PRÁCTICA 1

Modelos Bioinspirados y Heurísticas de Búsquedas

alberto rodero peña

2022

ÍNDICE

[GREEDY 2](#_Toc108432648)

[BÚSQUEDA ALEATORIA 3](#_Toc108432649)

[BÚSQUEDA LOCAL 4](#_Toc108432650)

[Tamaño de lotes 4](#_Toc108432651)

[Tamaño de movimiento 5](#_Toc108432652)

[Ejecución aleatoria 6](#_Toc108432653)

[Mejor vecino 8](#_Toc108432654)

[ENFRIAMIENTO SIMULADO 9](#_Toc108432655)

[Velocidad de enfriamiento 9](#_Toc108432656)

[Tamaño de cambio vecino 9](#_Toc108432657)

[Condición de parada 10](#_Toc108432658)

[Ejecución aleatoria 11](#_Toc108432659)

[BÚSQUEDA TABÚ 13](#_Toc108432660)

[Máximas iteraciones 13](#_Toc108432661)

[Número de vecinos 14](#_Toc108432662)

[Número de plazas por cambio 14](#_Toc108432663)

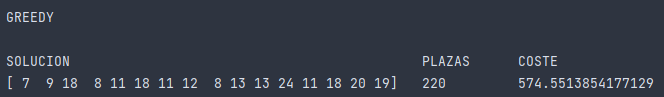
[División de plazas para matriz de frecuencias 15](#_Toc108432664)

[Número de reinicializaciones 15](#_Toc108432665)

[Ejecución aleatoria 16](#_Toc108432666)

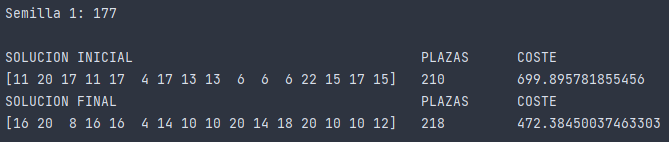
[COMPARACIÓN 18](#_Toc108432667)

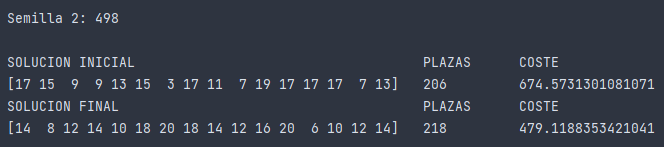
# GREEDY

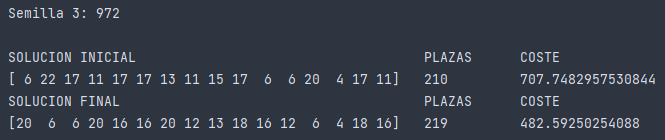


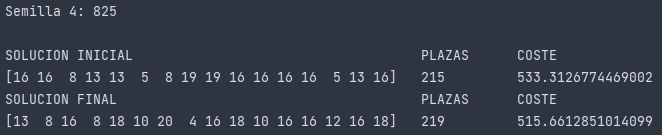
La solución greedy crea una solución manteniendo la misma relación que las plazas ocupadas inicialmente.

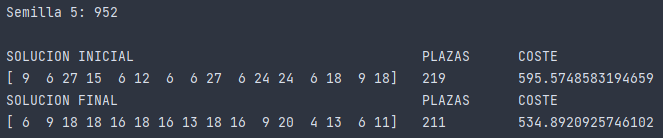
# BÚSQUEDA ALEATORIA











La búsqueda aleatoria genera 100 soluciones aleatorias y devuelve la que tiene menor coste

# BÚSQUEDA LOCAL

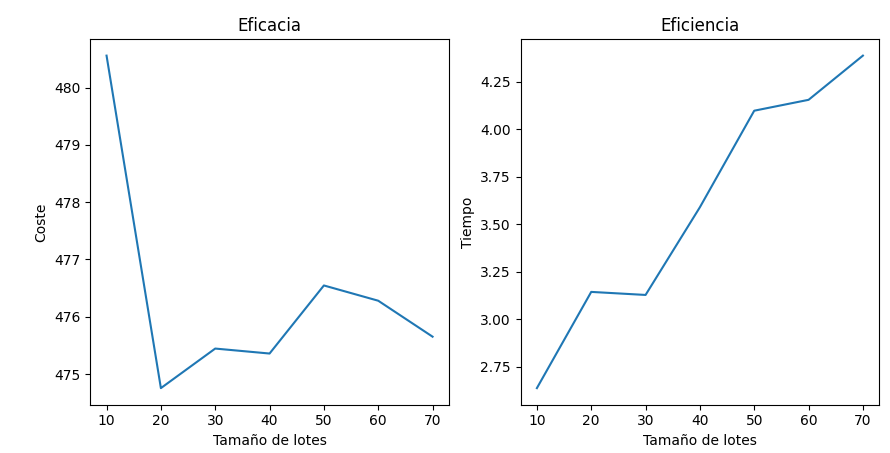
Implementación según el esquema del primer mejor vecino. Dividimos los vecinos en lotes de un tamaño determinado y comprobamos si por lote tenemos mejora. Esto se repetirá hasta que ningún vecino genere una mejora o hasta que ocurran 3000 evaluaciones. Los vecinos se generan como el traspaso de un número de espacios para bicis de una estación a otra.

