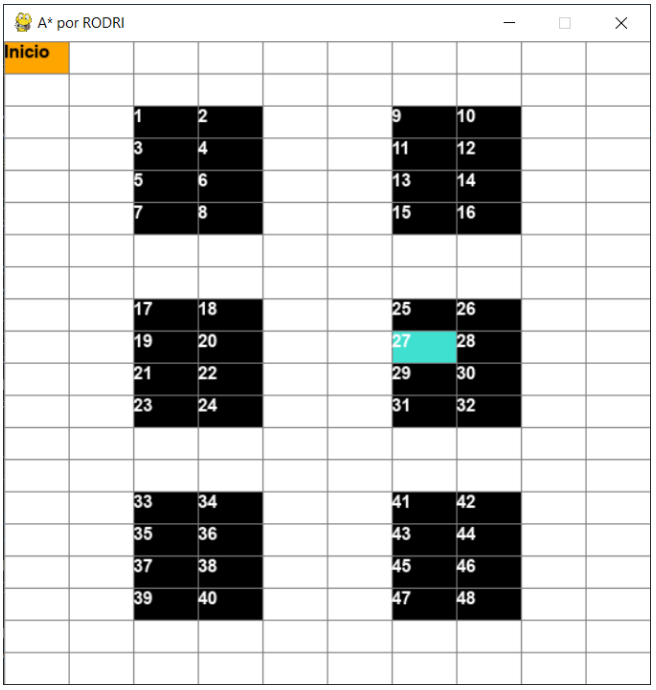


Figura 2: Punto de Inicio con coordenadas(0,0) y punto objetivo el estante 27.

Al pulsar espacio, comienza el algoritmo principal.

Relaciones entre nodos

En primer lugar, es necesario establecer cuáles son los nodos vecinos de un nodo determinado. Si nos encontramos sobre un nodo, sus vecinos serán los nodos ubicados en las direcciones arriba, abajo, izquierda y derecha. Sin embargo, es importante tener en cuenta que los almacenes de color negro son considerados obstáculos y por lo tanto no son clasificados como vecinos.

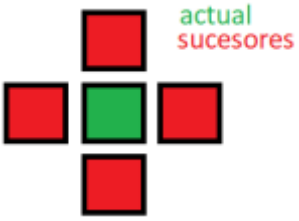


Figura 3: Movimientos posibles dentro del espacio de búsqueda.

Es relevante destacar que las estanterías sólo pueden ser accesibles desde los costados y no desde la parte superior o inferior en el caso de las estanterías ubicadas en las esquinas. Este

hecho, que implica que solo hay un punto de acceso a ciertos espacios de almacenamiento, simplifica el programa de optimización.

Inicialización para A*

La puntuación F de cada nodo se determina a partir de la suma de dos componentes: G (costo de camino) y H (función heurística). En cuanto a la función heurística, ésta se define como la cantidad de pasos necesarios para ir desde la posición actual del nodo hasta el nodo objetivo. Es importante señalar que el puntaje H es constante para cada nodo, es decir, no cambia durante el proceso de búsqueda.

Para comenzar la búsqueda, en una primera instancia se debe calcular el puntaje H para todos los nodos, y se inicializan G y F como infinitos. Asimismo, es necesario establecer que ningún nodo es padre de forma inicial, ya que la búsqueda comienza en el nodo raíz y a medida que se van explorando los nodos vecinos, se van estableciendo los nodos padres correspondientes.

Ejecución del Algoritmo A*

El nodo inicial, cuyo valor de G es igual a cero (G=0), se coloca en la cola de nodos a analizar. Dado que es el único nodo en la cola, será el primer nodo en ser seleccionado para el análisis. A continuación, se analizan los nodos vecinos al nodo seleccionado.

Al moverse un paso hacia uno de los vecinos, el valor de G aumenta en 1. Si el valor de G actualizado es menor que el valor de G que ya tenían los vecinos (recordemos que se inicializan en infinito), entonces se actualiza el valor de G de los nodos vecinos y se designa al nodo seleccionado como el padre de esos vecinos. Realizar este análisis es importante ya que si en algún momento se encuentra un camino más corto, se puede actualizar el valor de G y el padre del nodo correspondiente, lo que nos asegura obtener siempre el camino más corto posible y evitar dar vueltas innecesarias.

Una vez que se ha calculado el valor de G para un nodo, se puede calcular el valor de F correspondiente a cada uno de sus nodos vecinos. Si el valor F actualizado es menor que el valor F que ya tenían (recordemos que F se inicializa en infinito), entonces se colocarán estos nodos vecinos en la cola de nodos a analizar (open set, representados en color verde).

