2022

PRÁCTICA 1

MODELOS BIOINSPIRADOS Y HEURÍSTICAS DE BÚSQUEDAS

ALBERTO RODERO PEÑA

ÍNDICE

GREEDY

```
GREEDY

SOLUCION                                              PLAZAS      COSTE
[ 7  9 18  8 11 18 11 12  8 13 13 24 11 18 20 19]   220         574.5513854177129
```

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

BÚSQUEDA ALEATORIA

```
Semilla 1: 177

SOLUCION INICIAL                                    PLAZAS      COSTE
[11 20 17 11 17  4 17 13 13  6  6  6 22 15 17 15]   210         699.895781855456
SOLUCION FINAL                                      PLAZAS      COSTE
[16 20  8 16 16  4 14 10 10 20 14 18 20 10 10 12]   218         472.38450037463303
```

```
Semilla 2: 498

SOLUCION INICIAL                                    PLAZAS      COSTE
[17 15  9  9 13 15  3 17 11  7 19 17 17 17  7 13]   206         674.5731301081071
SOLUCION FINAL                                      PLAZAS      COSTE
[14  8 12 14 10 18 20 18 14 12 16 20  6 10 12 14]   218         479.1188353421041
```

```
Semilla 3: 972

SOLUCION INICIAL                                    PLAZAS      COSTE
[ 6 22 17 11 17 17 13 11 15 17  6  6 20  4 17 11]   210         707.7482957530844
SOLUCION FINAL                                      PLAZAS      COSTE
[20  6  6 20 16 16 20 12 13 18 16 12  6  4 18 16]   219         482.59250254088
```

```
Semilla 4: 825

SOLUCION INICIAL                                    PLAZAS      COSTE
[16 16  8 13 13  5  8 19 19 16 16 16 16  5 13 16]   215         533.3126774469002
SOLUCION FINAL                                      PLAZAS      COSTE
[13  8 16  8 18 10 20  4 16 18 10 16 16 12 16 18]   219         515.6612851014099
```

```
Semilla 5: 952

SOLUCION INICIAL                                    PLAZAS      COSTE
[ 9  6 27 15  6 12  6  6 27  6 24 24  6 18  9 18]   219         595.5748583194659
SOLUCION FINAL                                      PLAZAS      COSTE
[ 6  9 18 18 16 18 16 13 18 16  9 20  4 13  6 11]   211         534.8920925746102
```

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

BÚSQUEDA LOCAL

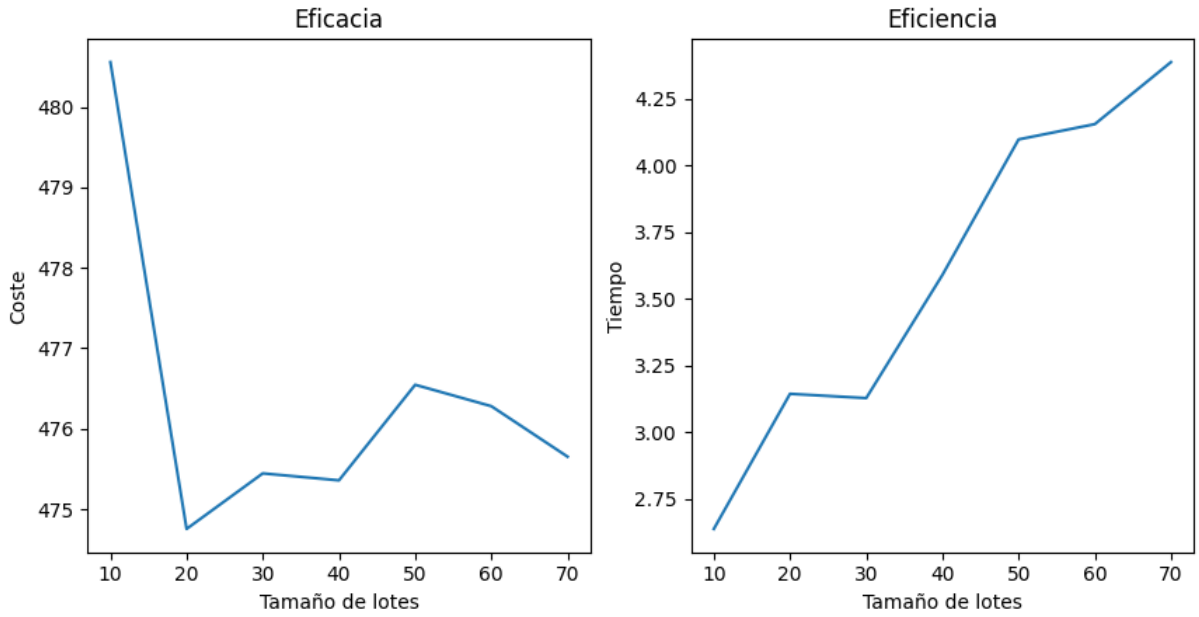
Implementación según el esquema del primer mejor vecino. Dividimos los vecinos en lotes de un tamaño determinado y comprobamos si por lote tenemos mejora. Esto se repetirá hasta que ningún vecino genere una mejora o hasta que ocurran 3000 evaluaciones. Los vecinos se generan como el traspaso de un número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



