# 

# 

# 

# 

# Centro de Investigación Científica y de Educación Superior de Ensenada

# Ciencias de la Computación

# Inteligencia Computacional para Optimización

# 

# 

# Problema del QAP

# 

# Autores:

# Yanet Perez Padilla

# Yan Carlos Leyva Labrador Lic.

# Tarea 2

# Posgrado: Ciencias de la Computación

# Ensenada, 2023

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 1. Introducción

El objetivo de esta práctica es resolver un problema de optimización típico utilizando técnicas de computación evolutiva. Para ello se van a implementar varias versiones o variantes sobre los algoritmos evolutivos para resolver el problema de la asignación cuadrática, incluyendo las variantes de:

* Algoritmo genético sin optimización local.
* Algoritmo genético con variante lamarckiana.
* Algoritmo genético con variante baldwidiana.

En este estudio, se analizarán las diferencias y rendimientos obtenidos con cada una de estar variantes, y se proporcionará un resultado de coste calculado para el conjunto de datos llamados *“tai256c.dat”*.

# 2. Hardware y software

Las pruebas del algoritmo evolutivo han sido ejecutadas con el siguiente hardware y sistema operativo:

* **Procesador:**
* **Memoria RAM:**
* **Tarjeta gráfica:**.

Para desarrollar el software necesario se ha utilizado lo siguiente:

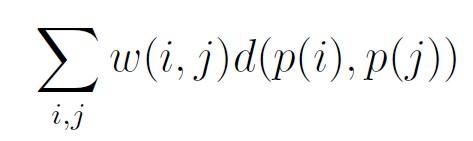
* **Lenguaje programación:**
* **Sistema operativo:**
* **IDE:**
* **Frameworks y librerías:**

# 3. Descripción del problema a abordar

El problema de la asignación cuadrática o QAP [Quadratic Assignment Problem] es un problema fundamental de optimización combinatoria con numerosas aplicaciones. En este caso tenemos una serie de flujos, que en este documento va a ser tratado como fábricas, y una serie de posibles localizaciones donde van a ser ubicadas dichas fábricas.

Para esto, se conoce la distancia que hay entre cada una de las posibles localizaciones y el flujo de materiales que existe para cada fábrica.

Formalmente, si llamamos d(i, j) a la distancia de la localización i a la localización j y w(i; j) al peso asociado al flujo de materiales que ha de transportarse de la fábrica *i* a la fábrica j, hemos de encontrar la asignación de instalaciones a localizaciones que minimice la función de coste:



Donde p() define una permutación sobre el conjunto de fábricas.

# 4. Representación del problema a abordar

En primer lugar, se va a representar la estructura de una posible solución del problema. Dicha estructura va a ser un **vector** donde cada posición va a representar a una unidad, y cada valor de dicha posición va a ser la localización.

Este vector ha sido incluido como atributo llamado “solutions” en una clase llamado ***“Individual”***. Dicha clase va a representar a lo que se correspondería con un individuo de la población.

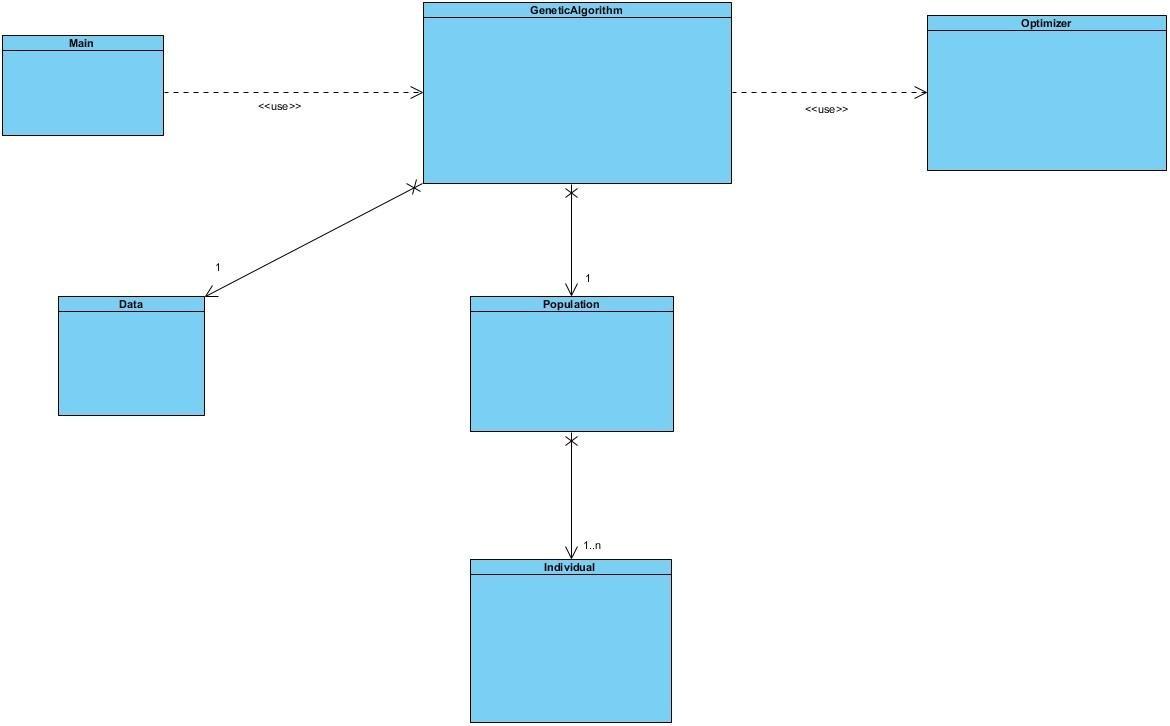
Tras esto, se ha creado una clase llamada ***“Data”***para poder cargar los datos desde los ficheros y que estos puedan ser tratados directamente desde nuestro algoritmo.