Los dos parámetros que debemos estudiar son el tamaño de los lotes y el tamaño del movimiento.

Las semillas para los estudios de parámetros siempre serán [10,20,30,40,50]

## Tamaño de lotes

Estudiamos los valores para el tamaño de los lotes: [10,20,30,40,50,60,70]



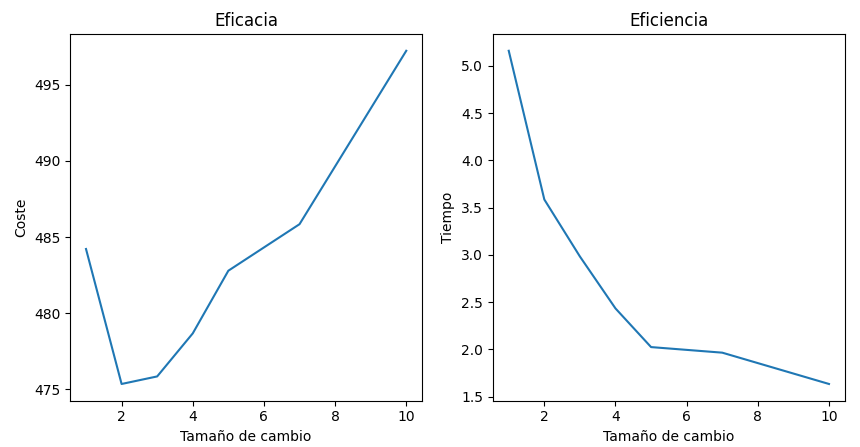
Se obtienen mejores resultados son para el tamaño de lotes 20

Medias: [480.55858257101147, 474.7542118638115, 475.4447989425654, 475.35803496815436, 476.54640009406756, 476.27997511914435, 475.6523844990741]

Tiempos: [2.6370771408081053, 3.14335618019104, 3.127345561981201, 3.5888957023620605, 4.096758460998535, 4.153993606567383, 4.386572599411011]

## Tamaño de movimiento

Estudiamos los valores para el tamaño de los lotes: [1,2,3,4,5,7,10]



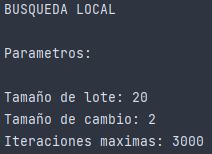
Se obtienen mejores resultados son para el tamaño de cambio 2

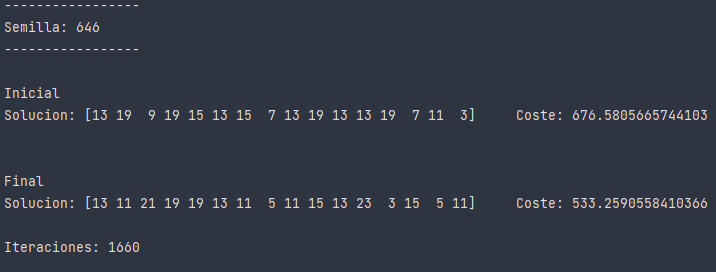
Medias: [484.2162842738286, 475.35803496815436, 475.8540852997081, 478.67902344939847, 482.79099153289235, 485.8433019797718, 497.22364965020796]

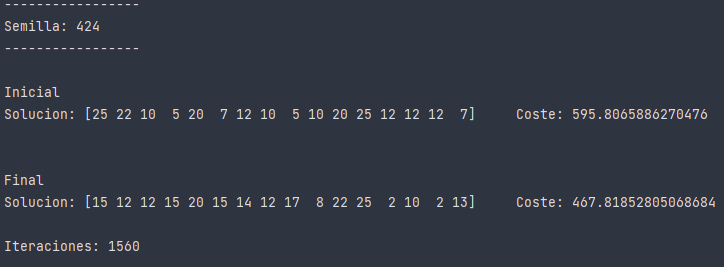
Tiempos: [5.15837607383728, 3.5858640670776367, 2.9831562519073485, 2.4340280532836913, 2.02528715133667, 1.9663840770721435, 1.6352266311645507]