Una vez que se han analizado todos los vecinos de un nodo, éste se extrae de la cola y se marca como visitado (close set, representado en color rojo). Cabe destacar que de los nodos que se encuentran en la cola de análisis, el próximo nodo a ser seleccionado siempre será aquel que tenga el valor de F más bajo (es decir, que se encuentre más cerca del objetivo). Es posible que un nodo que ya haya sido analizado vuelva a ser colocado en la cola de análisis si logra obtener una puntuación F menor a la que ya tenía.

En algún momento del proceso, una casilla tendrá como vecino al nodo objetivo. En este punto, el algoritmo se detiene y utiliza el atributo “padre” que tiene cada nodo para hacer backtracking desde el nodo objetivo hasta el nodo inicial, de modo que se pueda establecer el camino final.

En caso de que el nodo objetivo esté bloqueado o sea inaccesible, la cola de análisis eventualmente se vaciará, y en ese momento el algoritmo finalizará, informando que no se pudo encontrar una solución.

Resultados obtenidos

Con el algoritmo implementado, se han obtenido tiempos de ejecución que varían entre 0,017 s y 0,278 s, en función de la distancia entre el nodo inicial y el espacio de almacenamiento seleccionado. Sin embargo, es importante tener en cuenta que el rendimiento del algoritmo se ve afectado por la utilización de la interfaz gráfica de Pygame.

En cualquier caso, dada la distribución del espacio de búsqueda que se ha considerado, el algoritmo siempre será capaz de encontrar una secuencia de movimientos que conduzcan al nodo objetivo, ya que todas las casillas son accesibles desde cualquier punto de origen.

A continuación se muestra la ejemplificación de la trayectoria más corta entre la casilla inicial con coordenada (0,0) y la casilla objetivo cuyo estante es el 27.

Figura 4: El camino óptimo coloreadas en violeta, las casillas close set coloreadas en rojo y las open set coloreadas en verde.

Temple Simulado

Características del algoritmo

- Es un algoritmo de búsqueda local
- Tiene un consumo de memoria constante (inician y terminan con el mismo consumo de memoria)
- Mantienen un único estado (o conjunto pequeño)

Problemática a resolver

Dada una orden de pedido, que incluye una lista de productos del almacén anterior que deben ser despachados en su totalidad, determinar el orden óptimo para la operación de picking mediante Temple Simulado.

Implementación

Dataframe de distancias

Para resolver este ejercicio, es necesario aplicar el algoritmo A* de forma repetida. Para ahorrar tiempo de cálculo, se ha utilizado la librería Pandas de Python para almacenar los resultados de la ejecución del algoritmo en un dataframe.

En concreto, se ha ejecutado el algoritmo A* desde un espacio de almacenamiento hasta todos los demás, y se ha almacenado la longitud del camino resultante en el dataframe. Como resultado, se ha obtenido un dataframe de 48x48.

	1	3	5	7	17	19	21	23	33	35	...	14	16	26	28	30	32	42	44	46	48
1	0	1	2	3	6	7	8	9	12	13	...	11	12	13	14	15	16	19	20	21	22

Figura 5: Dataframe con la distancia entre cada punto.

En esta imagen podemos ver, por ejemplo, la distancia que se requiere para ir desde los puntos de almacenamiento que se encuentran en las columnas hasta el espacio de almacenamiento número 1. Luego, se procedió a guardar el dataframe como un archivo .csv, lo que permite cargar estos datos en cualquier momento y acceder a ellos rápidamente, sin necesidad de tener que volver a ejecutar el algoritmo A*.

Inicialización

Mediante una interfaz gráfica, el usuario puede seleccionar una posición inicial desde donde parte el operario y una posición final a donde debe llegar, así como también seleccionar cuántos productos debe buscar en el almacén. Para ahorrar tiempo de cálculo, se agregan las casillas de Inicio y de Final al dataframe de distancias y se realiza el algoritmo A* a las casillas

de cada uno de los productos seleccionados, de manera que las distancias ya queden guardadas y no sea necesario volver a calcularlas.