A continuación, se ha creado una clase llamada ***“Poblation”*,** con la que se va a modelar una población de individuos. Dicha población tendrá una serie de métodos asociados para trabajar con sus individuos.

Posteriormente se ha implementado una clase llamada **“*Optimizer”*** con la que se va a modelar los posibles algoritmos de optimización local que va a utilizar el algoritmo genético. En este caso se ha implementado un algoritmo de búsqueda local.

Por último, se ha diseñado la clase ***“GeneticAlgorithm”*** con la que se ha modelado el algoritmo genético. Esta clase es la principal de la aplicación y hace uso de todas las clases anteriores para su correcto funcionamiento. Esta clase contiene las variantes de los algoritmos (Lamarckiana, Baldwiniana y sin optimización local) junto con los operadores de selección cruce y mutación. Más adelante se detallará como se ha implementado cada uno de estos operadores.

El modelo conceptual de la aplicación se resume en la siguiente figura:



# 5. Descripción de los algoritmos utilizados

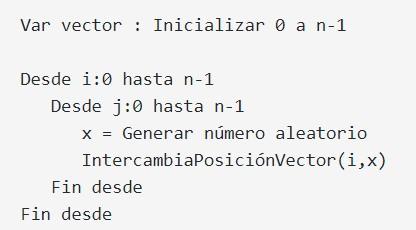
En esta sección se va a describir las principales estrategias utilizadas para la realización del algoritmo genético.

### 5.1 Individuos

#### 5.1.1 Generación de individuos

La generación de individuos se ha realizado totalmente de forma aleatoria. Un individuo va a formar parte de una posible solución al problema, por ello vamos a calcular una primera solución de forma aleatoria

Cada individuo se generará inicialmente de la siguiente manera: Se parte con un vector totalmente inicializado desde 0 hasta n-1, y la idea es intercambiar las posiciones de forma aleatoria. El pseudocódigo es el siguiente:



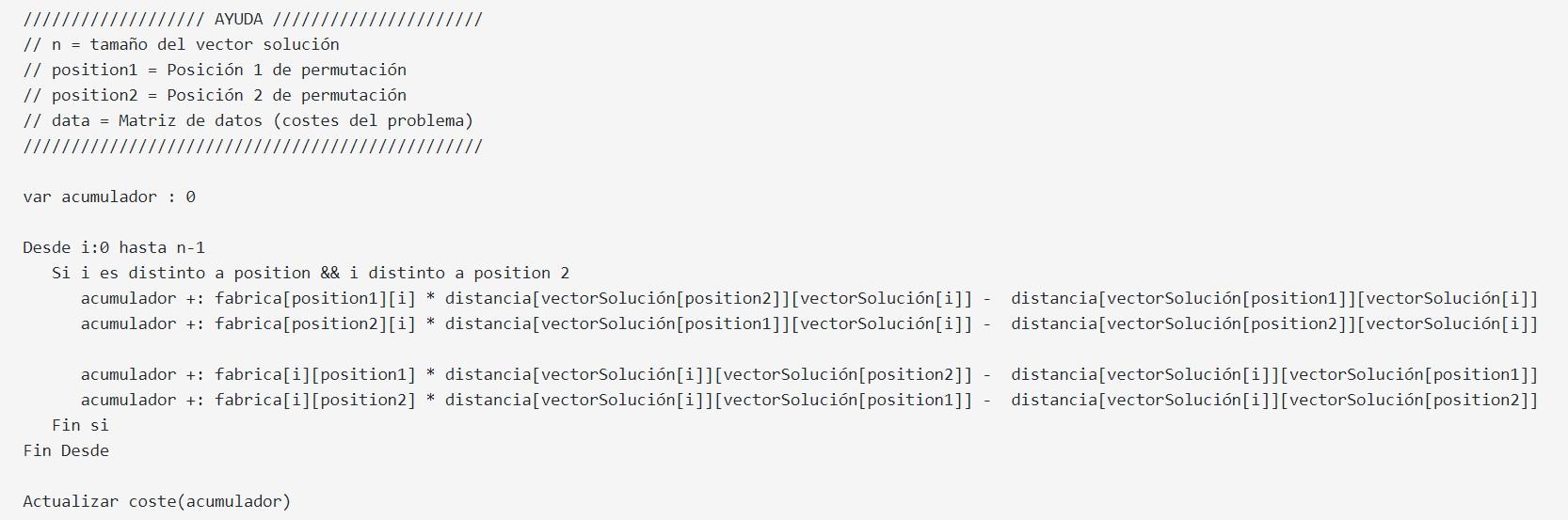
#### 5.1.2 Calcular coste de cada individuo

Cada individuo tendrá un coste asociado que se corresponderá con la bondad de la solución que representa. El cálculo de dicho coste se ha realizado de la siguiente forma:



#### 5.1.3 Ajustar nuevo coste del individuo

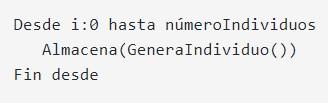
Con el objetivo de no tener que volver a recalcular todo el coste de un individuo cuando permutan solo dos valores (coste computacionalmente alto), se ha realizado una nueva función para poder recalcular el coste, partiendo como base el coste actual y calculando la diferencia que supondría respecto al nuevo cambio (permutación).