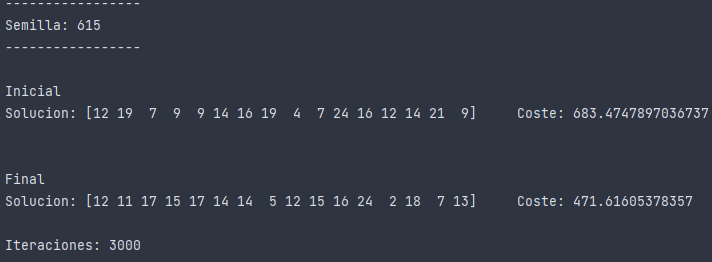
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Tamaño Lote | 10 | **20** | 30 | 40 | 50 | 60 | 70 |
| Media coste | 480.5585 | **474.7542** | 475.4447 | 475.3580 | 476.5464 | 476.2799 | 475.6523 |
| Tiempo | 2.6370 | **3.1433** | 3.1273 | 3.5888 | 4.0967 | 4.1539 | 4.3865 |
| Tamaño Cambio | 1 | **2** | 3 | 4 | 5 | 6 | 7 |
| Media coste | 484.2162 | **475.3580** | 475.8540 | 478.6790 | 482.7909 | 485.8433 | 497.2236 |
| Media tiempo | 5.1583 | **3.5858** | 2.9831 | 2.4340 | 2.0252 | 1.9663 | 1.6352 |

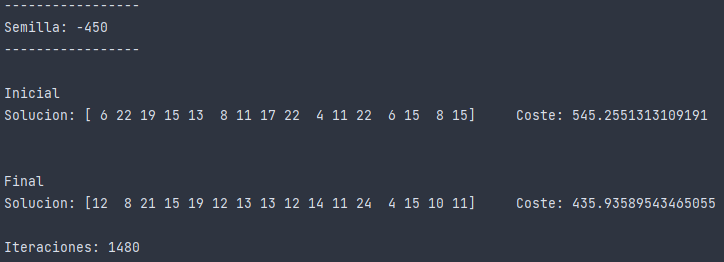
## Ejecución aleatoria

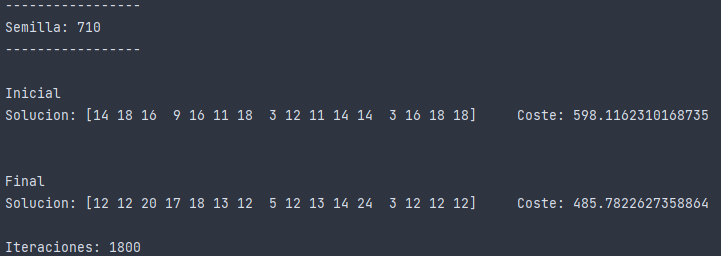


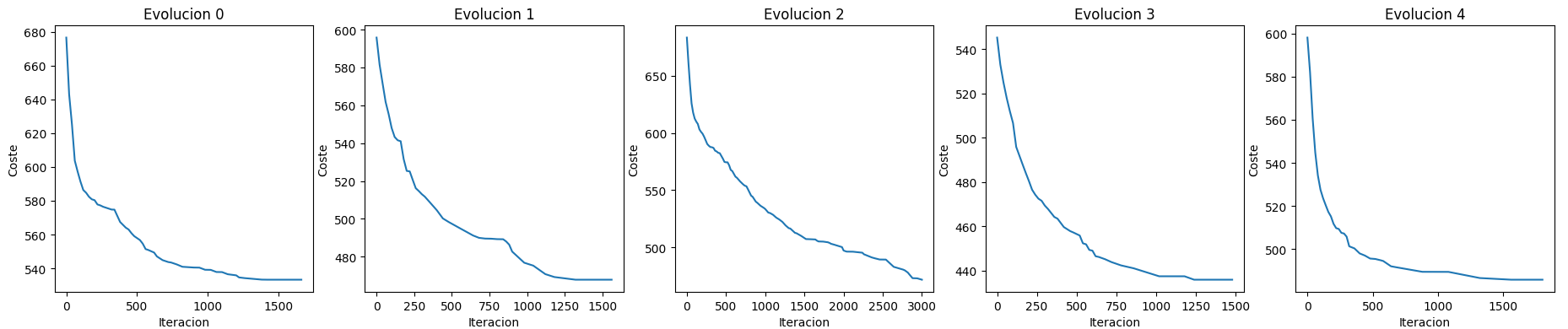












## Mejor vecino

Suponiendo que se quiera implementar el mejor vecino con un límite de 40 plazas por estación con límite de 20 minutos:

Tiempo por evaluación: 0.001980304718017578 segundos

240 permutaciones x 40 posibles cambios: 9600 posibles evaluaciones por mejora, 19.01092529296875 segundos

Para una media de 20 minutos tendrían que realizar se unas 60 mejoras:

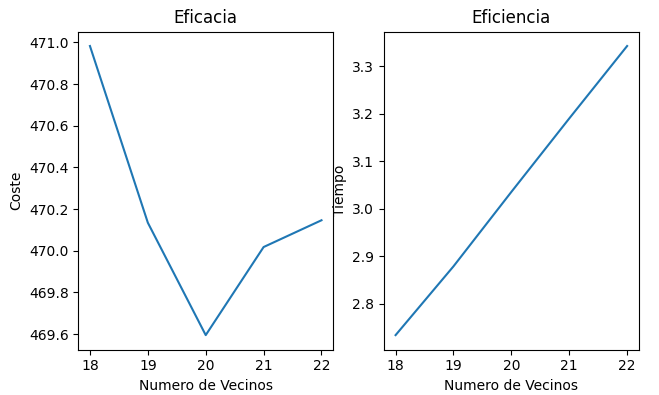
240 permutaciones \* 40 posibles cambios \* tiempo evaluación \* 60 mejoras / 60 segundos: 19.01092529296875 minutos

# ENFRIAMIENTO SIMULADO

Para valores de phi y mu entre 0.1 y 0.3 la mejor combinación es 0.11 y 0.11. Para estos parámetros la media de soluciones iniciales aceptadas es 90% con un mínimo del 85%. Ninguna otra combinación de parámetros aceptaba de media algo menor. El estudio se ha realizado con 100 iteraciones para temperatura inicial variando los valores phi y mu.

## Velocidad de enfriamiento

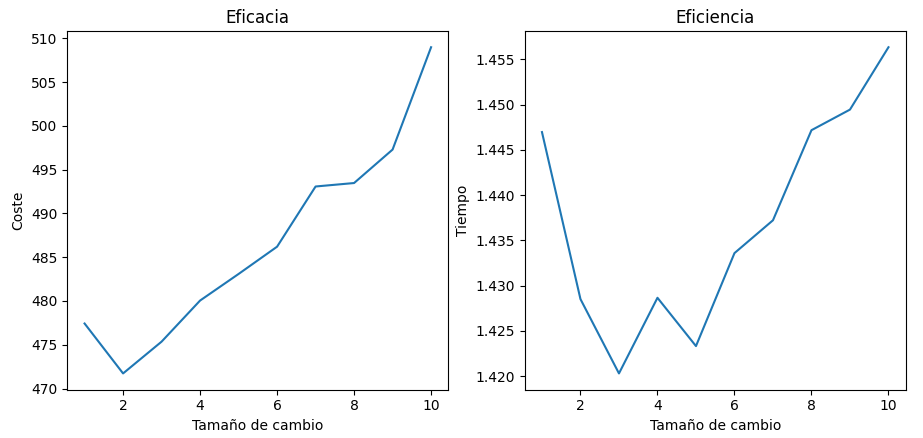
Vamos a variar la velocidad de enfriamiento (número de vecinos generados) para ver cómo afecta al coste



Como vemos el número de vecinos que ofrece el mejor coste es 20.

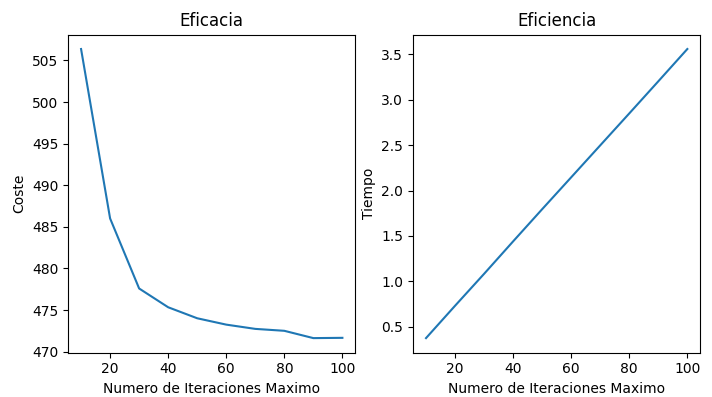
## Tamaño de cambio vecino

Para crear un vecino movemos un numero de plazas de una estación a otra, variamos esta cantidad para ver cómo afecta al coste



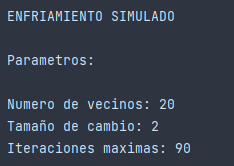
Para mejor coste usamos el tamaño de cambio 2