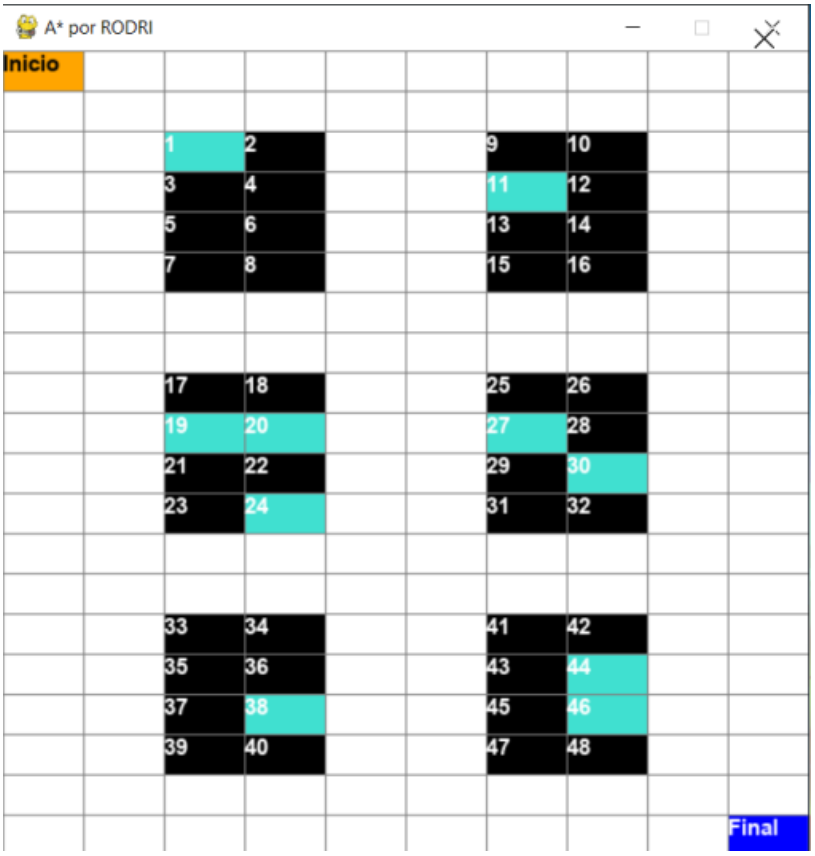


Figura 6: Punto de picking con coordenadas(0,0), Punto de packing con coordenadas(19,9) y puntos objetivos los estantes 1, 11, 19, 20, 24, 27, 30, 38, 44 y 46.

Ejecución del Algoritmo de Temple simulado

El algoritmo de temple simulado es una técnica heurística de optimización global que se utiliza para buscar soluciones en problemas de optimización combinatoria. El algoritmo busca mejorar la solución de un problema al permitir transiciones a soluciones peores en un principio, pero con una probabilidad decreciente a lo largo del tiempo. Esto permite escapar de los óptimos locales y explorar nuevas soluciones en el espacio de búsqueda.

El programa desarrollado inicialmente establece una solución inicial para la cual es el orden de los productos a recoger en el orden en el que fueron seleccionados. A esta solución se le calcula una puntuación que consiste en sumar las distancias del dataframe para ir desde el Inicio y, pasando por cada producto en el orden que dicta la solución, para luego ir hasta el final.

Luego se procede a generar soluciones vecinas intercambiando el orden de dos productos consecutivos en la solución inicial. Si la diferencia entre la puntuación de la solución vecina y la puntuación de la solución inicial es negativa, la solución vecina pasa a ser la nueva solución inicial en la próxima iteración.

El objetivo es minimizar la puntuación de la solución, es decir, minimizar la distancia total recorrida por el operario en el almacén para recoger todos los productos y llegar al punto de despacho final. En caso que esto no se cumpla, se obtiene un número al azar entre 0 y 1 y si es menor a la función de Boltzmann:

$$Boltzmann = e^{\frac{-\Delta E}{T}}$$

Donde ΔE es la diferencia entre la puntuación de las solución inicial y la vecina y T es la temperatura.

Mientras mayor sea la temperatura y mientras más cercano a 0 sea ΔE, más probabilidades hay de que una solución peor sea aceptada, lo que ayuda a evitar que el algoritmo se atasque en máximos locales. Este proceso se repite L veces antes de disminuir la temperatura en un 10%. La temperatura se disminuye en cada iteración hasta que T < T_final. Cuanto menor sea la temperatura, menor será la probabilidad de que el algoritmo acepte una solución peor. Durante el proceso, se mantiene una variable que guarda la mejor solución encontrada hasta el momento, que es la que se devuelve una vez que el algoritmo ha finalizado su ejecución.

Resultados obtenidos

Consideremos el siguiente pedido: 1, 11, 19, 20, 24, 27, 30, 38, 44 y 46, cuya coordenada de picking es (0,0) y coordenada de packing (9,19). Al ejecutar el programa desarrollado, se obtiene la mejor secuencia de búsqueda es 1, 19, 20, 11, 27, 20, 24, 38, 30, 44 y 46, con un costo de camino de 54 unidades.