### 5.2 Población

#### 5.2.1 Generación de la población

Para la generación de las poblaciones se generan una serie de individuos aleatorios y se añaden a dicha población hasta completarla.



### 5.3 Optimización local

Para optimizar los individuos que inicialmente se generan en la población se ha utilizado una búsqueda local del mejor vecino. En este caso he seguido siguiente pseudocódigo[1] para realizar mi implementación:



### 5.4 Algoritmo genético

Este algoritmo nos permitirá ir obteniendo una serie de generaciones de individuos que van evolucionando y mejorando las soluciones previamente obtenidas. El modelo de la población es de tipo **generacional**, es decir, cada individuo va a vivir durante una generación, ya que en la siguiente generación serán reemplazados.

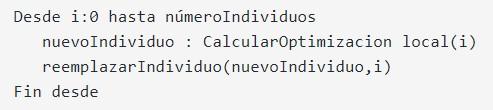
Como se ha comentado anteriormente, se va a implementar una variante lamarckiana, balwidiana y sin utilizar optimización local. El proceso básico que va a generar este algoritmo en cada iteración es el siguiente:



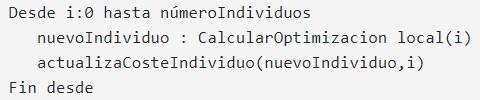
#### 5.4.1 Población inicial

Tal y como se ha comentado anteriormente, la generación de individuos se genera de forma aleatoria en la primera iteración. Tras la obtención de dichos individuos se procede a optimizarlos utilizando un algoritmo de búsqueda local y aplicando los siguientes cambios según la variante del algoritmo que se utiliza:

* **Variante lamarckiana:** Los nuevos individuos resultantes de la optimización local reemplazan al individuo de dicha población.



* **Variante balwidiana:** Se calculan los individuos resultantes de la optimización local, pero en este caso no reemplazan al individuo de la población, sino que el coste de esos individuos resultantes se utilizarán posteriormente en la selección.



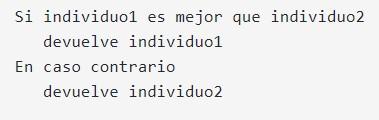
* **Variante sin optimización local:** No se aplica el algoritmo de optimización local.

#### 5.4.2 Selección de individuos

A la hora de realizar la selección de los individuos de la población inicial, para posteriormente convertirse en “padres” se tienen en cuenta dos factores:

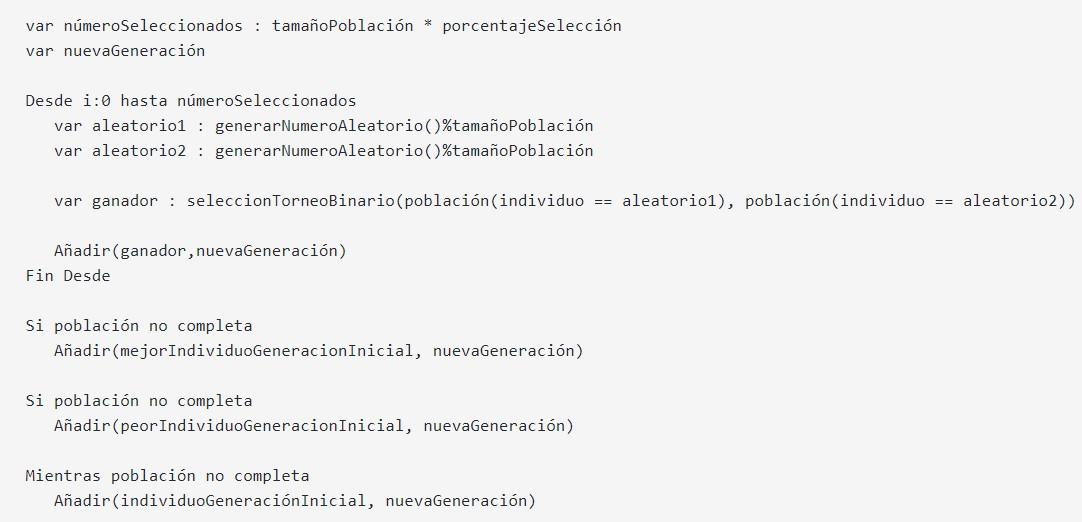
* **El primer factor es un porcentaje de selección.** Este es una variable del algoritmo que nos indica el porcentaje de individuos a los que se va a aplicar el operador de selección. El número restante de individuos es insertado directamente de la población inicial. Se añadirá el mejor individuo, el peor, y los mejores individuos restantes de la población (ordenada ascendentemente) hasta completar la población.

* **El segundo factor es la selección por torneo binario**. Este es el operador que se ha utilizado para realizar la selección, en el que se enfrentan dos individuos escogidos de forma aleatoria y se devuelve el mejor individuo para que pase a formar parte de los “padres”.



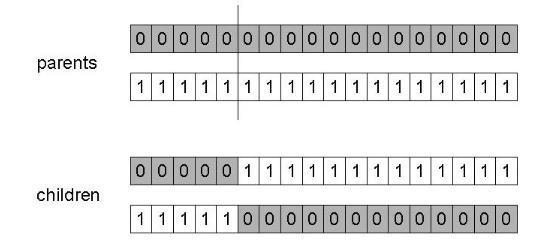
#### 5.4.3 Generación de padres

Tal y como se ha comentado en el punto anterior, la generación de los padres es realizada mediante la selección por torneo binario y parte de los mejores y el peor individuo de la generación inicial. Todo este procedimiento se describe a continuación:



#### 5.4.4 Cruce de individuos

Una vez se ha generado la generación de padres, se procede a realizar un cruce entre ellos para generar a la nueva generación “hija”. El operador de cruce que se ha empleado es de tipo **cruce en un punto**, en el que se selecciona un punto de cruce aleatoriamente, se dividen los padres en ese punto y se crean los hijos intercambiando partes de los cromosomas. En la siguiente figura se muestra un ejemplo de este tipo de cruce:



En el proceso de cruce, se seleccionarán dos padres de forma aleatoria y darán lugar a dos hijos resultantes (como se puede observar en la figura anterior). Es importante destacar que una vez que un padre ha sido seleccionado, dicho padre no se volverá a combinar con ninguno más, es decir, un padre solo podrá “reproducirse” una sola vez.

