BÚSQUEDA Y OPTIMIZACIÓN

INFORME DE PROYECTO

BENÍTEZ, Emanuel
BENÍTEZ, Lautaro

2022

UNIVERSIDAD NACIONAL DE CUYO
FACULTAD DE INGENIERIA

ÍNDICE

INTRODUCCIÓN

El proyecto propone abordar el desarrollo y entendimiento de los distintos algoritmos de búsqueda y optimización propuestos por la cátedra, entre ellos, el algoritmo A*, temple simulado y genético.

En principio se presenta la problemática de una industria, en particular la tarea de almacenamiento y despacho de sus productos. El objetivo del programa desarrollado es poder dar respuesta a tres cuestiones que se presentan al momento de ejecutar dichas tareas de la mejor manera posible, las cuales son ¿Qué camino tomar?, ¿En qué orden hacerlo? y ¿Cómo distribuir de manera correcta, los productos dentro del almacén? todo en base a minimizar el tiempo y/o costo de camino recorrido por el agente encargado de ejecutar la operación.

Una vez diseñado el escenario del almacén, el algoritmo A* se encargará de dar respuesta a la primera pregunta, ya que su estructura de funcionamiento nos permite resolver problemas de búsqueda de rutas, donde analiza el camino mas optimo en base a un criterio. Su búsqueda con información dada por la heurística aportará el mejor orden de exploración de los nodos en el grafo del almacén, llegando a minimizar el costo del camino recorrido.

En segunda instancia el algoritmo del temple simulado, en base a una orden dada, me dirá cual es el mejor picking de los productos, desde y hacia la bahía de carga, haciendo uso interno del algoritmo A* para evaluar una medida de rendimiento entre los distintos estados. A este algoritmo de búsqueda local solo le interesa llegar al estado final moviéndose entre estados vecinos de la orden, sin tener en cuenta el camino recorrido. Por lo tanto utilizará menos recursos de memoria y de manera constante durante su ejecución.

Es válido aclarar que las primeras preguntas deben responderse constantemente durante la ejecución de las tareas de almacenamiento, por ello debemos minimizar al máximo el tiempo de respuesta de nuestros algoritmos para tal fin. Sin embargo la última cuestión es saber, dado un historial de órdenes recibidas durante un periodo de tiempo, cómo reordenar los productos de las estanterías tal que aquellos que sean más demandados se encuentren cerca de la bahía de carga. Este análisis tiene un horizonte de tiempo largo, es decir que debe ser ejecutado cada tanto, ya que esto implica una modificación completa del layout del almacén generando que las líneas de producción se detengan. Con el uso del algoritmo genético, tendremos una búsqueda local con un cierto grado de estocacidad, que en el mejor de los casos con los operadores evolutivos adecuados, cada generación sucesiva estará más cerca de la solución, sino es que ya incluye al estado objetivo.

Para tales fines se buscó diseñar un software modular lo más versátil posible, como para poder generar distintos escenarios con el fin de obtener mejores conclusiones para cada algoritmo. Mediante la implementación de una consola de comandos de la clase **CMD** propia de python y la librería ***pygame,*** se consiguió generar un entorno amigable y entendible para el usuario con respecto a la utilización del software.

A continuación se presentan los parámetros del escenario en el que vamos a trabajar:

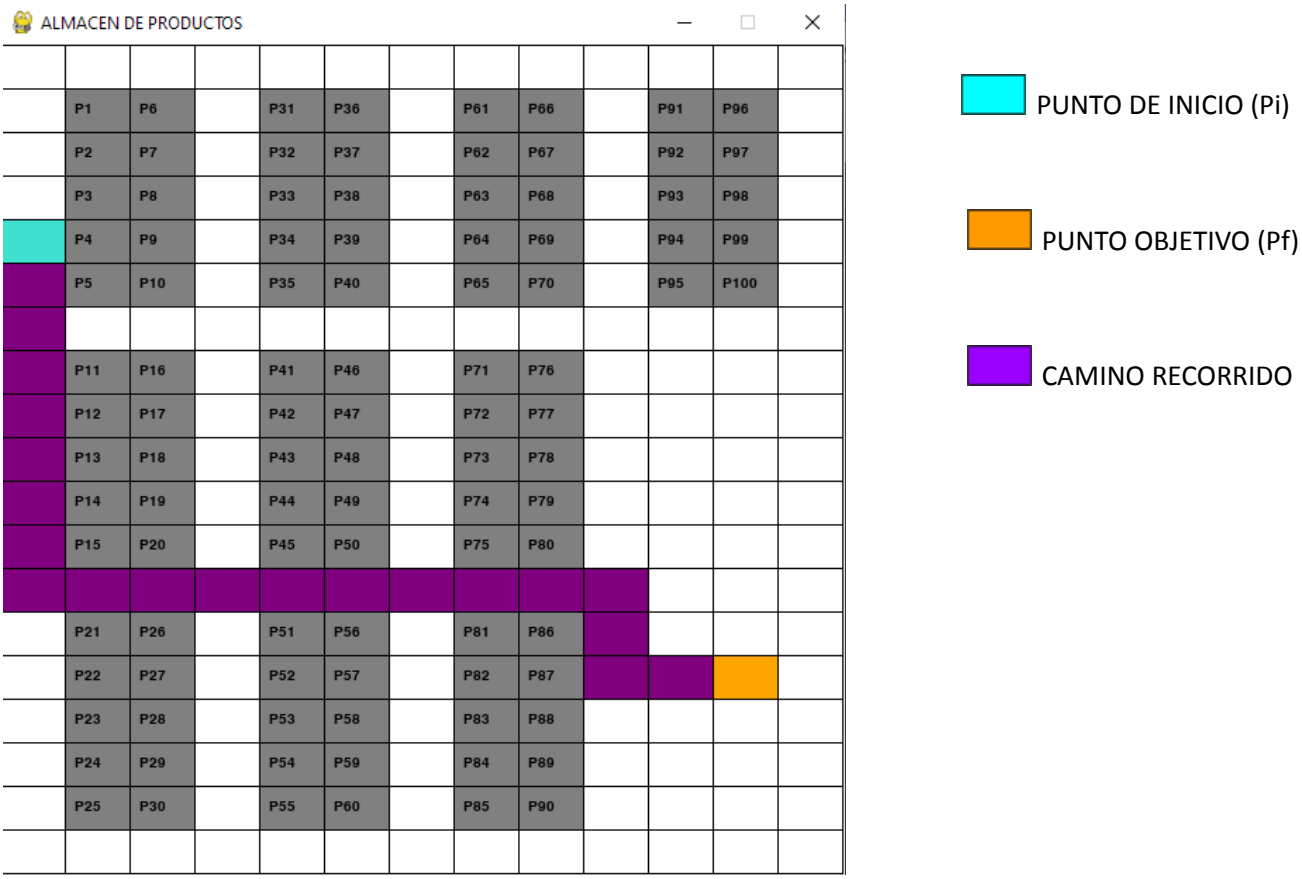
》Ingrese ancho de ventana en px: 600
》Ingrese alto de ventana en px: 600
》Ingrese la cantidad de estanterías a ubicar: 10
》Desea ubicarlas de manera horizontal (h) o vertical (v): v
》Indique la cantidad de productos por estantería (número par): 10
》Ingrese la cantidad de pasillos verticales: 5
》Ingrese la cantidad de pasillos horizontales: 4