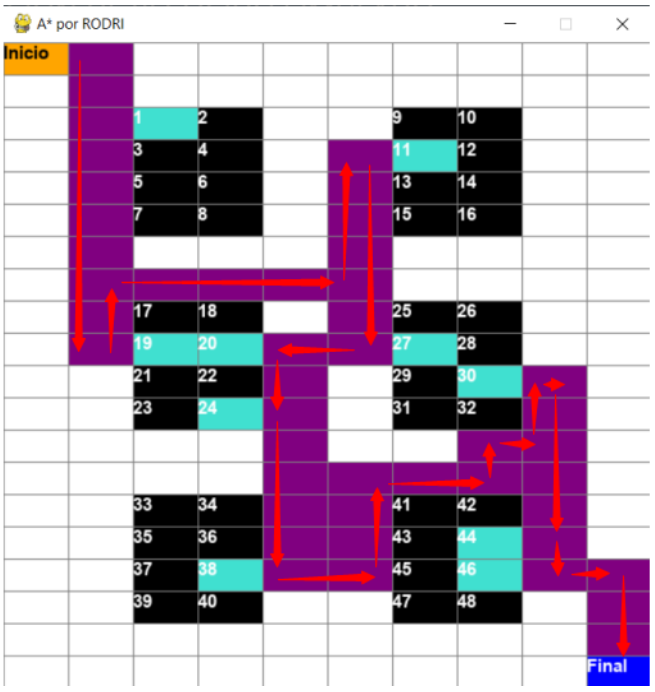


Figura 7: Resolución gráfica al pedido solicitado

Para verificar la calidad del algoritmo se lo ejecutó 500 veces con la misma orden de productos y variando los parámetros. Los resultados obtenidos fueron los siguientes:

Tinicial	Tfinal	L	% aciertos	Tiempo de ejecución para 1 iteración
50	1	300	74%	0.49 s
100	1	100	51%	0.19 s
100	1	200	59%	0.39 s
50	0.01	100	41%	0.35 s
500	100	100	0%	0.07 s
500	1	100	45%	0.26 s
25	1	500	77%	0.65 s
50	1	500	80%	0.81 s
25	1	300	68%	0.40 s
50	1	1500	96%	2.44 s

Los parámetros resaltados en celeste han demostrado tener una precisión mayor en términos de porcentaje de aciertos, pero su tiempo de ejecución es de 2,44 segundos, lo cual puede ser aceptable para una sola iteración. Sin embargo, para el punto 3 (implementación del algoritmo genético), donde se necesitará utilizar el algoritmo exhaustivamente, este tiempo puede ser demasiado largo. Por esta razón, se han seleccionado los parámetros en amarillo, ya que ofrecen un equilibrio entre velocidad y precisión.

Es importante tener en cuenta que este análisis se ha realizado para una lista de 10 productos. Para una lista de 5 productos, es de esperar que el valor de L pueda reducirse sin disminuir el porcentaje de aciertos, ya que las posibles combinaciones de 5 elementos son mucho menores que en una lista de 10.

Tinicial	Tfinal	L	% aciertos	Tiempo de ejecución para 1 iteración
50	1	10	96%	0.01 s
50	1	20	99%	0.02 s

Vemos que con un tiempo de cómputo de tan solo 0,02 s podemos obtener el resultado óptimo en el 99% de los casos. Finalmente, el valor de L utilizado será una ecuación lineal entre 20 y 300 en función de la cantidad de elementos en una orden, que será de 5 a 10 para el siguiente ejercicio.

Algoritmo Genético

Problemática a resolver

En este ejercicio, se debe optimizar la ubicación de los productos en el almacén basándose en listas históricas de pedidos realizados previamente. Se debe tomar en cuenta las siguientes consideraciones:

- El layout del almacén está fijo (tamaño y ubicación de pasillos y estanterías), solo debe determinarse la ubicación de los productos
- Cada orden incluye un conjunto de productos que deben ser despachados en su totalidad
- El picking comienza y termina en una bahía de carga, la cual tiene ciertas coordenadas en el almacén (por generalidad, puede considerarse la bahía de carga en cualquier borde del almacén)
- El "costo" del picking es proporcional a la distancia recorrida

Implementación del algoritmo

A modo demostrativo, la cantidad de individuos y la cantidad de listas fue acotada para disminuir los tiempos de cómputo. Esto dificulta la obtención de una solución óptima.