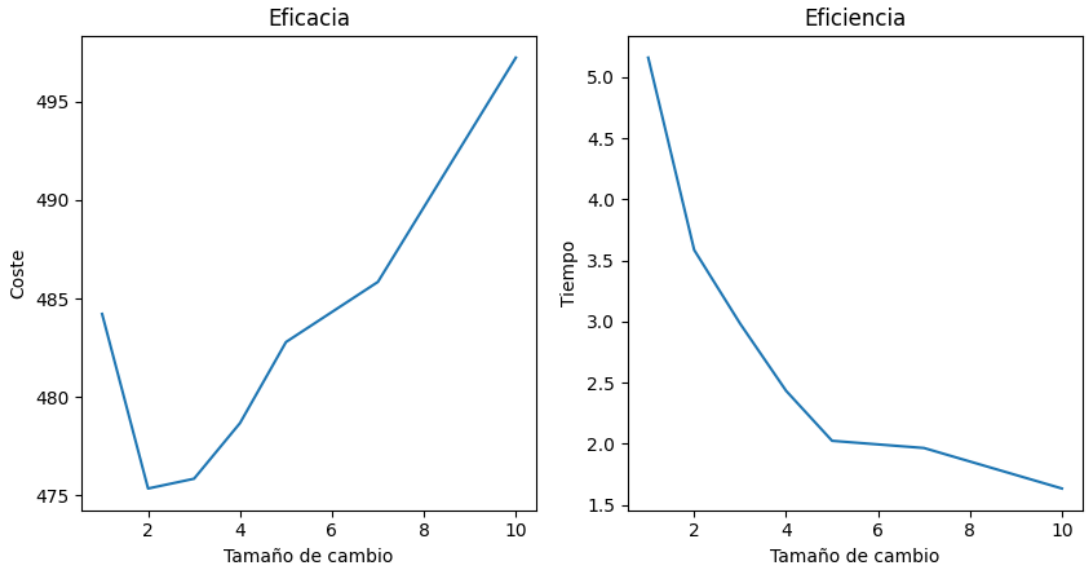
Se obtienen mejores resultados son para el tamaño de lotes 20

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

```
BUSQUEDA LOCAL

Parametros:

Tamaño de lote: 20
Tamaño de cambio: 2
Iteraciones maximas: 3000
```