ALGORITMO A*

La heurística utilizada en el algoritmo A*, para la medida entre el nodo actual y el nodo objetivo, es la distancia Euclidiana. Si bien el código permite optar por la variante de Manhattan, en base a sucesivas pruebas, se notó que particularmente al tener una distribución en el almacén con cierto grado de simetría entre estanterías, los resultados permanecen invariantes.

Prueba 1

Retorno >> El costo del Pi (4, 0) para llegar al Pf (14, 11) es: 21



ALGORITMO TEMPLE SIMULADO

Como ya se comentó anteriormente, este algoritmo no hace uso de una nueva heurística, sino que reutiliza el código A* para mensurar de alguna manera el estado actual de una orden y el de su vecino y a partir de ello, tomar una decisión de actualización de variable. Con lo cual, al iniciar el diseño del almacén, la consola pedirá al usuario ubicar la bahía de carga (en este caso al haber 100 productos, a la bahía de carga se le asignará el valor 101) y dará inicio al cálculo de distancias entre todos los puntos del escenario por completo, antes de correr el temple simulado.

A continuación se presentan los resultados y gráficos generados por el algoritmo, en base a la **orden N°4** del archivo “orders.txt”, para diferentes valores en los parámetros de entrada.

Prueba 1

22 iteraciones T=10 Tf=0.75

》El tiempo de cálculo fue: 0.0071 segundos

》EL PICKING MAS EFICIENTE PARA LA ORDEN, ES:

[**101**, 1, 8, 17, 20, 25, 73, 31, 36, 37, 41, 45, 50, 51, 52, 54, 30, 60, 64, 65, 56, 80, 87, 90, 92, 93, 96, 97, 98, **101**]

》CON UN COSTO MÍNIMO DE: **176**

Prueba 2

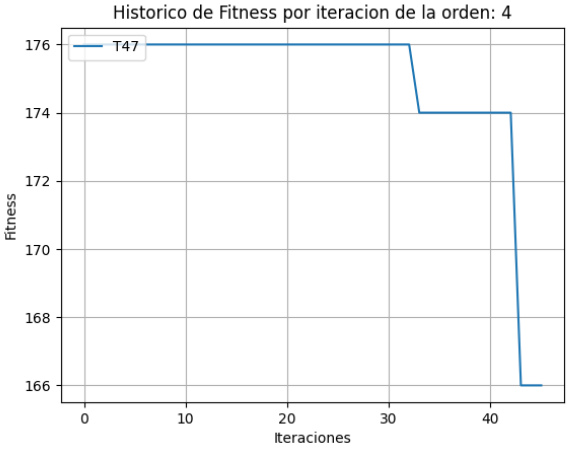
47 iteraciones T=10 Tf=0.6

》El tiempo de cálculo fue: 0.0079 segundos

》EL PICKING MAS EFICIENTE PARA LA ORDEN, ES:

[**101**, 1, 8, 17, 54, 25, 30, 37, 36, 31, 41, 20, 45, 51, 52, 50, 56, 60, 64, 65, 73, 90, 87, 80, 92, 93, 96, 97, 98, **101**]

》CON UN COSTO MÍNIMO DE: **166**



Prueba 3

100 iteraciones T=10 Tf=0.5

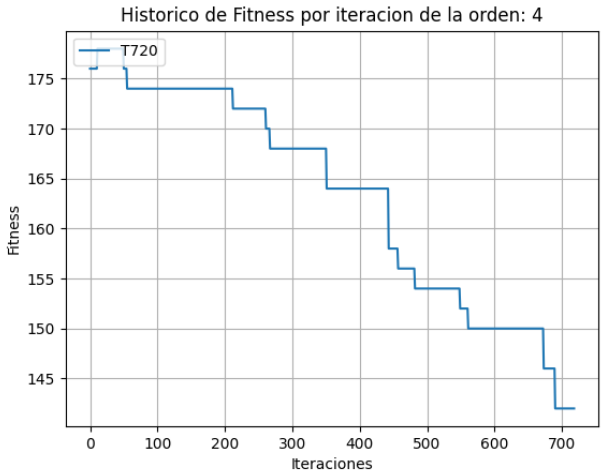
》El tiempo de cálculo fue: 0.007 segundos

》EL PICKING MAS EFICIENTE PARA LA ORDEN, ES:

[**101**, 17, 8, 31, 50, 25, 30, 1, 36, 37, 41, 45, 20, 51, 52, 54, 60, 56, 64, 65, 73, 80, 87, 90, 92, 93, 96, 97, 98, **101**]

》CON UN COSTO MÍNIMO DE: **166**

Prueba 4



720 iteraciones T=10 Tf=0.35

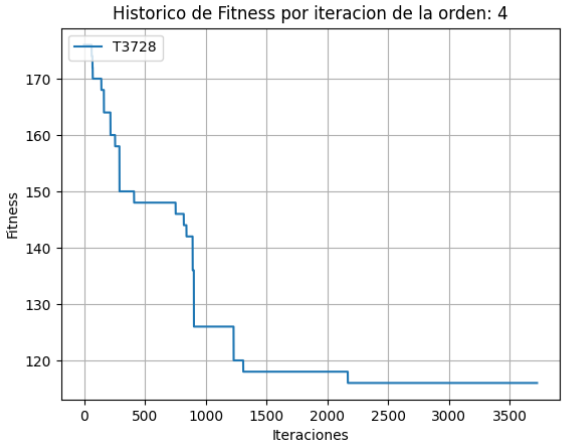
》El tiempo de cálculo fue: 0.0259 segundos