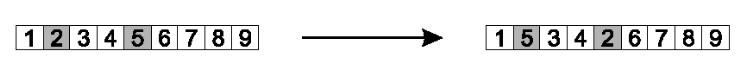
#### 5.4.5 Generación de hijos

La generación de hijos es resultado de realizar un cruce de individuos entre la generación de los padres. A continuación se muestra un pseudocódigo de la versión que he implementado para llevar a cabo esta tarea.

**Nota:** El siguiente pseudocódigo no es exacto a la representación real. Se ha realizado de esta forma para simplificar el pseudocódigo, pero en el código real implementado se realiza todas las verificaciones necesarias para que el proceso se realice correctamente. Para más información se puede consultar el método correspondiente *generateChildrenGeneration* en el archivo de *geneticAlgorithm.cpp*

#### 5.4.6 Mutación de individuos

Una vez que se ha generado la población de hijos, es posible que algún individuo de dicha población pueda mutar. Esta posibilidad está basada en una probabilidad de mutación que es establecida en el algoritmo genético. Dicha mutación se calcula mediante un número aleatorio generado según la probabilidad de mutación. El operador que se ha utilizado para realizar la mutación es mediante **intercambio**. Su funcionamiento se describe en la siguiente figura.



El pseudocódigo de este algoritmo es el siguiente:

#### 5.4.7 Evaluación de una generación

Tal y como se ha comentado en la introducción del punto 5 y haciendo un resumen en general, el proceso que sigue el algoritmo es la generación de una población inicial (lamarckiana, balwidiana o sin aplicar optimización local), posteriormente se selecciona y se generan unos padres que se cruzarán para dar a unos hijos, que tienen una probabilidad de mutar y dar lugar a una nueva generación.

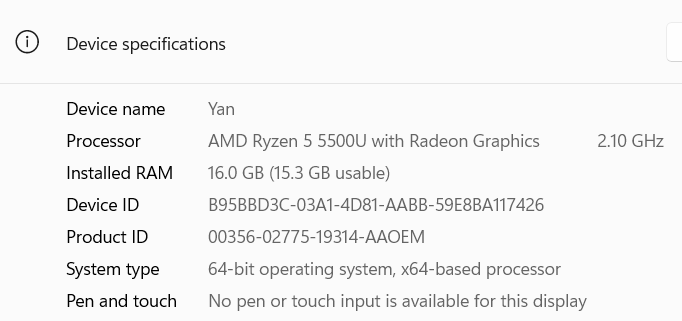
**Importante:** En cada uno de estos procesos en el que se genera una nueva población, se realiza una comparación con el mejor individuo obtenido, actualizándose en el caso de que se haya mejorado.

El criterio de parada del algoritmo es un valor de n generaciones, ya que no se conoce el óptimo del problema que se quiere resolver.

A continuación se muestra el pseudocódigo correspondiente al proceso de evaluación según la variante (lamarkiana, baldwidiana o sin optimización local).

## 6. Especificaciones del dispositivo

Las especificaciones del dispositivo y del sistema operativo empleado para ejecutar el algoritmo se muestran en la figura 4.



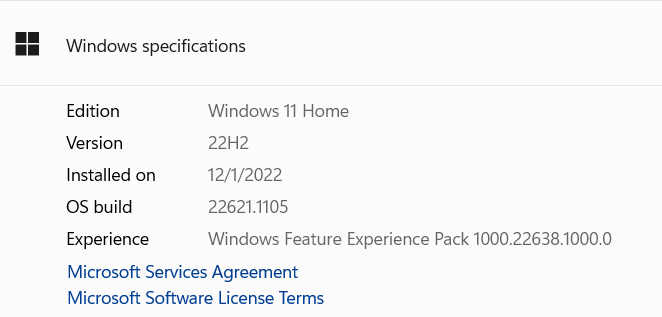


Figura 4. Especificaciones del dispositivo y del sistema operativo

Así mismo, el algoritmo se ejecutó en R-Studio Figura 4, culla versión de R es la 4.2.2 para Windows 64 bit.

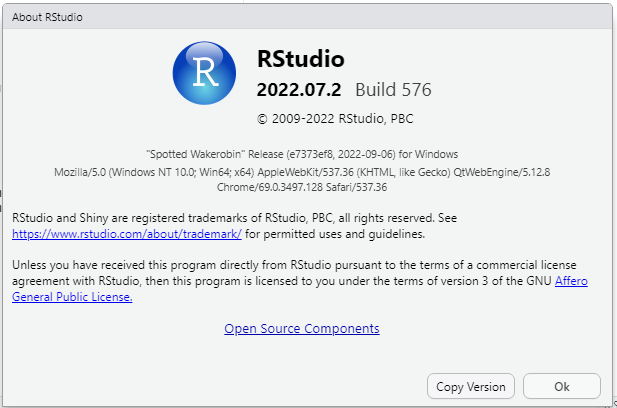


Figura 4. Versión de RStudio empleada

## 7. Análisis de resultados

**7.1 Algoritmo 1**

El algoritmo desarrollado es un algoritmo evolutivo. Este se programó en R y se ejecutó en paralelo, en 10 núcleos se hicieron tres repeticiones para cada base de datos, lo cual hace un total de 30 repeticiones por ejemplo.