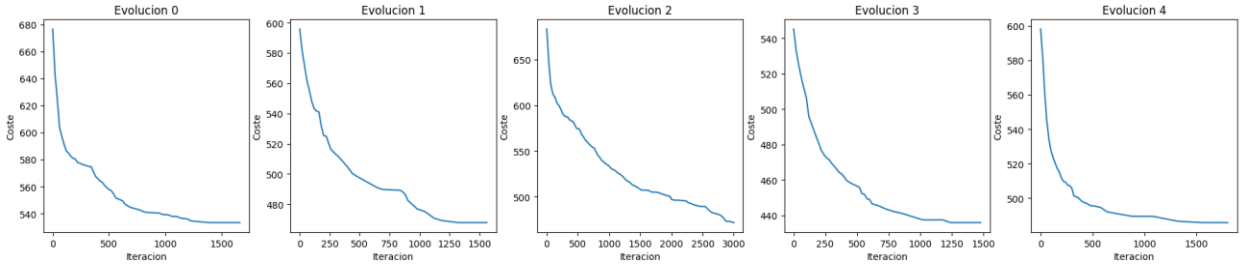
```
-----------------
Semilla: 646
-----------------

Inicial
Solucion: [13 19  9 19 15 13 15  7 13 19 13 13 19  7 11  3]     Coste: 676.5805665744103


Final
Solucion: [13 11 21 19 19 13 11  5 11 15 13 23  3 15  5 11]     Coste: 533.2590558410366

Iteraciones: 1660
```

```
-----------------
Semilla: 424
-----------------

Inicial
Solucion: [25 22 10  5 20  7 12 10  5 10 20 25 12 12 12  7]     Coste: 595.8065886270476


Final
Solucion: [15 12 12 15 20 15 14 12 17  8 22 25  2 10  2 13]     Coste: 467.81852805068684

Iteraciones: 1560
```

```
-----------------
Semilla: 615
-----------------

Inicial
Solucion: [12 19  7  9  9 14 16 19  4  7 24 16 12 14 21  9]     Coste: 683.4747897036737


Final
Solucion: [12 11 17 15 17 14 14  5 12 15 16 24  2 18  7 13]     Coste: 471.61605378357

Iteraciones: 3000
```

```
-----------------
Semilla: -450
-----------------

Inicial
Solucion: [ 6 22 19 15 13  8 11 17 22  4 11 22  6 15  8 15]     Coste: 545.2551313109191


Final
Solucion: [12  8 21 15 19 12 13 13 12 14 11 24  4 15 10 11]     Coste: 435.93589543465055

Iteraciones: 1480
```

```
-----------------
Semilla: 710
-----------------

Inicial
Solucion: [14 18 16  9 16 11 18  3 12 11 14 14  3 16 18 18]     Coste: 598.1162310168735


Final
Solucion: [12 12 20 17 18 13 12  5 12 13 14 24  3 12 12 12]     Coste: 485.7822627358864

Iteraciones: 1800
```



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: 0.0019803047180175 78 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:

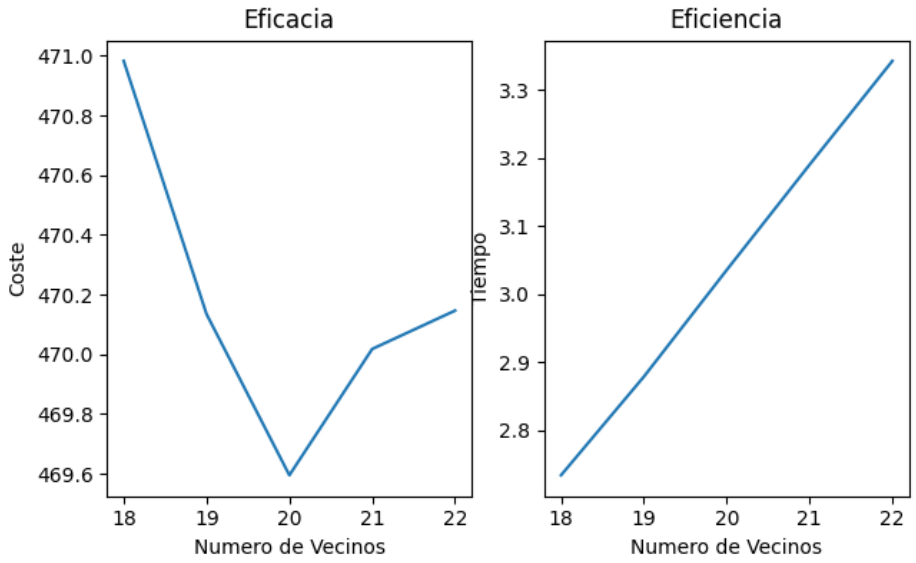
240 permutaciones * 40 posibles cambios * tiempo evaluación * 60 mejoras / 60 segundos: 19.01092529296875 minutos