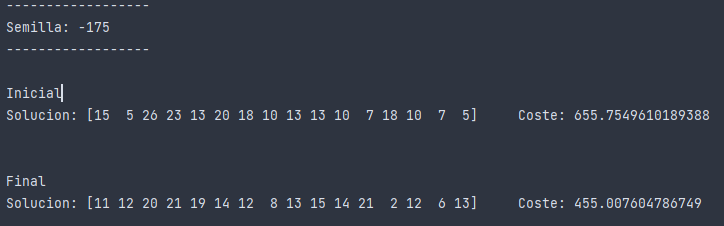
## Condición de parada

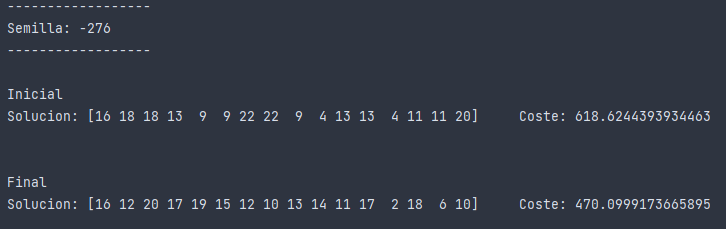


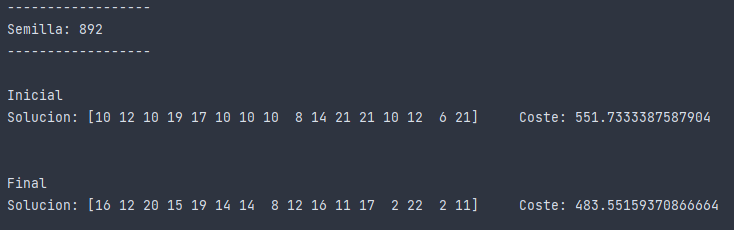
Se puede ver que cuantas más iteraciones mejores resultados, pero a partir de las 70 la mejora frena. Elegimos 90 iteraciones, ya que no tiene un tiempo muy elevado

