Implementación de un algoritmo genético para la optimización del picking en un almacén

Resumen

Un problema de optimización se presenta cuando se diseña el layout de un almacén. La ubicación de los productos debe ser tal que cuando llegue una orden, la distancia recorrida sea la mínima. Para resolver este problema se utilizó un algoritmo genético, con el cual se logró obtener el layout mas óptimo en función de varias órdenes. A su vez se analizaron diferentes pruebas donde se modificaron los hiperparámetros del algoritmo y así llegar a conclusiones sobre estos.

Introducción

Uno de los problemas mas comunes que se presenta al diseñar el layout de un almacén es tratar de colocar los productos en las estanterías de modo que, cuando llegue una orden, la distancia recorrida para recoger todos los productos sea la menor posible.
En la siguiente sección se detalla como el problema de ubicar los productos de la mejor manera posible, trae aparejado otro problema mas: que secuencia seguir para recoger los artículos según el layout planteado. Este problema se conoce como el problema del viajero.
Para resolver estos dos problemas, se utilizó un algoritmo genético, donde para la función de idoneidad se utilizó un algoritmo de búsqueda avara. Luego, se sometió a este algoritmo a varias pruebas haciendo un barrido de los hiperparámetros. El resto del trabajo se centra en analizar como el fitness del almacén varía según la modificación de los hiperparámetros. Se exponen al final conclusiones sobre los resultados obtenidos en las múltiples pruebas, así como futuras extensiones de este trabajo.

Descripción del problema

Diseñar el layout de un almacén es una tarea complicada. Se debe tener en cuenta la cantidad de productos a colocar, cómo se los va a clasificar, cuántas estanterías se van a utilizar, que tamaño estas últimas van a tener, etc. Por esto, muchos de estos parámetros se han fijado en este trabajo como se ve en la Fig. 1 .

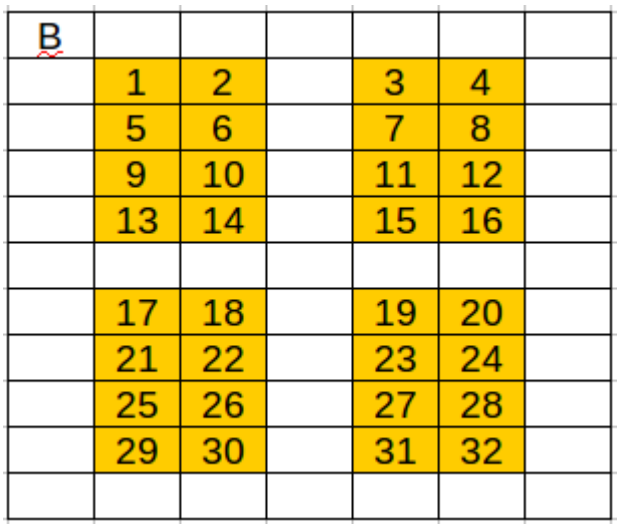

Fig.1

Se utilizó un almacén en el cual se van a colocar 32 tipos de productos distintos, a los cuales se les asigna un número del 1 al 32 según el tipo. El almacén cuenta con 4 estanterías, donde en cada estantería se colocan 8 tipos de productos. Vale aclarar que se ha considerado que cada estante con un tipo de producto puede almacenar una cantidad ilimitada de ese producto. La bahía de carga del almacén (lugar donde comienza y termina el picking) se simboliza con la letra B.

A este almacén llegan órdenes de productos, las cuales pueden contener cualquier tipo de producto, y a su vez varios productos del mismo tipo. También, estas órdenes, pueden tener cantidades de productos distintas entre sí, es decir, algunas órdenes van a ser mas cortas o mas largas que otras. A continuación se muestran ejemplos de distintas órdenes posibles.