》EL PICKING MAS EFICIENTE PARA LA ORDEN, ES:

[**101**, 50, 45, 20, 17, 52, 56, 37, 36, 1, 31, 8, 41, 51, 54, 30, 25, 60, 73, 64, 65, 80, 87, 90, 93, 92, 96, 97, 98, **101**]

》CON UN COSTO MÍNIMO DE: **142**

Prueba 5



3728 iteraciones T=10 Tf=0.28

》El tiempo de cálculo fue: 0.087 segundos

》EL PICKING MAS EFICIENTE PARA LA ORDEN, ES:

[**101**, 90, 60, 25, 30, 54, 52, 8, 31, 1, 36, 37, 64, 65, 41, 17, 20, 45, 51, 50, 73, 56, 87, 80, 93, 92, 96, 97, 98, **101**]

》CON UN COSTO MÍNIMO DE: **116**

Prueba 7

50000 iteraciones (max iteraciones) T=10 Tf=0.20

》El tiempo de cálculo fue: 1.1782 segundos

》EL PICKING MAS EFICIENTE PARA LA ORDEN, ES:

[**101**, 45, 20, 17, 41, 8, 31, 1, 36, 37, 64, 65, 73, 25, 30, 54, 52, 51, 50, 56, 60, 90, 87, 80, 93, 92, 96, 97, 98, **101**]

》CON UN COSTO MÍNIMO DE: **112**

ALGORITMO GENÉTICO

Prueba 1

Temple con 3728 iteraciones de las órdenes, Tf=0.28

》Ingrese la cantidad de individuos por población: 6

》Ingrese la cantidad de órdenes a analizar (max 100): 100

》Ingrese la cantidad de generaciones que desea generar: 10

Fitness promedio de la población 0 : 95.0 Tiempo: 54.0 seg

Fitness promedio de la población 1 : 95.33 Tiempo: 68.07 seg

Fitness promedio de la población 2 : 95.5 Tiempo: 55.19 seg

Fitness promedio de la población 3 : 96.83 Tiempo: 58.5 seg

Fitness promedio de la población 4 : 95.17 Tiempo: 51.82 seg

Fitness promedio de la población 5 : 96.5 Tiempo: 54.56 seg

Fitness promedio de la población 6 : 96.0 Tiempo: 56.94 seg

Fitness promedio de la población 7 : 96.67 Tiempo: 59.16 seg

Fitness promedio de la población 8 : 94.83 Tiempo: 59.76 seg

Fitness promedio de la población 9 : 96.33 Tiempo: 51.22 seg

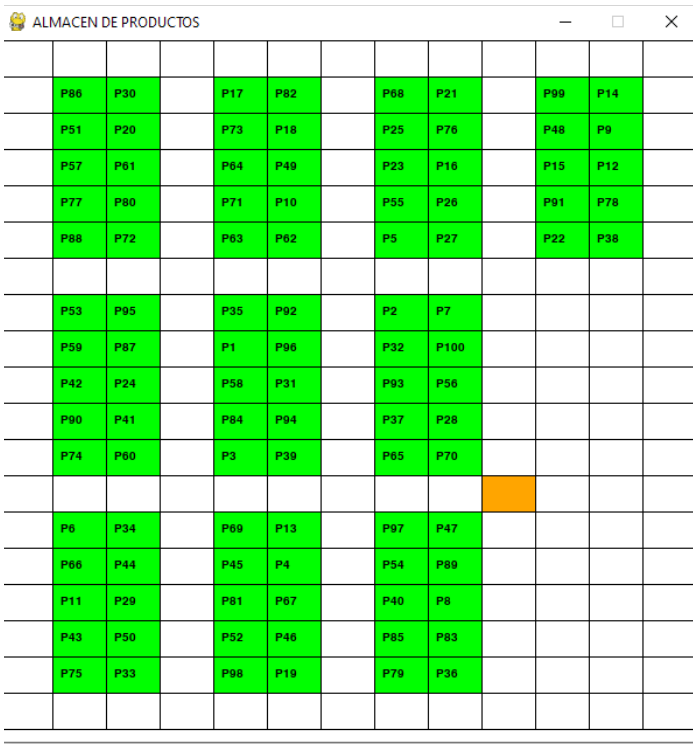
Fitness promedio de la población 10 : 96.17 Tiempo: 45.58 seg

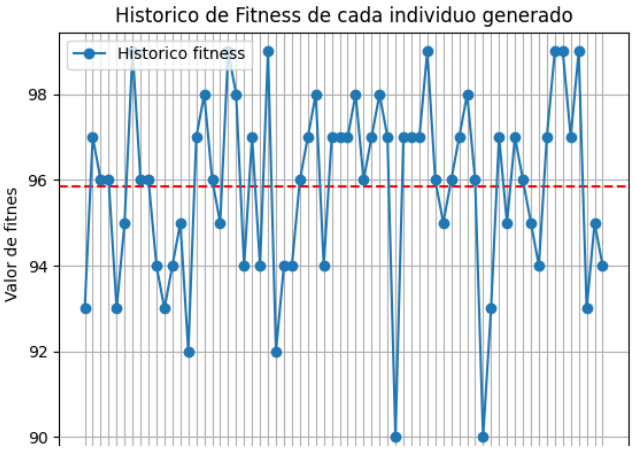
》LA DISTRIBUCIÓN MÁS ÓPTIMA ENCONTRADA, DE LOS PRODUCTOS EN EL ALMACÉN ES:

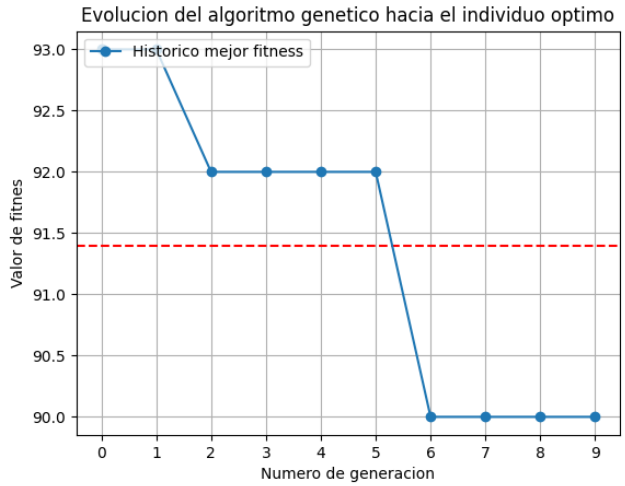
[86, 51, 57, 77, 88, 30, 20, 61, 80, 72, 53, 59, 42, 90, 74, 95, 87, 24, 41, 60, 6, 66, 11, 43, 75, 34, 44, 29, 50, 33, 17, 73, 64, 71, 63, 82, 18, 49, 10, 62, 35, 1, 58, 84, 3, 92, 96, 31, 94, 39, 69, 45, 81, 52, 98, 13, 4, 67, 46, 19, 68, 25, 23, 55, 5, 21, 76, 16, 26, 27, 2, 32, 93, 37, 65, 7, 100, 56, 28, 70, 97, 54, 40, 85, 79, 47, 89, 8, 83, 36, 99, 48, 15, 91, 22, 14, 9, 12, 78, 38]