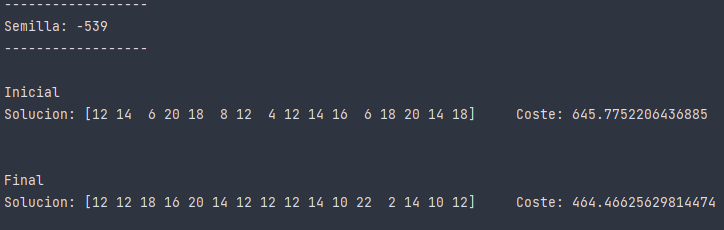
## Ejecución aleatoria

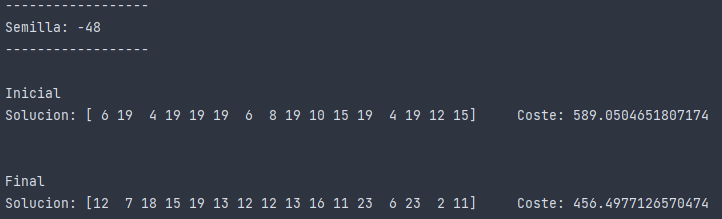


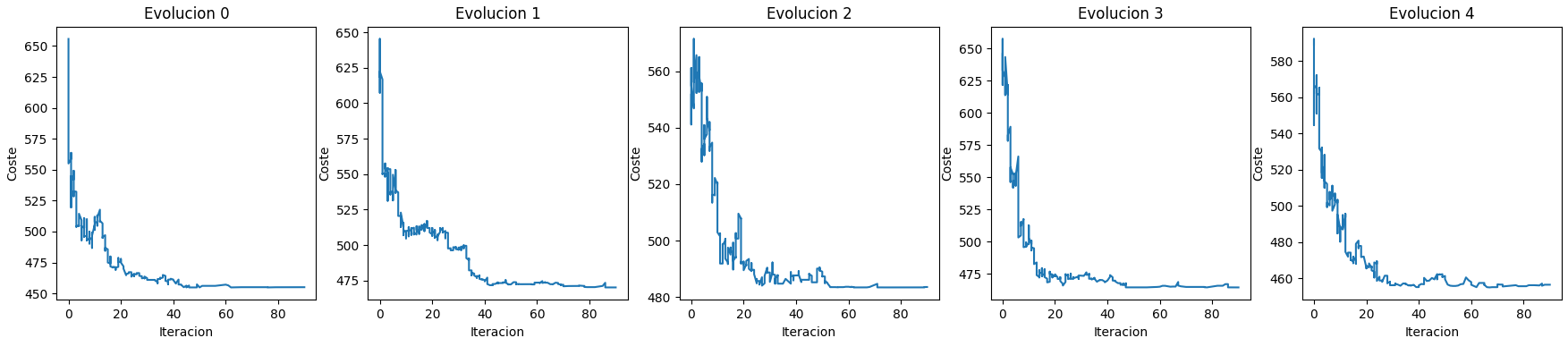












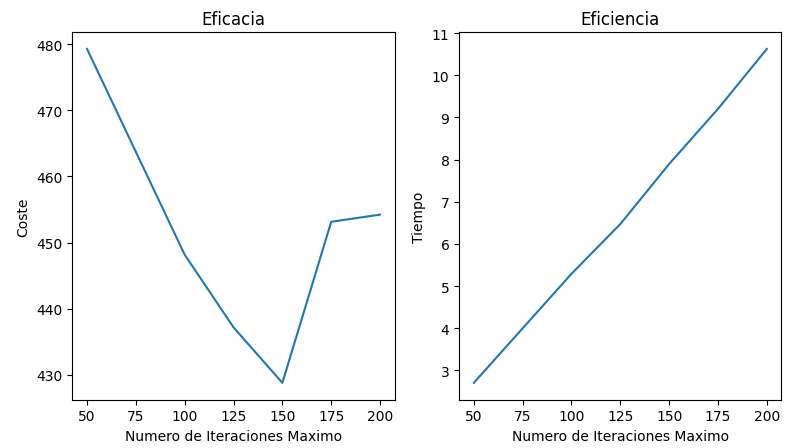
# BÚSQUEDA TABÚ

La implementación tiene 3 estrategias de reinicialización, 25% con una solución aleatoria, 25% desde la mejor solución obtenida y 50% partiendo de una nueva solución greedy.

La matriz de frecuencias se ha codificado en bloques de 10 plazas.

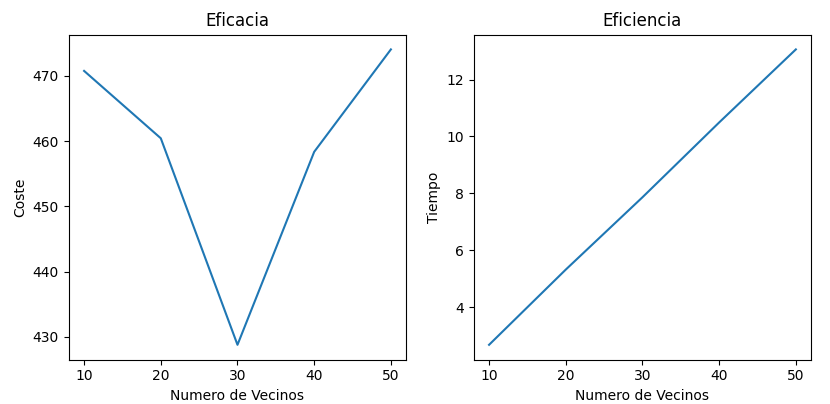
La tabla tabú tiene los movimientos restringidos y la tabla de tiempos tabú, el tiempo que se aplican estas restricciones

## Máximas iteraciones



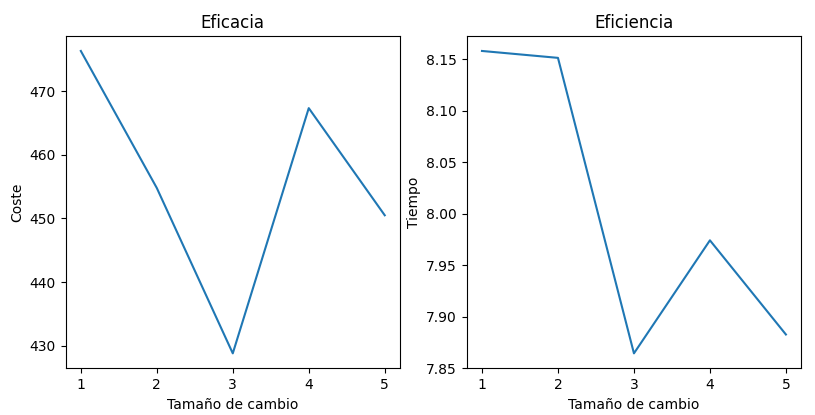
La mejor solución es con 150 iteraciones máximas, a una media de 8 segundos por semilla

## Número de vecinos



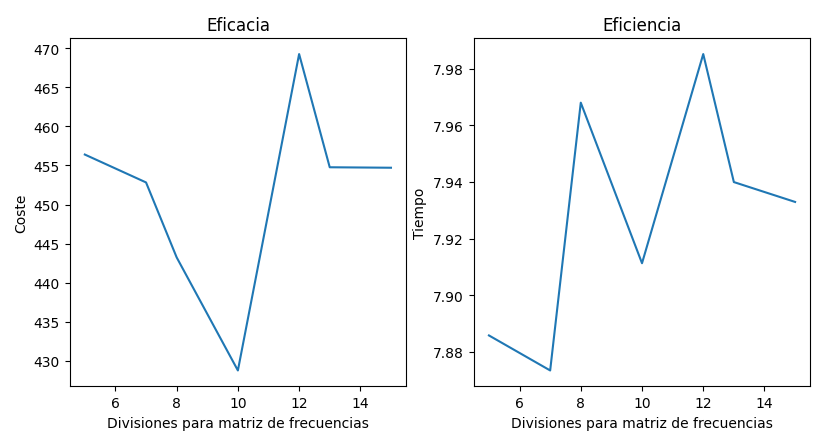
El mejor número de vecinos a generar por iteración es 30.