Orden 1 [18, 7, 32, 24, 30, 12, 16]
Orden 2 [14, 9, 5, 32, 9]
Orden 3 [29, 29, 1, 15, 6, 13, 30, 9, 18]
Orden 4 [13, 9, 26, 20, 31, 11, 16, 27, 9, 20, 30, 20, 28]
Orden 5 [32, 1, 1, 1, 10, 24, 10, 16]

El problema yace entonces en encontrar la mejor ubicación para los 32 tipos de productos de modo que cuando se realice el picking de una orden, la distancia a recorrer sea la menor posible.
En el caso donde solo tengamos una orden, el problema se hace mas sencillo, ya que, por ejemplo, podríamos colocar aquellos productos que más se repiten cercanos a la bahía. Pero, como se mencionó en el párrafo anterior, tenemos N órdenes, las cuales contienen, no solo distintos productos, si no también distintas longitudes. Esto hace que el layout del almacén este fuertemente ligado a las ordenes de los productos con los que se va a trabajar. Es decir, que se debe encontrar un layout de almacén, para el cual, la distancia promedio a recorrer por orden, sea la mínima.
Debido a esto se trabaja con una cantidad N de órdenes fijas, la cual se ha tomado como hiperparámetro, y mas adelante se lo varía para analizar su efecto sobre el almacén.

Al tratar de optimizar el problema del picking para este almacén, surge otro problema: no sólo se deben colocar los productos en una cierta posición, sino que además, el orden con el cual se realiza el picking de los productos debe también ser optimizado. En otras palabras, para un layout de almacén explicitado, se debe encontrar la secuencia para recoger los productos de una orden de modo tal que la distancia sea la menor posible. Este problema se conoce como “el problema del agente viajero”, el cual tiene una complejidad NP-Hard, y para resolverlo se optó por implementar una búsqueda avara, detallada en la sección siguiente.

Diseño del algoritmo genético

El primer componente a diseñar de este algoritmo es el Genoma/Cromosoma, el cual será el modelo de los individuos, siendo estos últimos posibles soluciones al problema. El Genoma utilizado es el siguiente:

[32, 4, 5, 2, 9, 11, 14, 17, 24, 21, 8, 19, 16, 26, 25, 3, 18, 28, 31, 13, 1, 20, 12, 29, 6, 15, 27, 10, 22, 7, 30, 23]

donde cada gen representa un tipo de producto, y el índice del arreglo corresponde a la ubicación de dicho producto en el almacén de la Fig 1. Por ejemplo, en este caso, el

producto tipo “32” irá a parar en la ubicación con el número 1 de la Fig 1., el producto tipo “4” se ubicará en el número 2, el tipo “5” en la posición 3, y así sucesivamente.

El tamaño de la población se predefine antes de correr el programa, y es un hiperparámetro, el cual en la sección siguiente, se variará para ver como afecta al almacén.

La Figura 2 muestra el diagrama de flujo del algoritmo genético usado, junto con las partes que lo conforman