Las bases de datos empleadas fueron: Lipa20a, kra30a, Tai35b, Tho40, sko100d

**7.1.1 Inicialización**

Se seleccionan aleatoriamente 300 individuos de la población.

**7.1.2 Selección de individuos**

Se selecciona por seis métodos diferentes un subconjunto de individuos de la población, y los dos individuos que resulten con menor fitness son los que se escogen para ser los padres de la siguiente generación.

Cada método tiene asociado un score y se escoge uno de manera proporcional a su respectivo score. Al inicio del algoritmo el score de cada método es 1. Según el método escogido su score se incrementa en una unidad por cada hijo resultante que tenga un fitness menor o igual que al menos uno de sus padres.los métodos son.

1- seleccionar 5 individuos de manera aleatoria de la población

2- seleccionar 10 individuos de manera aleatoria de la población

3- seleccionar 5 individuos con probabilidad proporcional a 1/fitness

4- seleccionar 10 individuos con probabilidad proporcional a 1/fitness

5- ordenar el fitness de la población de manera decreciente y seleccionar 5 individuos con probabilidad proporcional a la posición que ocupan en el ordenamiento (notar que los individuos de menor fitness serán los que se encuentran en una posición superior, por tanto estos tendrán mayor probabilidad de ser seleccionados)

6- ordenar el fitness de la población de manera decreciente y seleccionar 10 individuos con probabilidad proporcional a la posición que ocupan en el ordenamiento (notar que los individuos de menor fitness serán los que se encuentran en una posición superior, por tanto estos tendrán mayor probabilidad de ser seleccionados)

De los individuos seleccionados se escogen los dos que menor fitness tengan para ser los padres de la siguiente generación.

**7.1.3 Cruzamiento**

Para determinar si habrá cruzamiento se realiza un ensayo Bernoulli, el cual consiste en: generar un valor entre 0 y 1 de manera aleatoria, si este es menor que 0.9 entonces habrá cruzamiento, de lo contrario no se cruzaran los dos individuos resultantes en el proceso de selección, esto indica que los padres se cruzan con probabilidad de 0.9.

Para el cruzamiento se desarrollaron dos métodos:

1- el método explicado en **5.4.4**

2- este consiste en aplicar el método 1- a los padres y seguido se vuelve a aplicar al método 1 a los dos hijos resultantes.

La elección del método de cruzamiento es idéntica a la elección del método de selección.

**7.1.4 Mutaci**ó**n**

Para determinar si habrá mutación se realiza un ensayo Bernoulli, el cual consiste en: generar un valor entre 0 y 1 de manera aleatoria, si este es menor que 0.2 entonces se mutara cada hijo resultante del cruzamiento, esto indica que los hijos mutan con probabilidad de 0.2. notar que si en el paso anterior no hubo cruzamiento entonces la mutación se realizará sobre los padres. En caso de que los padres resultantes del proceso de selección no se hayan cruzado ni mutado, entonces se vuelve a repetir el proceso de selección.

Para la mutación se desarrollaron tres métodos:

1- el método explicado en **5.4.6**

2- se escogen dos posiciones aleatoriamente del vector de representación de la solución y se re-acomodan todos los elementos comprendidos en las dos posiciones seleccionadas, el reacomodo se realiza de manera aleatoria.

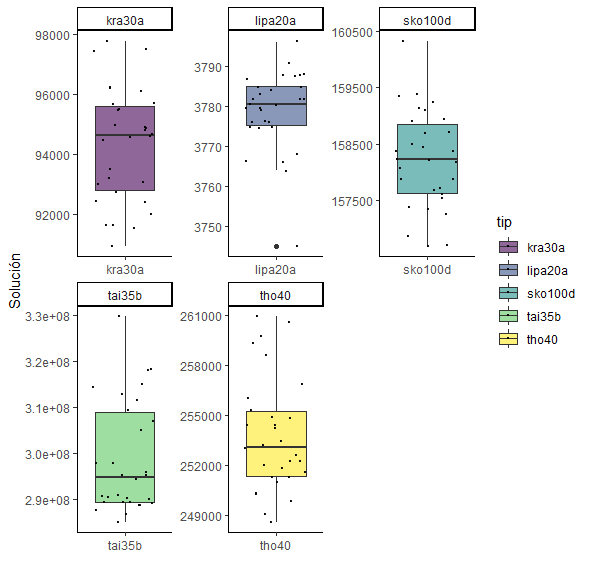
3- Se ejecuta una realización de una variable aleatoria con distribución Bin(n,p), donde n es el tamaño del vector de representación de la solución y p es una realización de una variable aleatoria uniforme en el intervalo (1/300,1/n), notar que 300 es el tamaño de la población, después se seleccionan, 2 veces la cantidad reportada por la variable aleatoria con distribución Binomial, de posiciones en el vector de representación de la solución, y por último se intercambian dos a dos los elementos que se encuentran en las posiciones antes seleccionadas.