》CON UN VALOR DE FITNESS DE: **90**

》EL ALGORITMO DEMORÓ EN TOTAL: 614.81 seg







Prueba 2

Temple con 3728 iteraciones de las órdenes tf=0.28

》Ingrese la cantidad de individuos por población: 6

》Ingrese la cantidad de órdenes a analizar (max 100): 100

》Ingrese la cantidad de generaciones que desea generar: 50

Fitness promedio de la población 0 : 95.17 Tiempo: 46.0 seg

Fitness promedio de la población 1 : 95.5 Tiempo: 58.39 seg

Fitness promedio de la población 2 : 96.33 Tiempo: 62.34 seg

Fitness promedio de la población 3 : 96.83 Tiempo: 60.25 seg

Fitness promedio de la población 4 : 95.5 Tiempo: 59.16 seg

Fitness promedio de la población 5 : 94.5 Tiempo: 58.97 seg

Fitness promedio de la población 6 : 94.5 Tiempo: 59.42 seg

Fitness promedio de la población 7 : 96.0 Tiempo: 60.31 seg

Fitness promedio de la población 8 : 96.5 Tiempo: 59.96 seg

Fitness promedio de la población 9 : 96.0 Tiempo: 60.39 seg

Fitness promedio de la población 10 : 95.83 Tiempo: 58.44 seg

Fitness promedio de la población 11 : 96.33 Tiempo: 59.19 seg

Fitness promedio de la población 12 : 96.67 Tiempo: 59.55 seg

Fitness promedio de la población 13 : 97.33 Tiempo: 58.86 seg

Fitness promedio de la población 14 : 96.67 Tiempo: 55.43 seg

Fitness promedio de la población 15 : 93.83 Tiempo: 46.21 seg

Fitness promedio de la población 16 : 95.5 Tiempo: 46.24 seg

Fitness promedio de la población 17 : 96.17 Tiempo: 46.5 seg

Fitness promedio de la población 18 : 95.0 Tiempo: 45.8 seg

Fitness promedio de la población 19 : 95.17 Tiempo: 46.62 seg

Fitness promedio de la población 20 : 95.83 Tiempo: 47.55 seg

Fitness promedio de la población 21 : 96.33 Tiempo: 53.37 seg

Fitness promedio de la población 22 : 95.67 Tiempo: 65.07 seg

Fitness promedio de la población 23 : 93.33 Tiempo: 71.19 seg

Fitness promedio de la población 24 : 94.67 Tiempo: 75.67 seg

Fitness promedio de la población 25 : 94.67 Tiempo: 74.43 seg

Fitness promedio de la población 26 : 94.83 Tiempo: 70.74 seg

Fitness promedio de la población 27 : 94.83 Tiempo: 67.73 seg

Fitness promedio de la población 28 : 96.17 Tiempo: 69.13 seg

Fitness promedio de la población 29 : 96.0 Tiempo: 69.12 seg