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

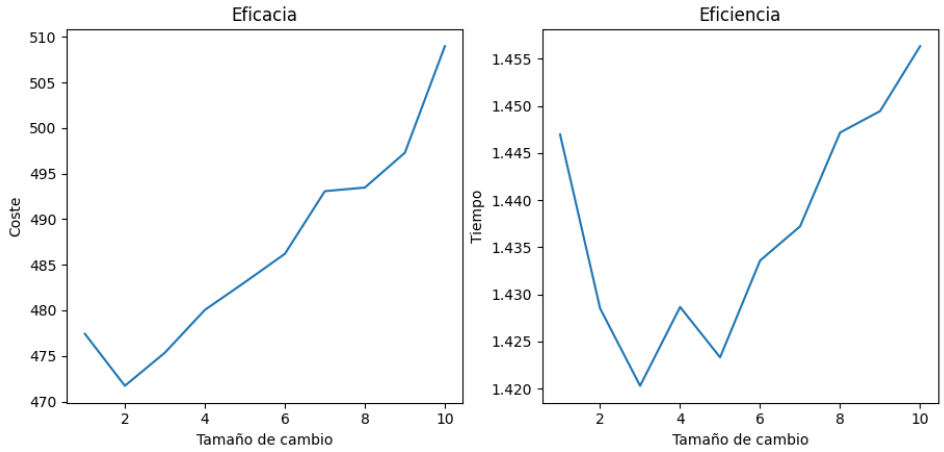
Vamos a variar la velocidad de enfriamiento (número de vecinos generados) para ver cómo afecta al coste