**7.1.5 Reemplazo**

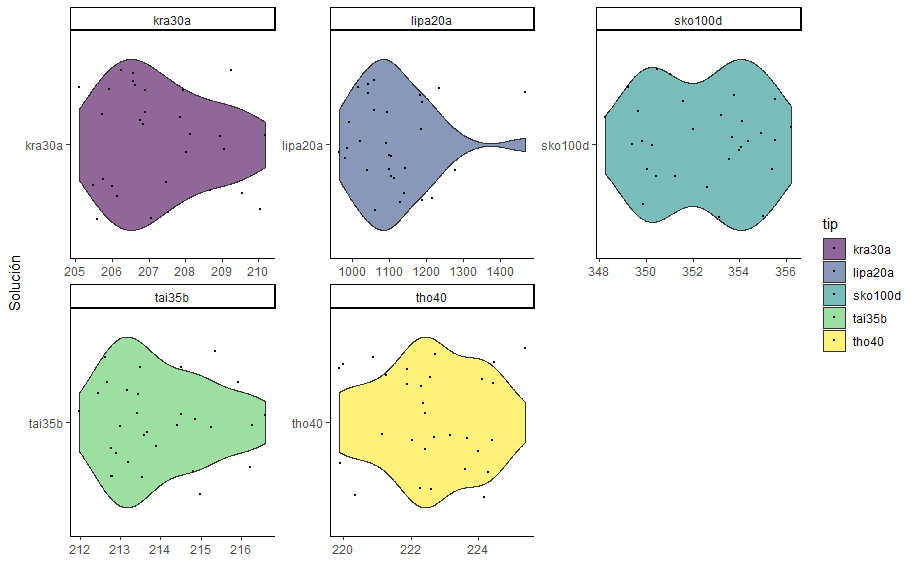
Si cada uno de los hijos tiene un fitness menor que el elemento en la población de mayor fitness entonces se realiza el reemplazo.

**7.2 Resultados del Algoritmo 1**

En ninguno de los casos se obtuvo el mínimo.



el tiempo de ejecucion



**7.3 Algoritmo 2**

Además del algoritmo evolutivo explicado anteriormente se desarrolló un segundo algoritmo, el cual es una instancia del algoritmo evolutivo antes explicado.

En el segundo algoritmo se introduce el concepto de vector de características asociado a una representación de la solución del problema, este vector recoge características topológicas y descriptivas de cada representación.

Sea x=[x1,x2,x3,x4,x5,....,xn] la representación de la solución, se determinan 9 características:

1- distancia de la solución al vector [1,2,3,4,5,...,n], donde n es la longitud de la solución.

2-área bajo la curva

3-distancia entre y1=[x1,x2,x3,x4,x5,....,xn/2] y y2=[xn/2,x2,x3,x4,x5,....,xn]

4-

5- cantidad de pendientes entre los puntos xi, xi+1 positivas (rectas orientadas hacia arriba)

6- cantidad de pendientes entre los puntos xi, xi+1 negativas (rectas orientadas hacia abajo)

7- cantidad de triángulos orientados hacia arriba

8- cantidad de triángulos orientados hacia abajo

9- cantidad de tres puntos consecutivos, xi=m,xi+1=m+1,xi+2=m+2

Las características topológicas extraídas siguen las ideas de los complejos simpliciales (más o menos)

el algoritmo consiste**:**

1- ejecutar el Algoritmo 1 hasta que se realicen 1000 evaluaciones de la función fitness.

2- estimar el vector de características (k ésima característica del i ésimo hijo) de cada hijo que se genera

3- crear una variable yi asociada al hijo i, tal que:

,donde pji es el padre j ésimo del hijo i ésimo.

4- entrenar un modelo de regresión logística

5- Ejecutar el Algoritmo 1 desde 7.1.2 al 7.1.4, a los hijos (h1, h2) que resulten en 7.1.4 se le aplica el modelo entrenado

6- generar una realización de

7- si entonces se acepta h1 y se calcula su fitnes

8- si entonces se acepta h2 y se calcula su fitnes

9- continuar con 7.1.5

10- Se almacenan las características de los nuevos hijos y cada 1000 hijos nuevos entonces se recalca el modelo.

Notas:

1- El algoritmo 2 es más lento que el 1, fundamentalmente porque se están almacenando las características de los hijos y eso es lo que repercute principalmente en el algoritmo.

2- Para obtener mejores resultados es necesario extraer un mejor vector de características asociado a cada representación de la solución.

3-lo que se está haciendo en los pasos del 7 al 10 es un ensayo bernoulli que sigue la misma idea del algoritmo de Metropolis Hasting

4- se puede crear un modelo que puede generar mejores resultados:

modelo jerarquico bayesiano:

son hiperparametros

vector de características respectivo al padre 1, padre 2 e hijo i

indicadora por ejemplo (tipo de selección, tipo de mutación, tipo de cruzamiento)

2\* aporte del hijo al fitness

aporte del padre 1 al fitness

aporte del padre 2 al fitness

se utiliza la distribución Poisson pues esta es la que mejor describe los eventos poco probables y como el mínimo es muy poco probable que ocurra, entonces creo que es la que mejor describe el problema (pero eso se puede pensar)