Fitness promedio de la población 30 : 94.33 Tiempo: 70.82 seg

Fitness promedio de la población 31 : 95.0 Tiempo: 98.42 seg

Fitness promedio de la población 32 : 95.33 Tiempo: 76.18 seg

Fitness promedio de la población 33 : 95.33 Tiempo: 63.14 seg

Fitness promedio de la población 34 : 95.17 Tiempo: 63.43 seg

Fitness promedio de la población 35 : 96.33 Tiempo: 62.74 seg

Fitness promedio de la población 36 : 97.0 Tiempo: 63.81 seg

Fitness promedio de la población 37 : 97.67 Tiempo: 62.81 seg

Fitness promedio de la población 38 : 96.33 Tiempo: 62.57 seg

Fitness promedio de la población 39 : 94.67 Tiempo: 62.7 seg

Fitness promedio de la población 40 : 96.0 Tiempo: 59.76 seg

Fitness promedio de la población 41 : 96.33 Tiempo: 55.23 seg

Fitness promedio de la población 42 : 95.33 Tiempo: 65.9 seg

Fitness promedio de la población 43 : 95.17 Tiempo: 67.77 seg

Fitness promedio de la población 44 : 96.0 Tiempo: 62.8 seg

Fitness promedio de la población 45 : 95.83 Tiempo: 60.86 seg

Fitness promedio de la población 46 : 96.17 Tiempo: 67.04 seg

Fitness promedio de la población 47 : 96.67 Tiempo: 77.91 seg

Fitness promedio de la población 48 : 96.83 Tiempo: 68.31 seg

Fitness promedio de la población 49 : 94.67 Tiempo: 67.75 seg

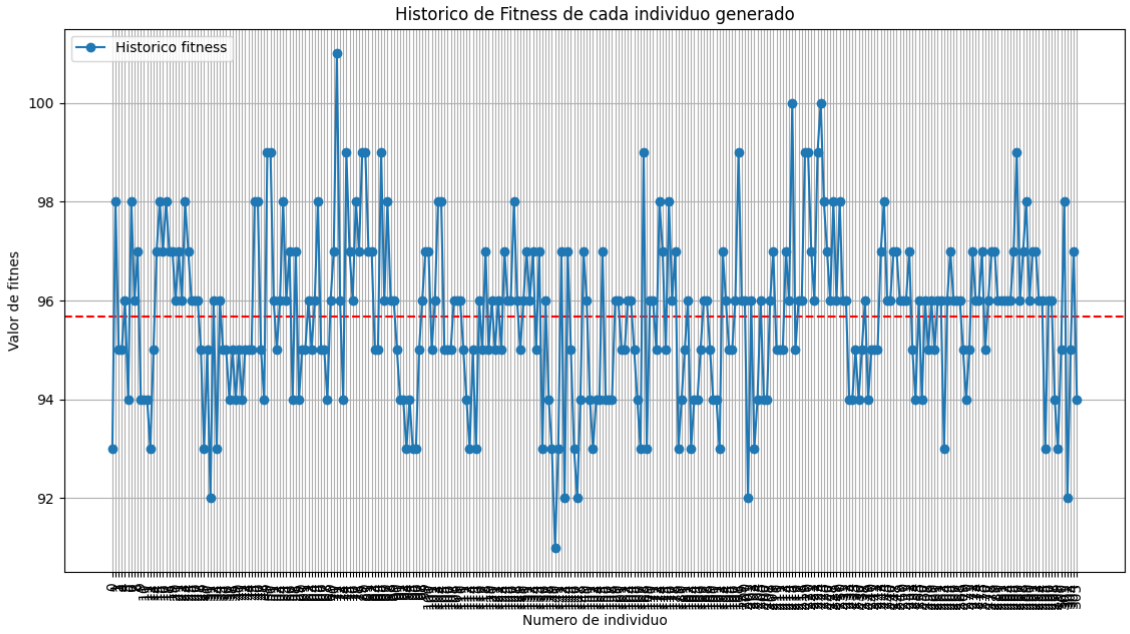
Fitness promedio de la población 50 : 95.17 Tiempo: 55.6 seg

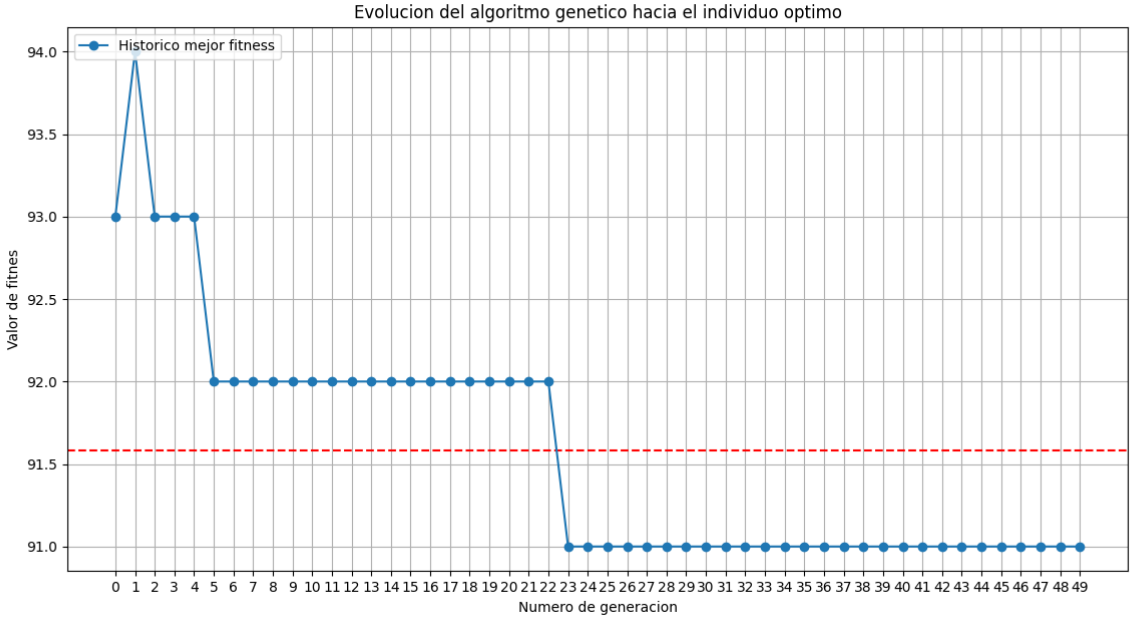
》LA DISTRIBUCIÓN MÁS ÓPTIMA ENCONTRADA, DE LOS PRODUCTOS EN EL ALMACÉN ES:

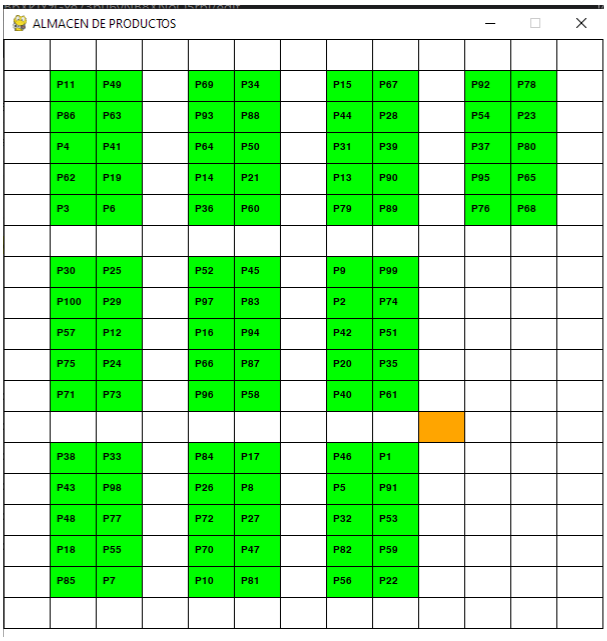
[11, 86, 4, 62, 3, 49, 63, 41, 19, 6, 30, 100, 57, 75, 71, 25, 29, 12, 24, 73, 38, 43, 48, 18, 85, 33, 98, 77, 55, 7, 69, 93, 64, 14, 36, 34, 88, 50, 21, 60, 52, 97, 16, 66, 96, 45, 83, 94, 87, 58, 84, 26, 72, 70, 10, 17, 8, 27, 47, 81, 15, 44, 31, 13, 79, 67, 28, 39, 90, 89, 9, 2, 42, 20, 40, 99, 74, 51, 35, 61, 46, 5, 32, 82, 56, 1, 91, 53, 59, 22, 92, 54, 37, 95, 76, 78, 23, 80, 65, 68]

》CON UN VALOR DE FITNESS DE: **91**

》EL ALGORITMO DEMORÓ EN TOTAL: 3167.61 seg


Historico de Fitness de cada individuo generado
Historico fitness
Valor de fitnes
Numero de individuo


Evolucion del algoritmo genetico hacia el individuo optimo
Historico mejor fitness
Valor de fitnes
Numero de generacion