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

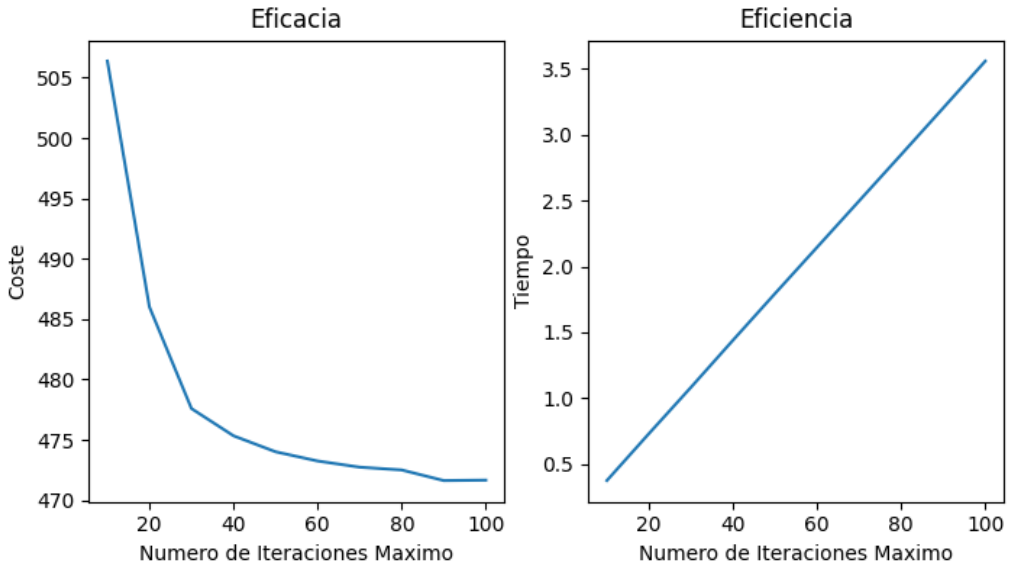
Tamaño de cambio vecino

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

```
ENFRIAMIENTO SIMULADO

Parametros:

Numero de vecinos: 20
Tamaño de cambio: 2
Iteraciones maximas: 90
```

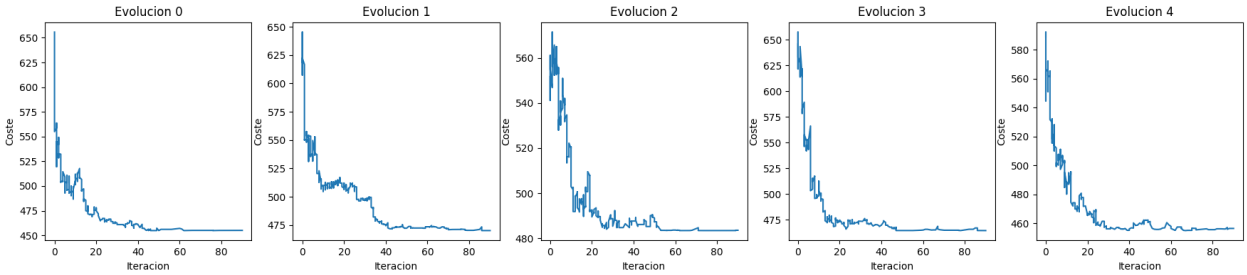
```
-----------------
Semilla: -175
-----------------

Inicial
Solucion: [15  5 26 23 13 20 18 10 13 13 10  7 18 10  7  5]     Coste: 655.7549610189388


Final
Solucion: [11 12 20 21 19 14 12  8 13 15 14 21  2 12  6 13]     Coste: 455.007604786749
```

```
-----------------
Semilla: -276
-----------------

Inicial
Solucion: [16 18 18 13  9  9 22 22  9  4 13 13  4 11 11 20]     Coste: 618.6244393934463


Final
Solucion: [16 12 20 17 19 15 12 10 13 14 11 17  2 18  6 10]     Coste: 470.0999173665895
```

```
-----------------
Semilla: 892
-----------------

Inicial
Solucion: [10 12 10 19 17 10 10 10  8 14 21 21 10 12  6 21]     Coste: 551.7333387587904


Final
Solucion: [16 12 20 15 19 14 14  8 12 16 11 17  2 22  2 11]     Coste: 483.55159370866664
```

```
------------------
Semilla: -539
------------------

Inicial
Solucion: [12 14  6 20 18  8 12  4 12 14 16  6 18 20 14 18]     Coste: 645.7752206436885


Final
Solucion: [12 12 18 16 20 14 12 12 12 14 10 22  2 14 10 12]     Coste: 464.46625629814474
```

```
------------------
Semilla: -48
------------------

Inicial
Solucion: [ 6 19  4 19 19 19  6  8 19 10 15 19  4 19 12 15]     Coste: 589.0504651807174


Final
Solucion: [12  7 18 15 19 13 12 12 13 16 11 23  6 23  2 11]     Coste: 456.4977126570474
```