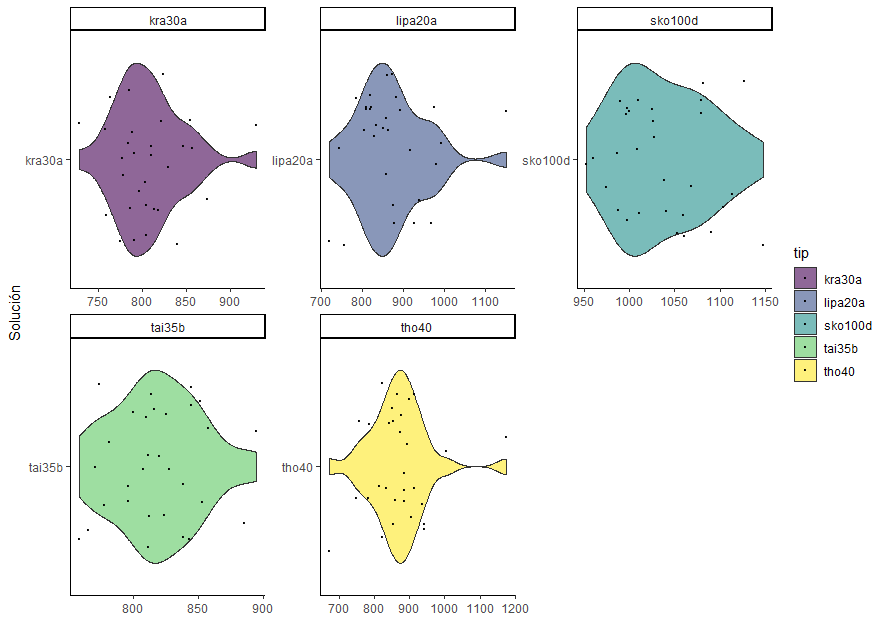
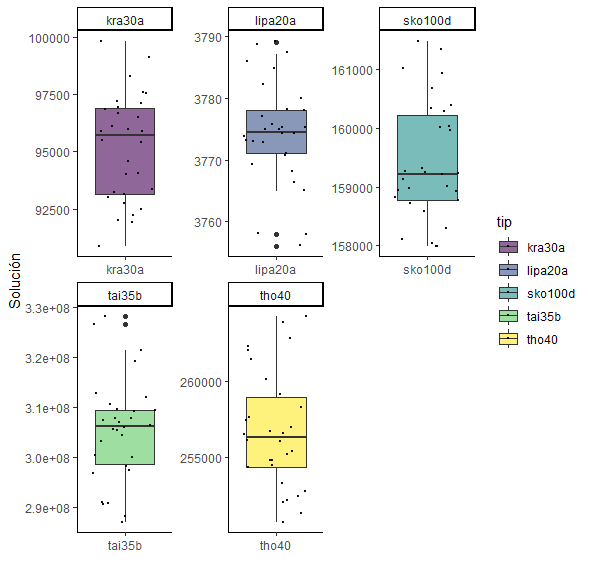
este modelo esta bueno pues es una formula de predecir el fitness del hijo y el aporte tanto del hijo como de los padres, esto ofrece otra manera de seleccionar los padres. Para determinar si se puede aceptar el hijo se puede hacer un ensayo Berboully