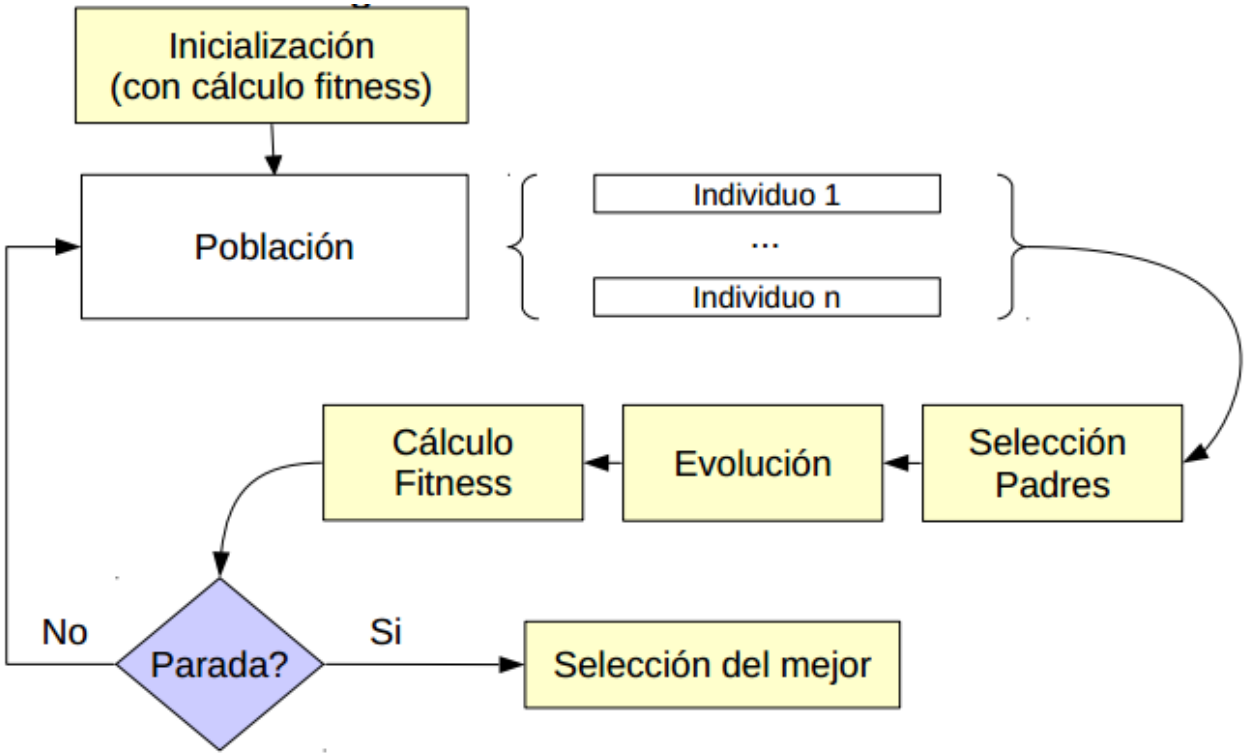

Fig.2

Para la selección de los padres, se utiliza la selección por ranking, donde se seleccionan a los N/2 mejores individuos, siendo N el tamaño de la población. Estos mejores padres, pasan a la siguiente generación, la cual esta conformada por dichos padres, y los N/2 hijos que surgan de aplicar procedimientos evolutivos a estos padres (mutación y crossover).

Para la mutación, el proceso es sencillo, se eligen dos genes al azar, y se los intercambia de posición. Esto asegura que seguimos teniendo los 32 tipos de productos, y que ningún tipo se repite.

Para el proceso de crossover, se utiliza el “cruce de orden”, el cual preserva un orden relativo de los genes de los padres. El procedimiento consiste en seleccionar dos puntos de cruce al azar, se copian los elementos del padre 1 entre los puntos de cruce y luego se copian los valores restantes a partir del segundo corte, en orden, evitando duplicar valores. El segundo hijo es creado de manera análoga el primero

Para calcular el fitness de cada individuo, como función de idoneidad se utiliza un algoritmo de búsqueda avara. Este algoritmo comienza en la posición de la bahía, y de ahí calcula, usando la distancia de Manhatan, cual es el producto (de todos los productos

de una orden dada) que minimiza dicha distancia. Luego, realiza el mismo procedimiento pero ahora desde la posición que había elegido anteriormente, y así continúa hasta que ya no tiene mas productos en la orden. Este algoritmo, entonces, nos devuelve la distancia mínima que se debe recorrer para realizar el picking de una orden para un layout de almacén dado. Este procedimiento se realiza a todas las órdenes, y luego se realiza un promedio con las distancias de todas ellas. Por lo tanto, como fitness de un individuo, se obtiene la distancia mínima promedio por orden según el layout dado.
Es importante resaltar dos puntos; primero, el algoritmo de búsqueda avara, es un algoritmo de búsqueda local, con lo cual la distancia obtenida es un sub-óptimo. Segundo, este algoritmo no contempla las restricciones del movimiento en el almacén, con lo cual el problema que se resuelve es una “versión relajada del problema”.

Por último, para la condición de parada, se eligió el numero de iteraciones, el cual, en la sección siguiente se modifica en las diferentes pruebas para obtener diferentes resultados.

Pruebas y Resultados

Se realizaron distintas pruebas usando el almacén de la Fig.1. En todas las pruebas, las órdenes fueron las mismas, excepto en aquellas donde la intención era modificar la cantidad de órdenes.
Estas pruebas se dividieron en 3 grupos.

En el primer grupo se utilizó una población de 1000 individuos, 20 órdenes, y lo que se varió fueron la cantidad de iteraciones que ejecutó el programa. Las Figuras 3,4 y 5 muestran el comportamiento. La Tabla 1 muestra los valores obtenidos.