Creación de individuos

Se crearon 10 individuos, donde cada uno es un posible orden al azar de los productos en los espacios de almacén. Como se trabaja con 48 productos, cada individuo tendrá esta longitud (cantidad de genes).

Creación de listas históricas

Para simular la existencia de órdenes de productos que han sido solicitados del almacén, se crearon al azar 20 órdenes, donde cada una puede tener entre 5 y 10 de los 48 posibles productos. Este análisis se realiza a modo de ejemplo, ya que en la realidad las órdenes no se generan al azar sino que son solicitadas por los clientes o usuarios.

```
[28, 29, 6, 18, 32]
[11, 14, 30, 6, 47, 22, 7, 15]
[37, 23, 22, 40, 9, 42]
[27, 11, 25, 22, 13, 41]
```

Figura 8: Ejemplo de listas de pedidos.

Inicialización del Algoritmo

De forma predeterminada se estableció el Inicio arriba a la izquierda y el Final abajo a la derecha.

Puntuación de individuos

Usando el temple simulado con los parámetros preestablecidos anteriormente, se procede a aplicar el algoritmo de temple simulado a cada uno de los individuos para cada una de las listas. Para cada una de estas listas, se obtiene una puntuación de costo de camino total y el promedio de todas las listas genera la puntuación promedio de costo de camino para un individuo. Como es un problema de minimización, se busca que cada individuo tenga el menor valor posible.

Selección de padres

Para crear la siguiente generación de individuos, se debe cruzar a los de la generación actual. Se seleccionan dos padres al azar de la población actual utilizando el método de selección por ruleta, en el cual la probabilidad de selección de un individuo está en proporción directa a su puntaje de adaptación en relación con la suma de los puntajes de adaptación de todos los individuos, es decir, el individuo con mejor puntaje tendrá más posibilidades de ser elegido como padre. Cada pareja de padres genera una pareja de hijos.

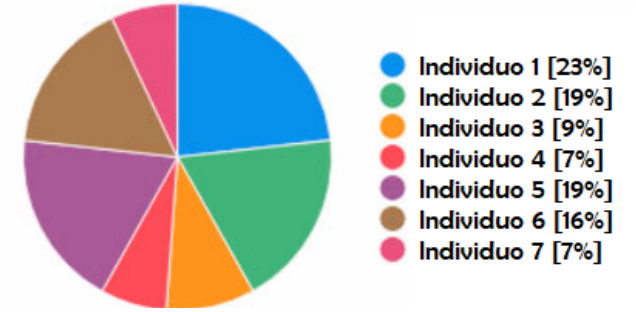


Figura 9: Método de selección por ruleta

Generación de hijos

Cada par de padres genera hijos usando la técnica de Ordered Crossover.

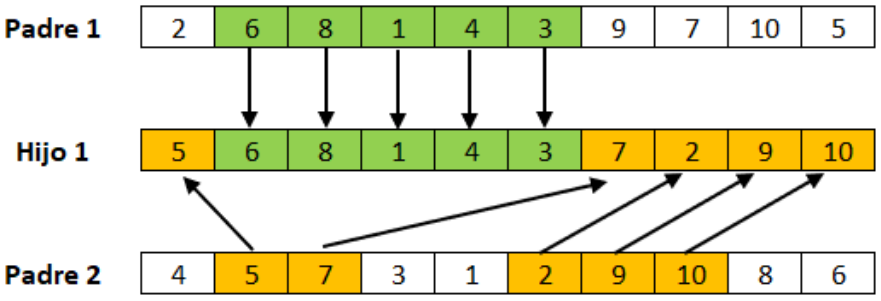


Figura 10: Ejemplo de Ordered Crossover

Del padre 1 se toma una sección de los genes tal cual los tiene él, respetando su orden y posición. Los espacios libres en el hijo 1 son rellenados con los elementos del padre 2 (respetando su orden) y teniendo en cuenta no repetir los elementos que fueron extraídos del padre 1.

Para el hijo 2 se realiza el proceso inverso, tomando una sección de los genes del padre 2, respetando su orden y posición, y rellenando los espacios libres con los elementos del padre 1 (respetando su orden) y sin repetir los elementos que ya estaban en el hijo 2 provenientes del padre 2. De esta forma, se generan dos hijos que combinan los genes de ambos padres, y se asegura que no haya duplicación de elementos en cada hijo.