## Número de plazas por cambio



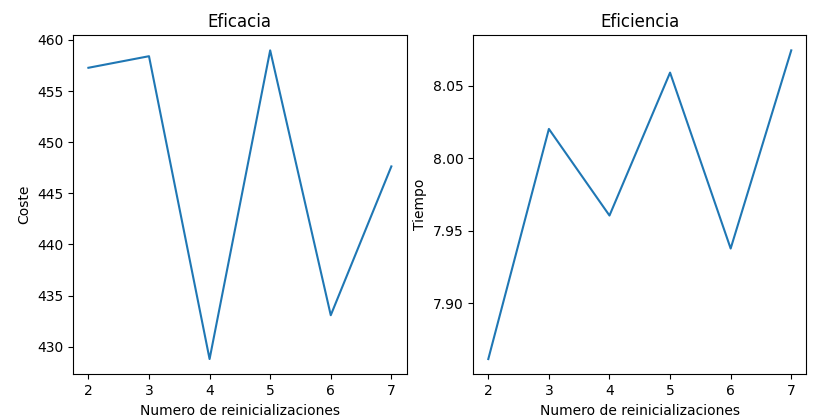
Como podemos ver el número óptimo de plazas cambiadas por generación de vecino es 3.

## División de plazas para matriz de frecuencias



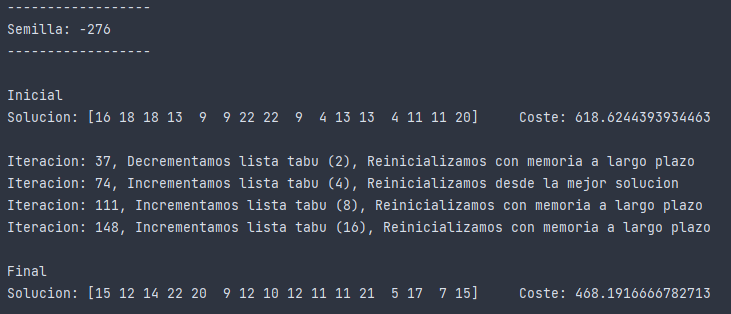
Lo mejor es dividir las plazas para la matriz de frecuencia en bloques de 10. 0-9 🡪 bloque 1, 10-19 🡪 bloque 2, etc.

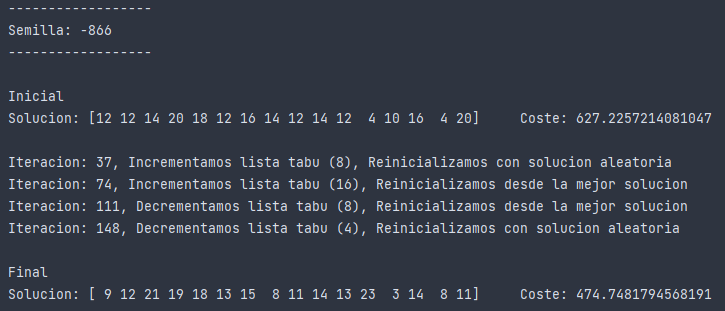
## Número de reinicializaciones

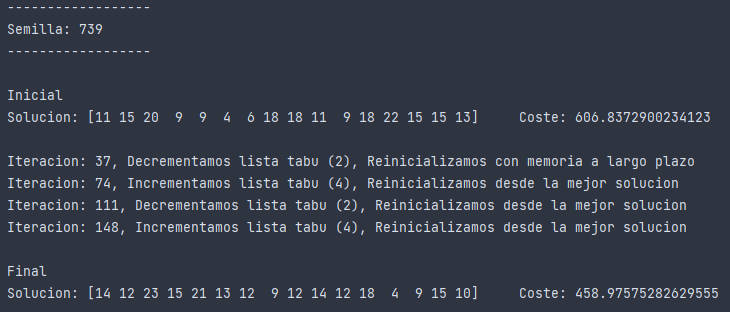


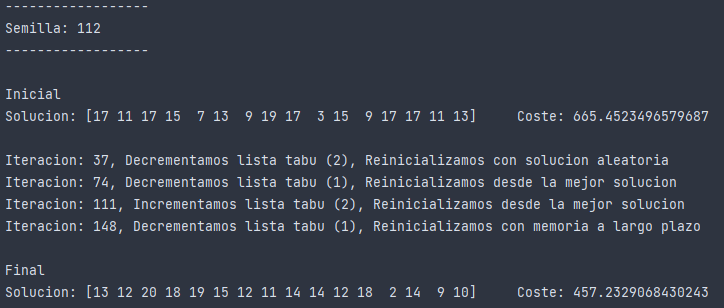
El mejor número de reinicializaciones, como se nos indica en la práctica, es 4.