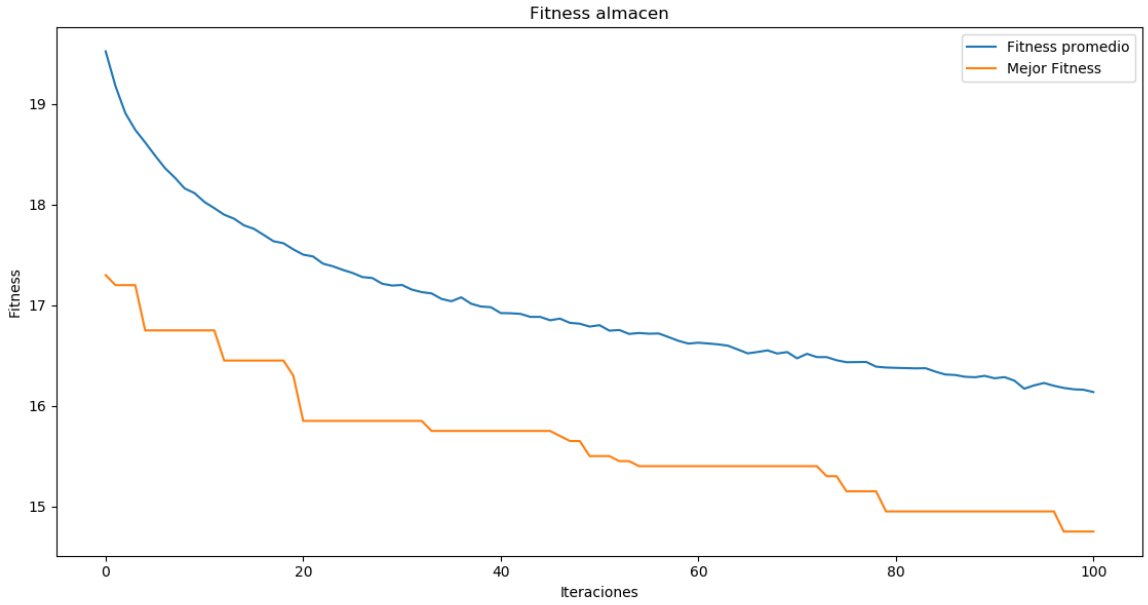

Fig. 3

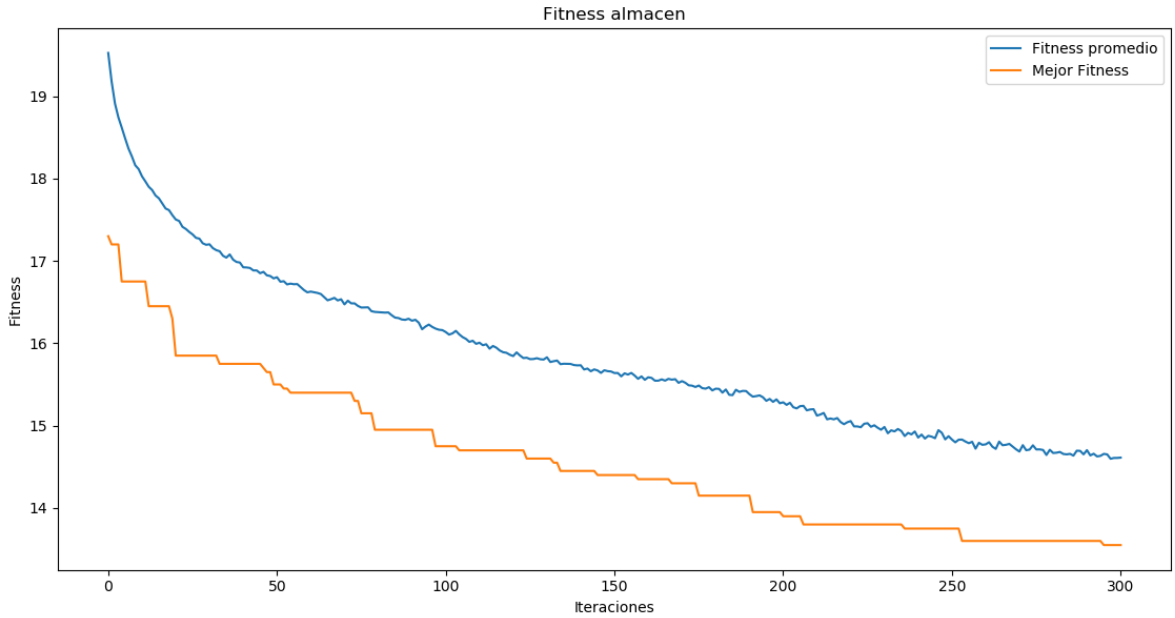

Fig. 4

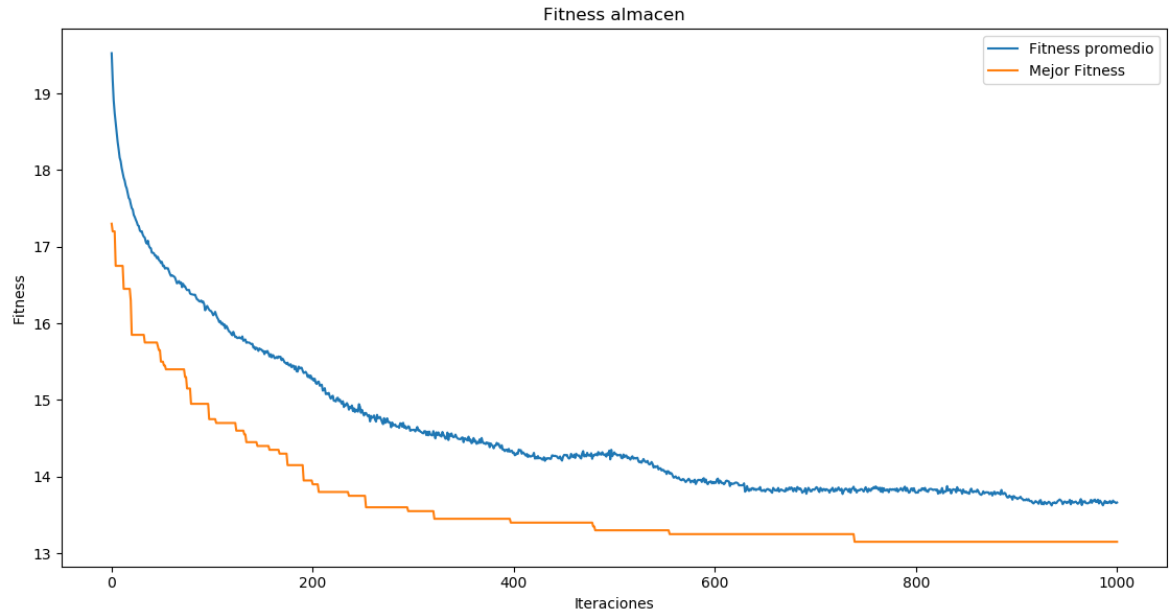

Fig. 5

Iteraciones	Tiempo de Ejecución	Mejor Fitness	Mejor Individuo
100	10 min.	14,75	[32 4 5 2 9 11 14 17 24 21 8 19 16 26 25 3 18 28 31 13 1 20 12 29 6 15 27 10 22 7 30 23]
300	30 min	13,55	[32 31 5 2 16 20 4 14 24 21 8 19 9 26 25 3 12 13 29 11 1 18 30 7 10 23 27 28 17 22 6 15]
1000	1 hora y 20 min	13,15	[20 31 4 17 16 32 5 14 9 24 3 28 26 21 19 8 13 18 29 11 1 12 30 25 10 23 27 15 2 22 7 6]

Tabla 1

En primer lugar se puede notar como el algoritmo, en todas las pruebas, logra mejorar el fitness un numero considerable, es decir que ,a medida que va iterando, la distancia promedio por orden (esto es el fitness) va disminuyendo, hasta converger en un valor.
En segundo lugar, la Tabla 1 nos muestra que, a medida que aumentamos las iteraciones, obtenemos mejores valores del fitness, pero esta mejora es pequeña en comparación con la cantidad de tiempo necesario para lograrla.

El segundo grupo de pruebas consistió en utilizar 1000 iteraciones, 20 órdenes y variar la cantidad de individuos que conforman la población. Las Figuras 6 y 7 muestran este comportamiento. La Tabla 2 muestra los valores obtenidos.