Posibilidad de mutación

Cada hijo creado tiene un 5% de probabilidades de mutación, que consiste en intercambiar al azar 2 de sus elementos:

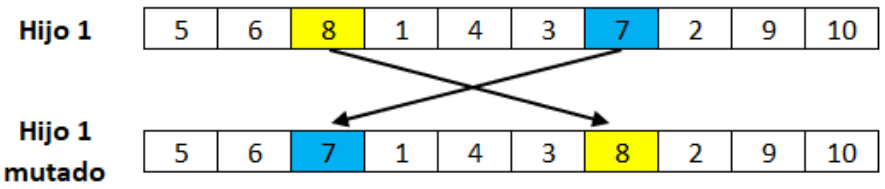


Figura 11: Ejemplo de mutación

Esto introduce una componente aleatoria en el programa, lo cual puede ser útil para generar soluciones que no hayan sido exploradas previamente.

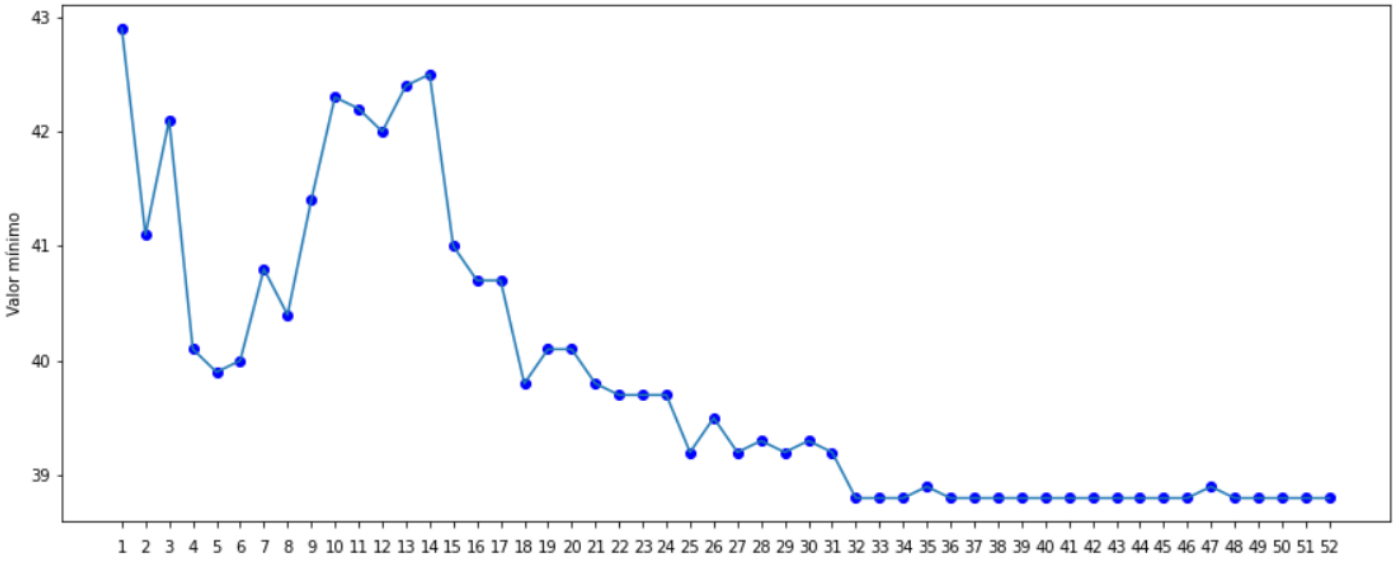
Criterio de parada

Una vez creada la nueva generación, se repite todo el proceso hasta que se cumpla con un criterio de parada que detenga la ejecución del programa. Se probaron varios criterios, como la diferencia con la generación anterior o el cálculo de la varianza de las últimas generaciones, pero se optó por otro enfoque. Se analizan todos los individuos de una población y se compara la puntuación del mejor individuo con la mejor puntuación global. Si esta puntuación global no se supera en las próximas n iteraciones (en este caso n=20), el algoritmo se detiene y devuelve el individuo que generó la mejor puntuación de forma global. Además, para evitar un exceso de tiempo de cómputo, se estableció un límite de 100 generaciones. Esto agrega una componente aleatoria al programa que puede ser útil para generar soluciones que no han sido exploradas previamente.