Prueba 3

Temple con 3728 iteraciones de las órdenes tf=0.28

》Ingrese la cantidad de individuos por población: 50

》Ingrese la cantidad de órdenes a analizar (max 100): 100

》Ingrese la cantidad de generaciones que desea generar: 6

Fitness promedio de la población 0 : 95.3 Tiempo: 378.61 seg

Fitness promedio de la población 1 : 95.5 Tiempo: 394.02 seg

Fitness promedio de la población 2 : 96.42 Tiempo: 392.72 seg

Fitness promedio de la población 3 : 96.24 Tiempo: 381.59 seg

Fitness promedio de la población 4 : 95.72 Tiempo: 383.52 seg

Fitness promedio de la población 5 : 96.2 Tiempo: 383.16 seg

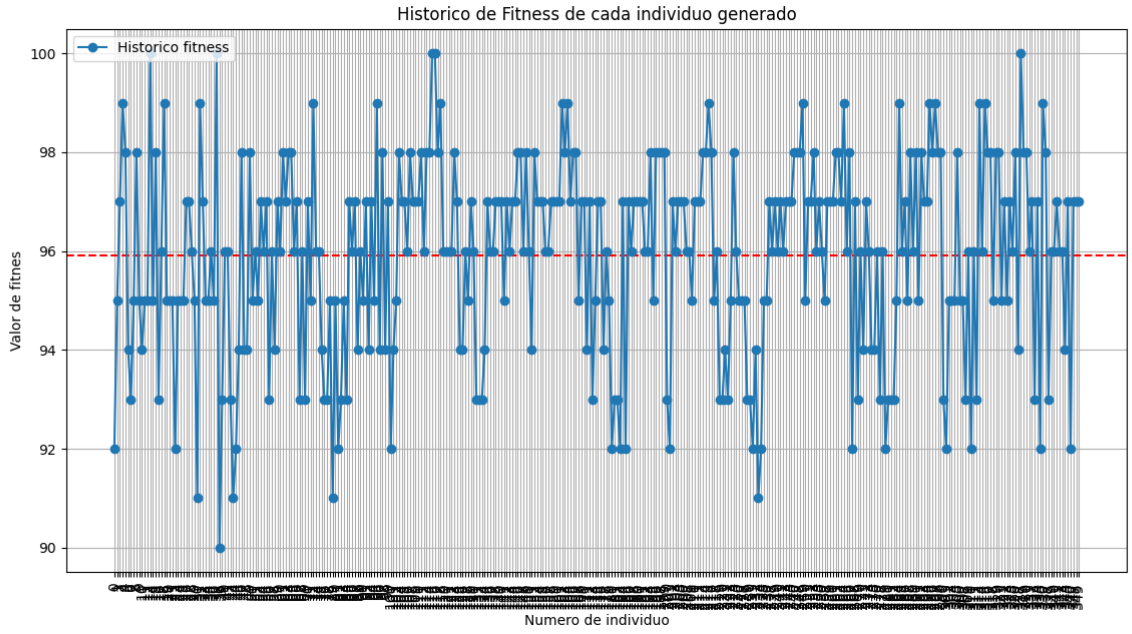
Fitness promedio de la población 6 : 95.94 Tiempo: 382.71 seg

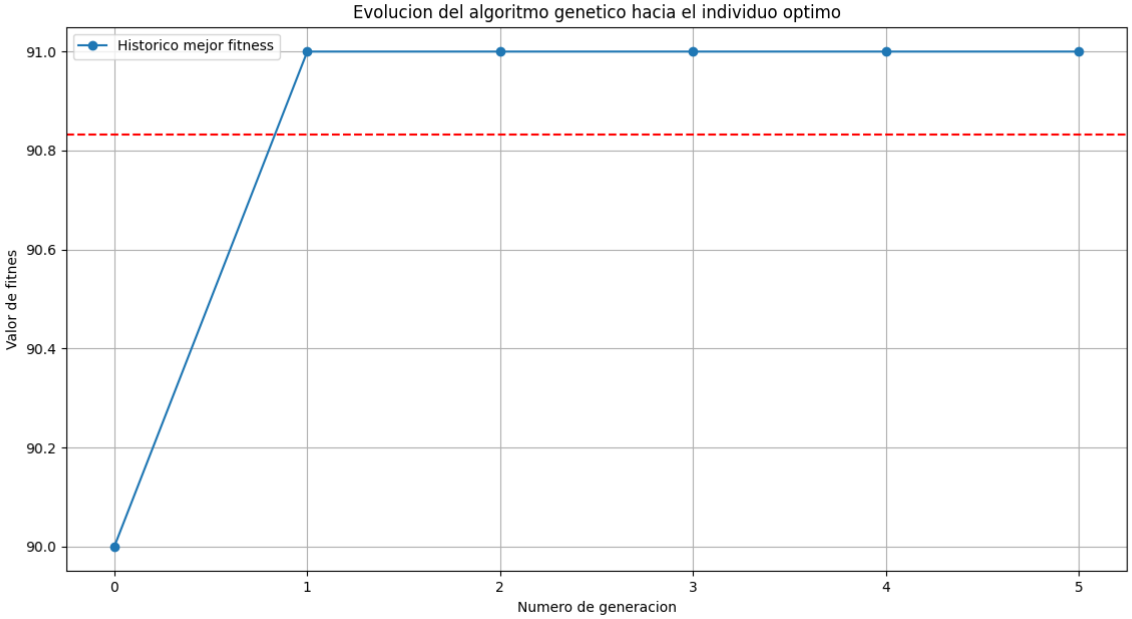
》LA DISTRIBUCIÓN MÁS ÓPTIMA ENCONTRADA, DE LOS PRODUCTOS EN EL ALMACÉN ES:

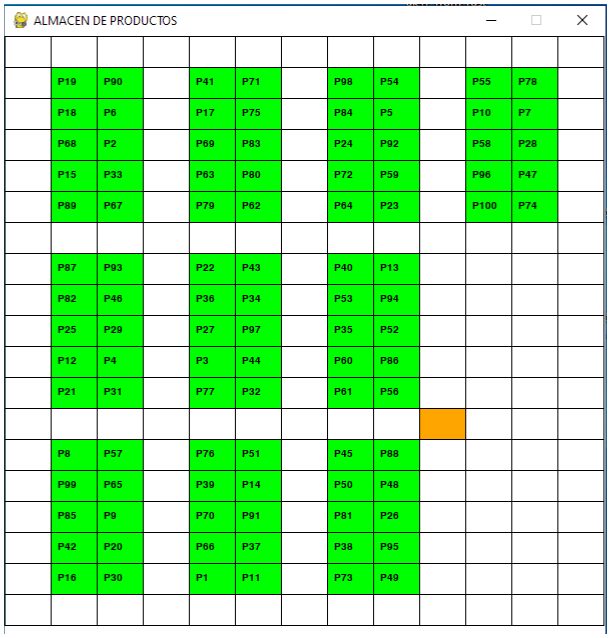
[19, 18, 68, 15, 89, 90, 6, 2, 33, 67, 87, 82, 25, 12, 21, 93, 46, 29, 4, 31, 8, 99, 85, 42, 16, 57, 65, 9, 20, 30, 41, 17, 69, 63, 79, 71, 75, 83, 80, 62, 22, 36, 27, 3, 77, 43, 34, 97, 44, 32, 76, 39, 70, 66, 1, 51, 14, 91, 37, 11, 98, 84, 24, 72, 64, 54, 5, 92, 59, 23, 40, 53, 35, 60, 61, 13, 94, 52, 86, 56, 45, 50, 81, 38, 73, 88, 48, 26, 95, 49, 55, 10, 58, 96, 100, 78, 7, 28, 47, 74]

》CON UN VALOR DE FITNESS DE: **91**

》EL ALGORITMO DEMORÓ EN TOTAL: 2696.34 seg


Historico de Fitness de cada individuo generado
Historico fitness
Valor de fitnes
Numero de individuo


Evolucion del algoritmo genetico hacia el individuo optimo
Historico mejor fitness
Valor de fitnes
Numero de generacion



Prueba 4

Temple con 3728 iteraciones de las órdenes

》Ingrese la cantidad de individuos por población: 6

》Ingrese la cantidad de órdenes a analizar (max 100): 100

》Ingrese la cantidad de generaciones que desea generar: 10

Fitness promedio de la población 0 : 99.17 Tiempo: 62.02 seg

Fitness promedio de la población 1 : 104.83 Tiempo: 64.14 seg

Fitness promedio de la población 2 : 103.0 Tiempo: 65.38 seg

Fitness promedio de la población 3 : 103.5 Tiempo: 65.15 seg

Fitness promedio de la población 4 : 101.5 Tiempo: 62.23 seg

Fitness promedio de la población 5 : 101.67 Tiempo: 59.65 seg

Fitness promedio de la población 6 : 104.17 Tiempo: 58.58 seg

Fitness promedio de la población 7 : 101.17 Tiempo: 61.44 seg

Fitness promedio de la población 8 : 102.0 Tiempo: 62.0 seg

Fitness promedio de la población 9 : 102.83 Tiempo: 61.61 seg

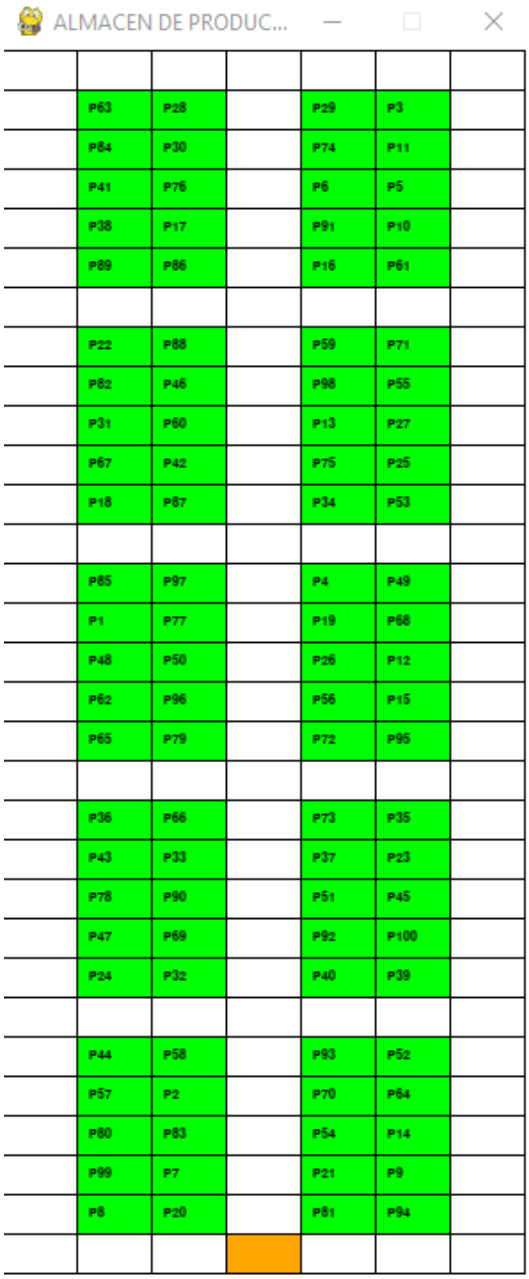
Fitness promedio de la población 10 : 104.17 Tiempo: 62.19 seg

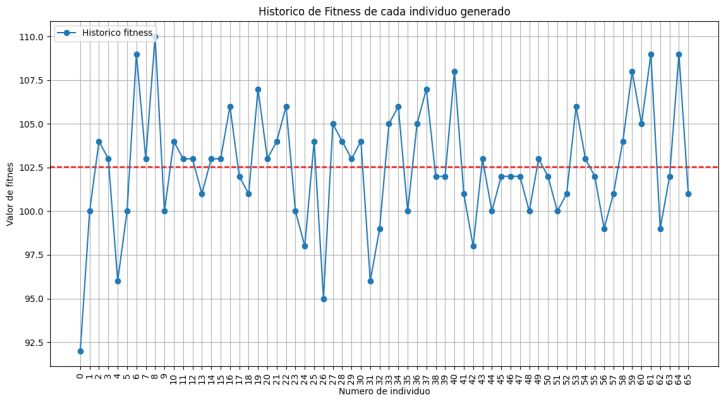
》LA DISTRIBUCIÓN MÁS ÓPTIMA ENCONTRADA, DE LOS PRODUCTOS EN EL ALMACÉN ES:

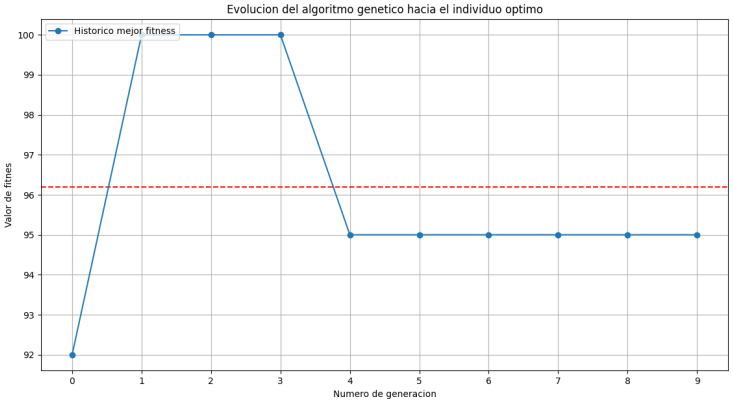
[63, 84, 41, 38, 89, 28, 30, 76, 17, 86, 22, 82, 31, 67, 18, 88, 46, 60, 42, 87, 85, 1, 48, 62, 65, 97, 77, 50, 96, 79, 36, 43, 78, 47, 24, 66, 33, 90, 69, 32, 44, 57, 80, 99, 8, 58, 2, 83, 7, 20, 29, 74, 6, 91, 16, 3, 11, 5, 10, 61, 59, 98, 13, 75, 34, 71, 55, 27, 25, 53, 4, 19, 26, 56, 72, 49, 68, 12, 15, 95, 73, 37, 51, 92, 40, 35, 23, 45, 100, 39, 93, 70, 54, 21, 81, 52, 64, 14, 9, 94]

》CON UN VALOR DE FITNESS DE: **95**

》EL ALGORITMO DEMORÓ EN TOTAL: 684.38 seg







CONCLUSIÓN

Gracias a este informe se logró tener una buena práctica de los algoritmos A*, temple simulado y genético, aunque sobre la marcha se tuvieron que adquirir conocimientos extras sobre métodos para poder visualizar el comportamiento de los mismos y que el aprendizaje sea más eficiente. En el grupo se consideró conveniente la utilización de una interfaz gráfica, entre uno de los métodos, para poder generar un programa más robusto que permitiera modelar diferentes escenarios de forma rápida y sencilla lo que trajo aparejado un costo elevado en tiempo de programación, pruebas y análisis de errores aunque permitio poder visualizar de forma muy sencilla el comportamiento de cada uno de los algoritmos. También, fue conveniente confeccionar una consola de comandos que le permitiera al operario poder configurar sobre la marcha distintos valores de parámetros, que si bien en cada intento se necesitaban unos segundos más para ello, a la larga terminó siendo mucho más eficiente para llegar a mejores conclusiones.

Para asegurar la optimalidad y la completitud en el algoritmo A* se le sumó a la función de costo una función heurística euclidiana (distancia diagonal entre dos puntos) la cual garantiza una suma más precisa al tener valores decimales comparables sin tener que ver únicamente los valores enteros, que en algunos casos pueden coincidir.

Al trabajar con el algoritmo de temple simulado, el grupo se encontró con algunas complicaciones de comprensión del algoritmo, ya que se aprendieron varios conceptos que por su nombre generaron entendimientos ambiguos, como lo fueron: “vecinos” e Iteraciones y por otro lado la aplicación del método con dos posibles formas. En una se itera tantas veces la orden como la función de temperatura esté graduada desde una temperatura inicial a una temperatura final, y en contraparte, a esta sumarle en cada iteración, iteraciones intermedias para que generan variaciones del individuo en cada temperatura. Por cuestiones prácticas y para reducir el tiempo de cálculo, se optó por aplicar la primera opción con la que obtuvimos muy buenos tiempos de cálculo en el orden de los 0,08s y costos de picking razonablemente bajos para aquellas órdenes que contenían la mayor cantidad de productos.

El algoritmo genético es el que mayor maduración requirió y para el cual se invirtió más tiempo debido a que para el funcionamiento de este se necesitaba la mayor optimización de los algoritmos anteriores para su cálculo. Una vez garantizada la optimización de los algoritmos se pudieron evidenciar las siguientes conclusiones:

Con la tabla siguiente se pudo ver como un aumento en la cantidad de individuos por generación no genera mejor valor de picking, sin embargo el tiempo de cálculo es cada vez mayor.

PRUEBAS DEL ALGORITMO GENETICO (individuos variable)				
Numero de prueba	1	2	3	4
Combinaciones de temple	3728	3728	3728	3728
Individuos	4	6	10	20
Ordenes	100	100	100	100
Generaciones	5	5	5	5
Tiempo total	184.64	302.83	538.74	882.38
Costo	93	91	91	91

En el siguiente ejemplo se ha variado el valor en la cantidad de generaciones a calcular por el algoritmo y se pudo concluir que al igual que en las pruebas anteriores el valor aumenta considerablemente pero no se evidencia un menor costo.

PRUEBAS DEL ALGORITMO GENETICO (generaciones variable)				
Numero de prueba	1	2	3	4
Combinaciones de temple	3728	3728	3728	3728
Individuos	4	4	4	4
Ordenes	100	100	100	100
Generaciones	4	6	10	20
Tiempo total	137.77	221.77	328.93	623.93
Costo	93	93	92	92

Finalmente dejando seteados los mejores valores de las experiencias anteriores se decidió variar el valor de la cantidad de iteraciones en el temple, y en este caso, si bien el tiempo aumenta considerablemente se logra obtener un costo de picking mucho más bajo que en todas las pruebas anteriores, con lo cual se evidencia la relevancia de un buen temple en el algoritmo genético.

PRUEBAS DEL ALGORITMO GENETICO (combinaciones variable)				
Numero de prueba	1	2	3	4
Combinaciones de temple	100	720	3728	50000
Individuos	4	4	4	4
Ordenes	100	100	100	100
Generaciones	10	10	10	10
Tiempo total	13.61	63.3	311.43	924.42
Costo	106	98	91	83

Si analizamos la variación en el tiempo con respecto al picking en las últimas dos pruebas (3 y 4) de la última variación podemos ver que T4-T3=612,69 osea un 66% de tiempo más garantizando un picking del 9% mejor. En síntesis un aumento considerable en las iteraciones del temple no consiguen una disminución considerable en el picking con lo cual podríamos decir que una buena combinación de valores para tener un buen costo-beneficio podría ser:

PRUEBA FINAL ALGORITMO GENETICO	
Numero de prueba	1
Combinaciones de temple	3728
Individuos	6
Ordenes	100
Generaciones	10
Tiempo total	471.46
Costo	84

De esta forma conseguimos un muy buen costo en un tiempo mucho menor al de la prueba anterior (4). Cabe destacar que en este algoritmo se podría haber incluido la mutación y las pruebas con otros tipos de crossover para analizar las posibles mejoras.