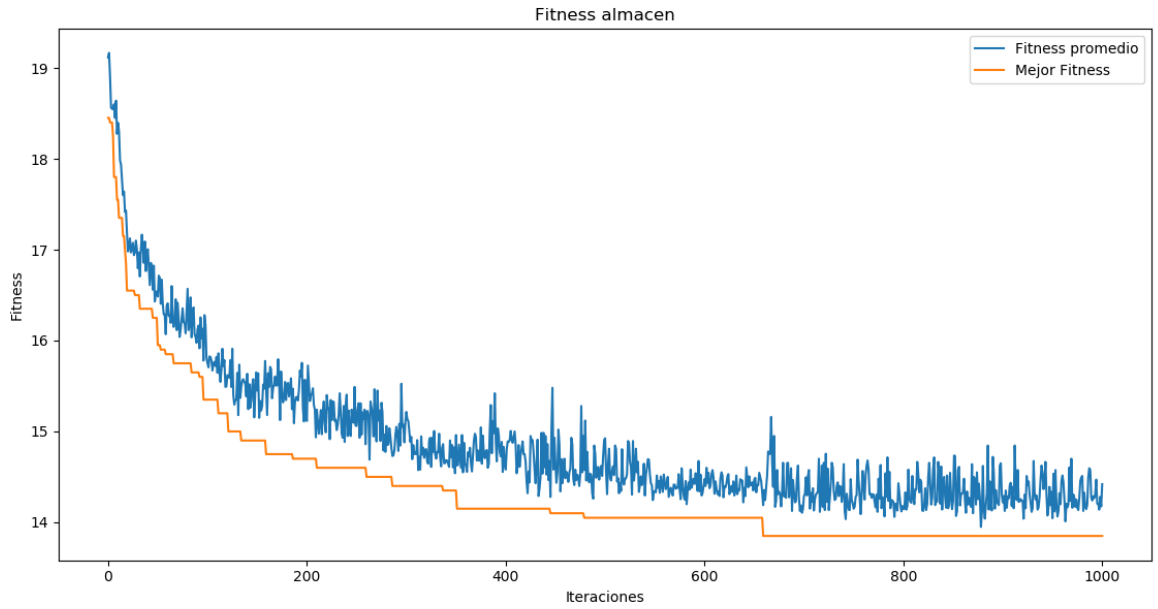

Fig. 6

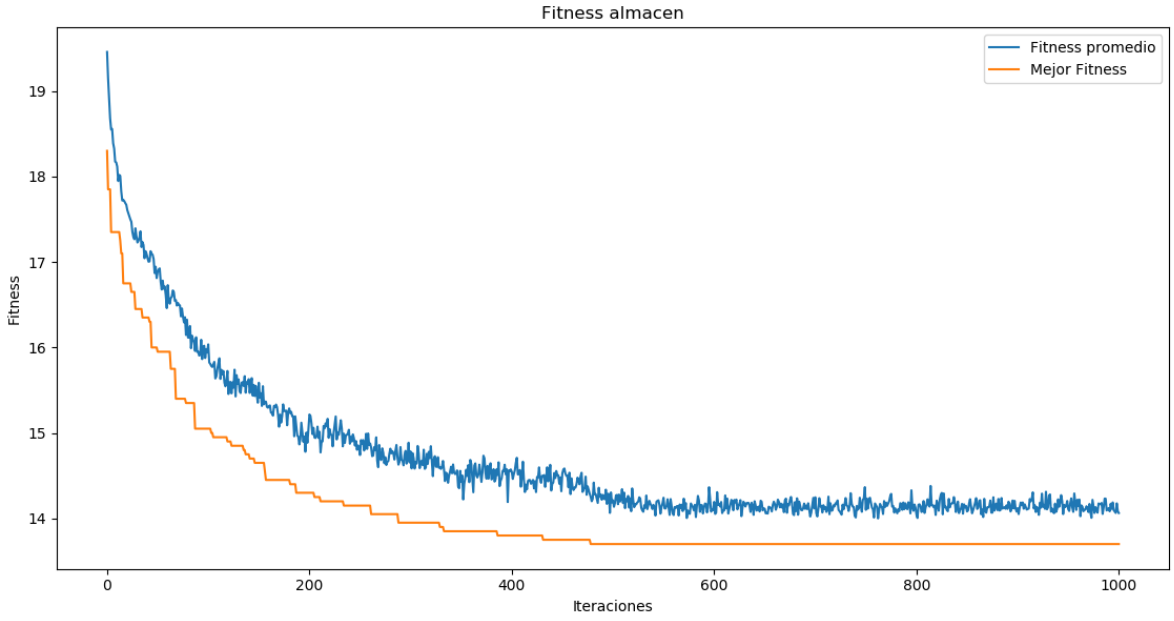

Fig. 7

Tamaño de la Población	Tiempo de Ejecución	Mejor Fitness	Mejor Individuo
10	3 min.	13,85	[20 6 1 17 31 15 12 7 29 4 18 8 30 9 13 11 32 5 25 21 10 16 24 19 22 23 28 26 2 14 27 3]
50	6 min.	13,7	[9 32 5 14 20 28 3 10 4 15 27 2 31 6 16 17 19 29 30 23 21 8 24 22 26 11 18 13 7 25 12 1]
100	9 min.	13,8	[29 32 12 22 26 9 1 18 19 8 20 31 3 5 24 4 14 10 28 11 27 16 21 13 23 30 15 6 17 2 25 7]
500	36 min.	13,85	[31 7 27 2 5 32 9 14 4 25 20 13 1 18 28 11 24 30 29 19 12 16 15 21 22 3 6 26 23 10 8 17]