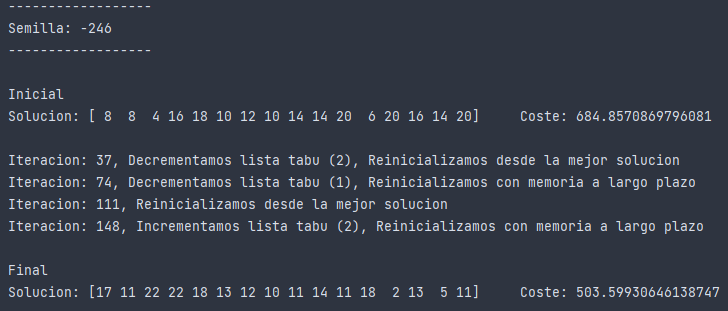
## Ejecución aleatoria

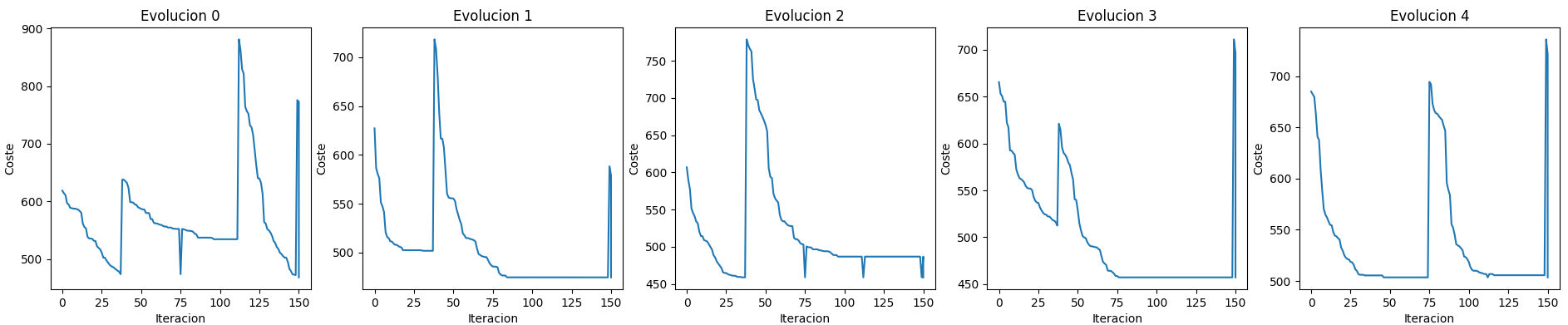












# COMPARACIÓN

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Algoritmo | Media Ev. | Mejor Ev. | Desv. Ev. | Media Coste | Mejor Coste | Desv. Coste |
| Greedy | 1 | 1 | 0 | 574.55 | 574.55 | 0 |
| Búsqueda Aleatoria | 100 | 100 | 0 | 480.49 | 454.84 | 22.31 |
| Búsqueda Local | 1848 | 1340 | 463.66 | 474.75 | 453.24 | 21.5 |
| Enfriamiento Simulado | 1801 | 1801 | 0 | 475.57 | 453.24 | 21.55 |
| Búsqueda Tabú | 4500 | 4500 | 0 | 428.79 | 385.41 | 33.85 |

Como se puede ver la **búsqueda tabú** es la que mejor resultado ofrece, aunque también es la que más tarda. En nuestro caso el tiempo de ejecución es suficientemente bajo como para utilizar la **búsqueda tabú** en más semillas para buscar la mejor solución posible.

La **búsqueda local** también tiene muy buen resultado comparado con el tiempo que tarda. Es la única que tiene evaluaciones variables, ya que en las demás la condición de parada es simplemente un número de iteraciones máximas. Aquí también tenemos ese número, pero podemos salir antes si no se produce una mejora en los vecinos.

El **enfriamiento simulado** debería ofrecer mejores resultados que la **búsqueda local**, ya que tiene una parte de exploración, pero aún optimizando los parámetros todo lo que pude no conseguí obtener mejores resultados.

El resultado de la **búsqueda aleatoria** no es completamente malo, al menos ofrece un mejor resultado que la solución greedy. Sin embargo, el usarlo es poco viable, ya que el resultado es completamente aleatorio.