con

si

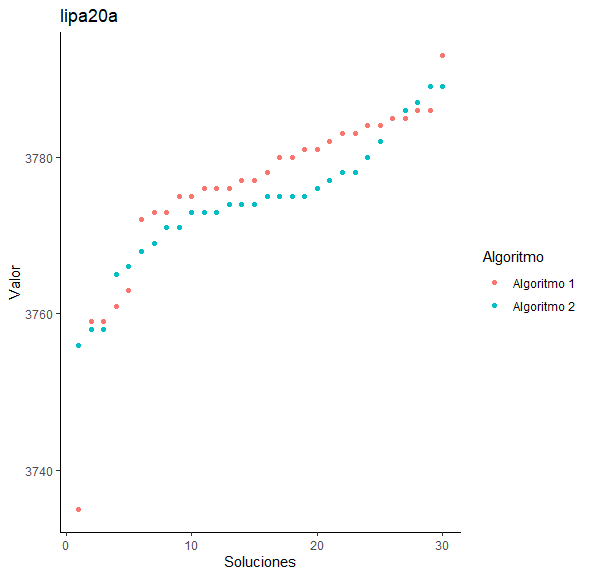
**7.4 Resultados del Algoritmo 2**

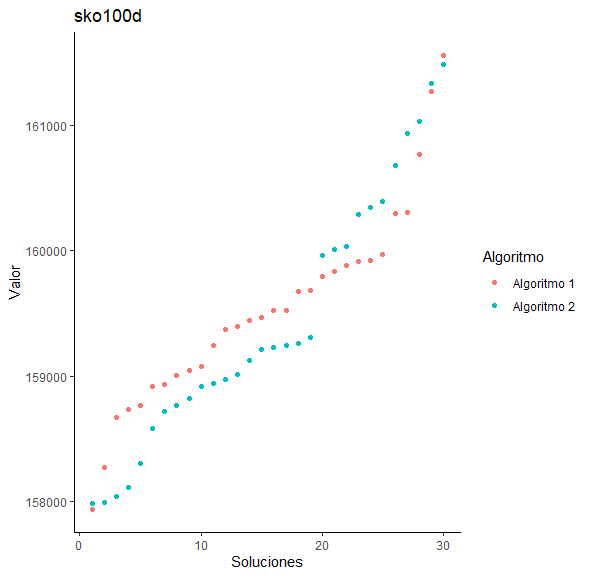
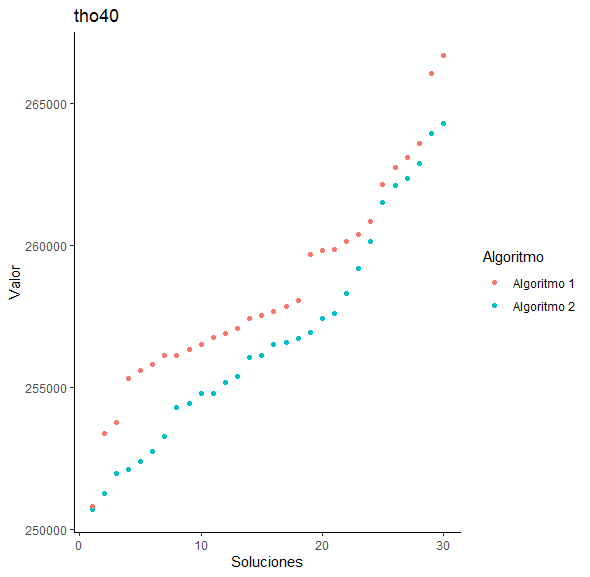
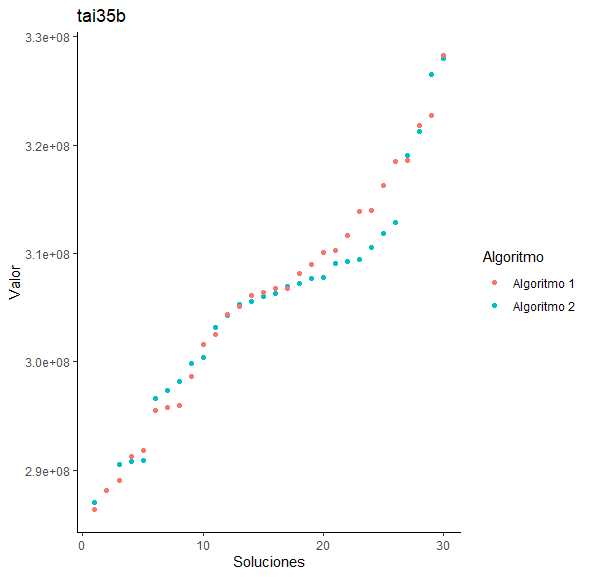
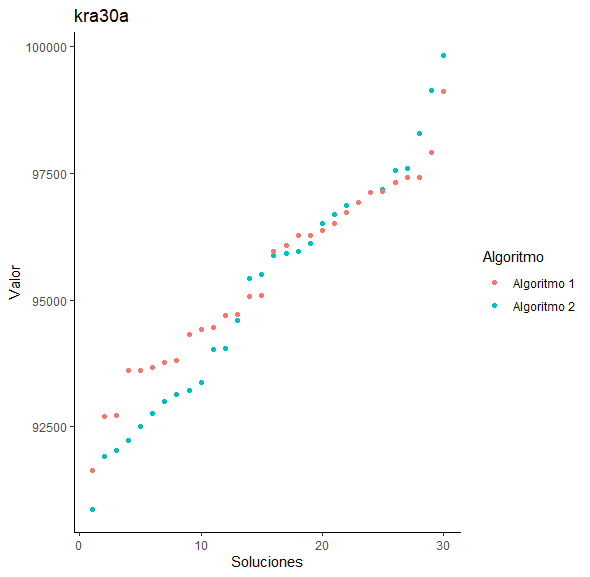
En ninguno de los casos se obtuvo el mínimo.

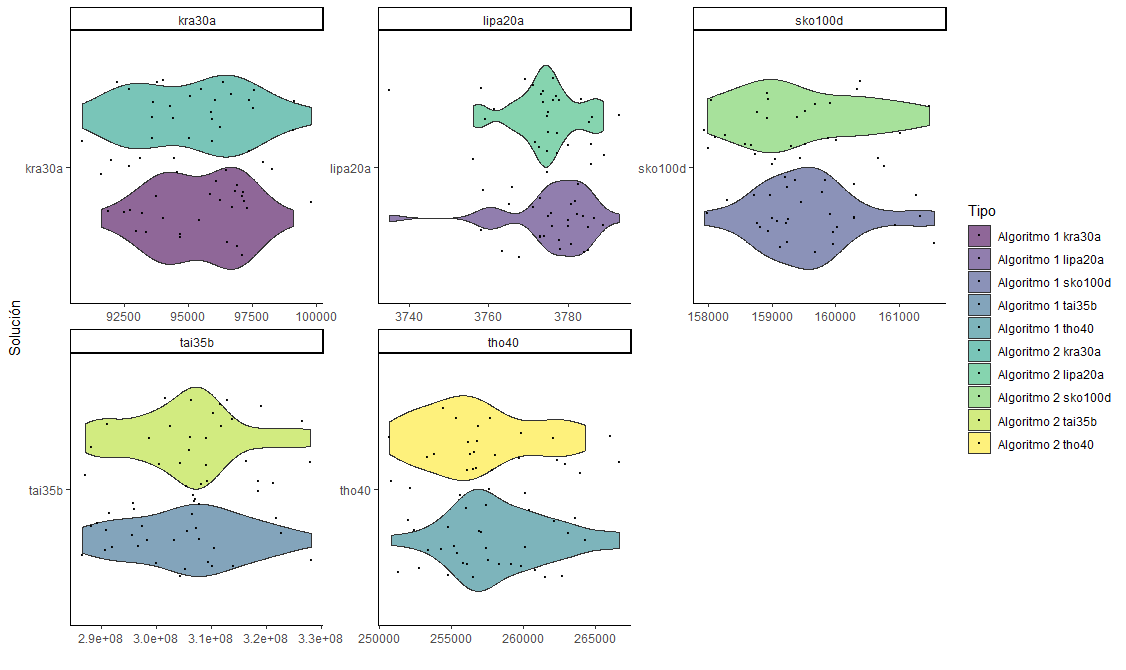


**7.5 Comparación entre algoritmos**

se muestran las comparaciones de las soluciones ordenadas entre algoritmos según la base de dato







Aparentemente existe una pequeña mejoría del algoritmo 2 con respecto al algoritmo 1