Tabla 2

La Tabla 2 nos muestra que a medida que aumentamos el tamaño de la población, el fitness mejora, pero luego, si seguimos aumentado, este fitness comienza a empeorar. Es interesante notar que para obtener un fitness de 13,7 solo se necesitaron 6 minutos, mientras que en la Tabla 1 se ve como para un fitness de 13,15 se necesitó 1 hora y 20 minutos. Es decir, que para disminuir en 0,55 el valor del fitness se requirieron 74 minutos adicionales. Esto es importante notarlo, ya que, en situaciones donde el factor tiempo es primordial, según estos resultados y con este mismo algoritmo, convendría enfocarse en una población pequeña, pero iterando varias veces, en vez de iterar menos veces y agrandar la población.

El tercer grupo de pruebas consistió en utilizar una población de 100, iterar 1000 veces, y variar la cantidad de órdenes. Las Figuras 8 y 9 muestran este comportamiento. La Tabla 3 muestra los valores obtenidos.

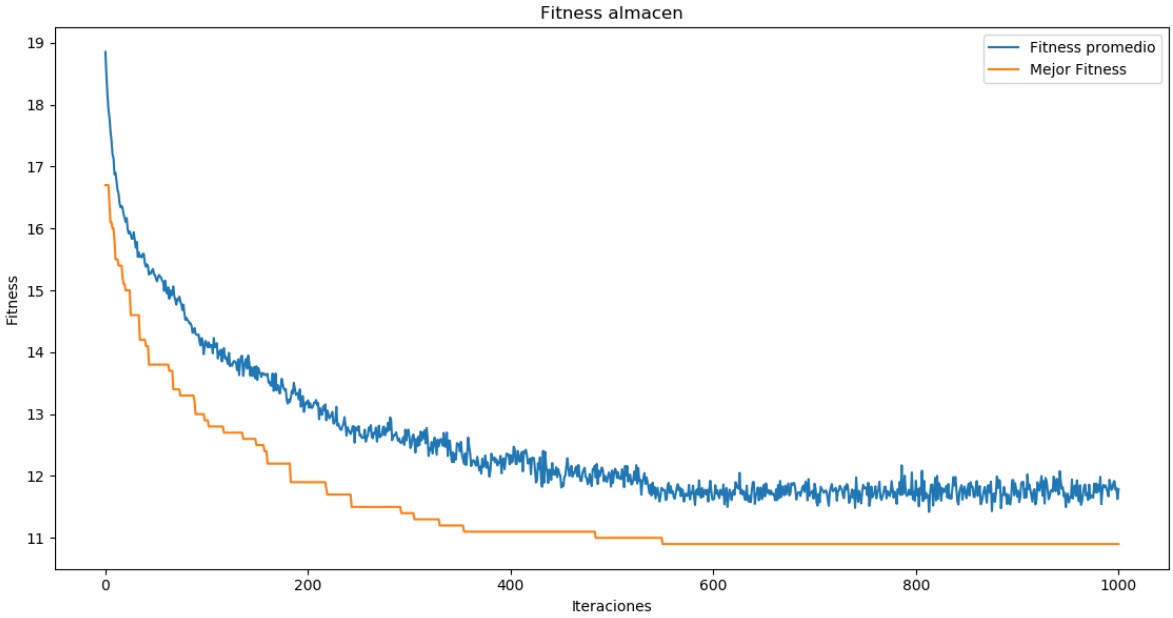

Fig. 8

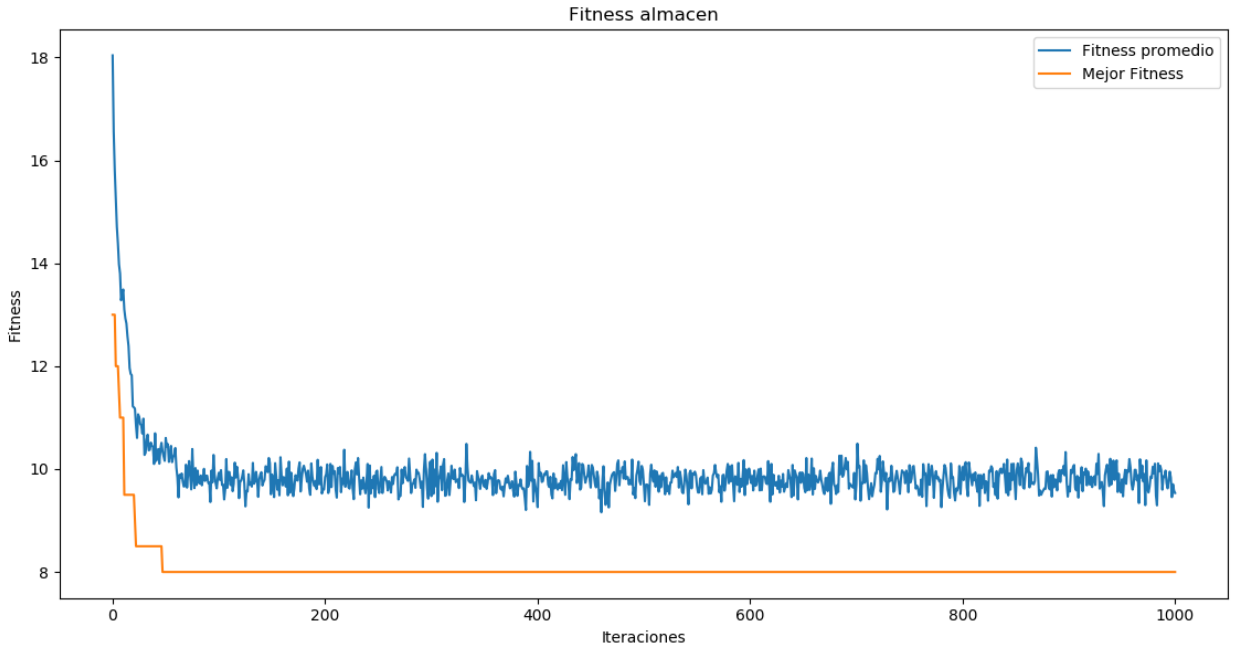

Fig. 9

Cantidad de Órdenes	Tiempo de Ejecución	Mejor Fitness	Mejor Individuo
20	9 min.	13.8	[29 32 12 22 26 9 1 18 19 8 20 31 3 5 24 4 14 10 28 11 27 16 21 13 23 30 15 6 17 2 25 7]
10	3 min.	10.9	[32 15 8 7 20 11 21 25 5 14 19 24 27 9 28 12 4 13 18 1 31 26 30 16 22 17 29 10 23 3 6 2]
5	2 min.	10.4	[9 13 4 22 14 31 3 17 32 30 6 15 5 26 19 1 11 20 29 18 8 16 21 12 28 27 24 7 25 2 10 23]