Resultados obtenidos

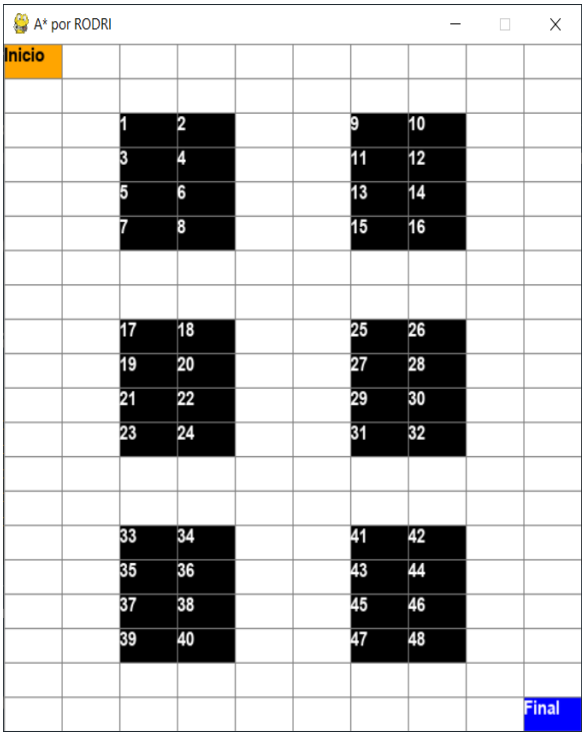
Cada individuo demora aproximadamente 2,3 s en ser analizado y en otorgarle una puntuación (para estas listas generadas en particular). Por lo tanto, una generación de 10 individuos es analizada en aproximadamente 23 s.

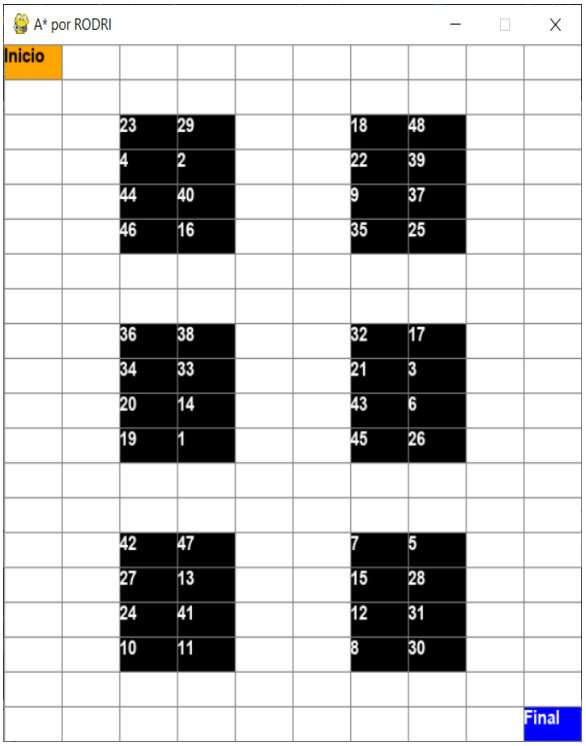
Luego de aproximadamente 24 minutos de ejecución, el algoritmo arrojó la siguiente gráfica:



De todas las generaciones se extrae la mejor puntuación global. Vemos como a partir de la generación 32 el algoritmo converge a una solución y la mantiene casi invariante por las próximas generaciones.

El almacén antes y después puede verse en las siguientes imágenes:

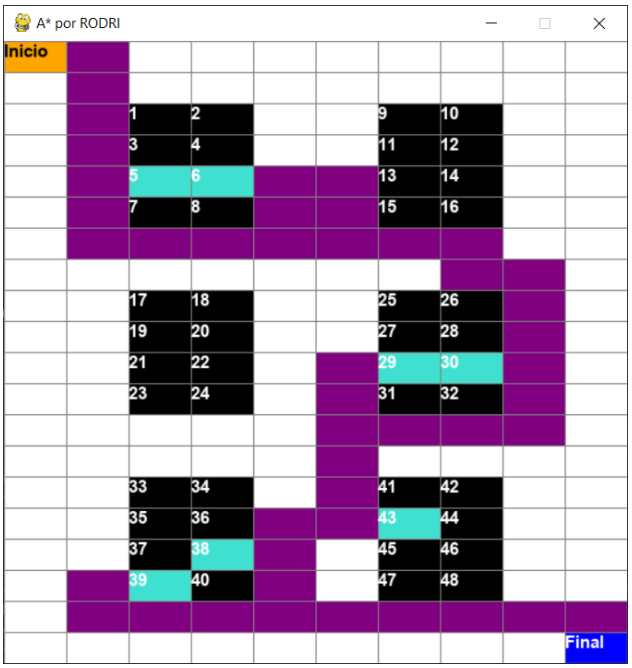




Se puede ver a continuación algunas comparaciones de listas entre ambos almacenes:

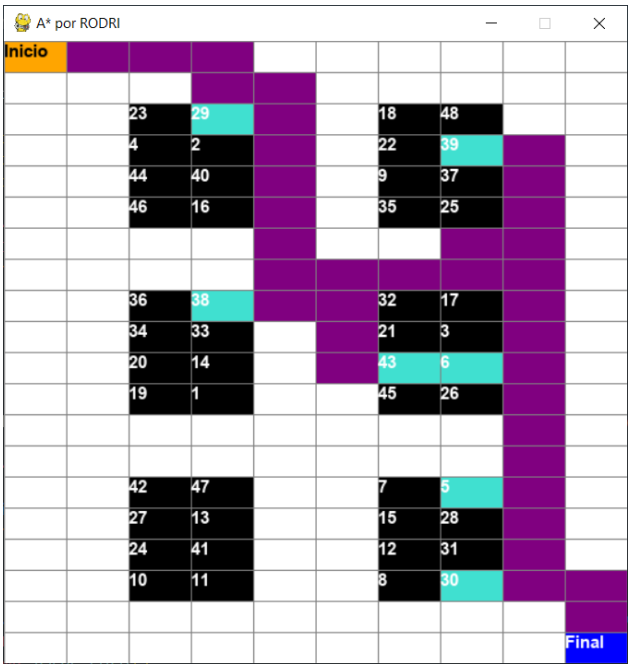
Orden 1

Antes



Coste = 52

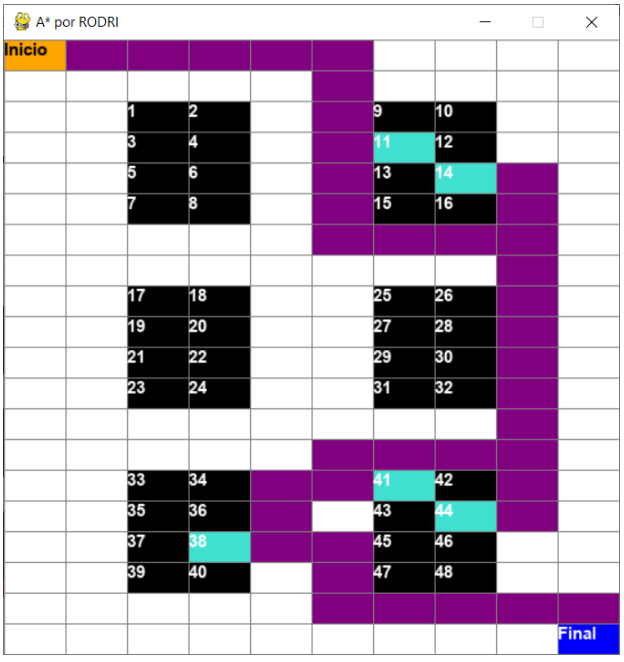
Después



Coste = 42

Orden 2

Antes



Coste = 44

Después

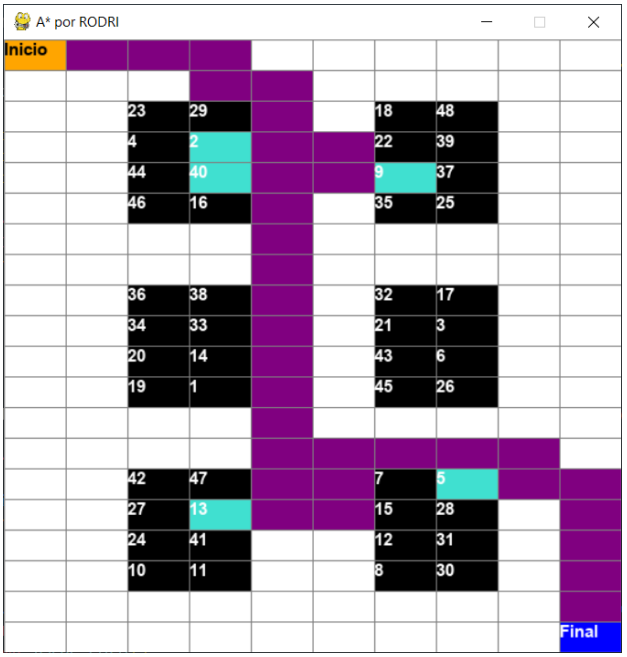
Coste = 28

Orden 3

Antes

Coste = 36

Después



Coste = 34

En líneas generales, el costo de camino mejora considerablemente del almacén anterior al posterior, pero por supuesto puede ocurrir que alguna lista de productos no se haya visto favorecida con el cambio.