2	1 min.	8,0	[5 15 20 13 32 19 17 2 9 14 25 21 18 26 31 22 24 7 10 23 6 30 29 27 16 12 8 3 28 1 4 11]

Tabla 3

Este grupo de pruebas muestra como, a medida que se disminuye la cantidad de órdenes, el fitness mejora notablemente. Esto es lógico, ya que, al agregar órdenes (las cuales son variables en tamaño) podemos tener varias que necesiten distancias grandes para completarse, lo que hace que el fitness, que es sólo el promedio de las distancias de las órdenes, empeore.

Conclusiones

En este trabajo se presentó como un problema de optimización puede ser resulto con un algoritmo genético, el cual, desde el punto de vista de programación, no tiene una alta complejidad, y sin embargo, se obtienen buenos resultados.
Durante las pruebas realizadas, se mostró como al aumentar la población, el fitness mejoraba, pero a costo de un incremento muy grande de tiempo de procesamiento.
También se mostró como, con una población relativamente pequeña, pero con varias iteraciones, se obtienen resultados aceptables. Por último, se vió como la cantidad de órdenes esta fuertemente ligada al resultado del fitness.
En este trabajo se implementó una versión relajada del problema, donde no se tuvieron en cuenta las restricciones del almacén. Se deja para futuros análisis la implementación de

un algoritmo que evalúe dichas restricciones. A su vez, la comparación de otro tipo de algoritmo para resolver el problema del agente viajero (como por ejemplo otro AG, o recocido simulado) se plantea como otra alternativa.