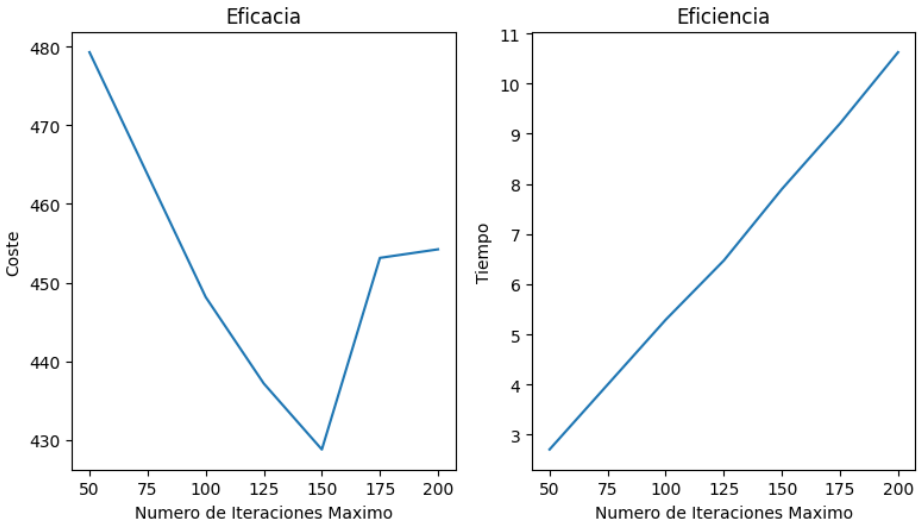
BÚSQUEDA TABÚ

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.

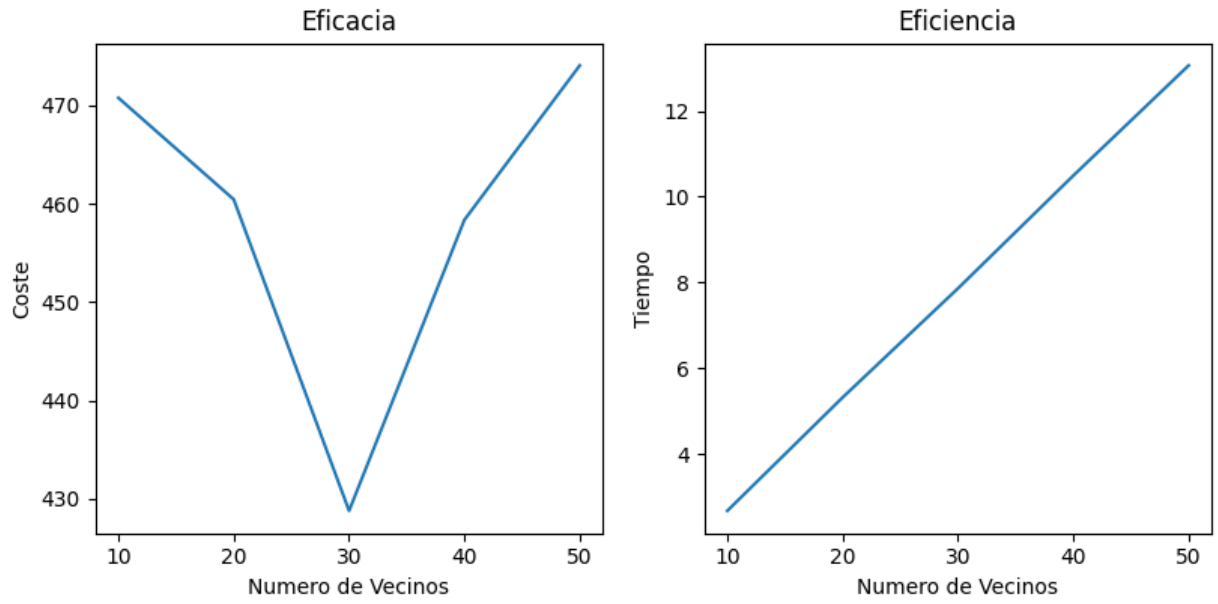
La tabla tabú tiene los movimientos restringidos y la tabla de tiempos tabú, el tiempo que se aplican estas restricciones

Máximas iteraciones



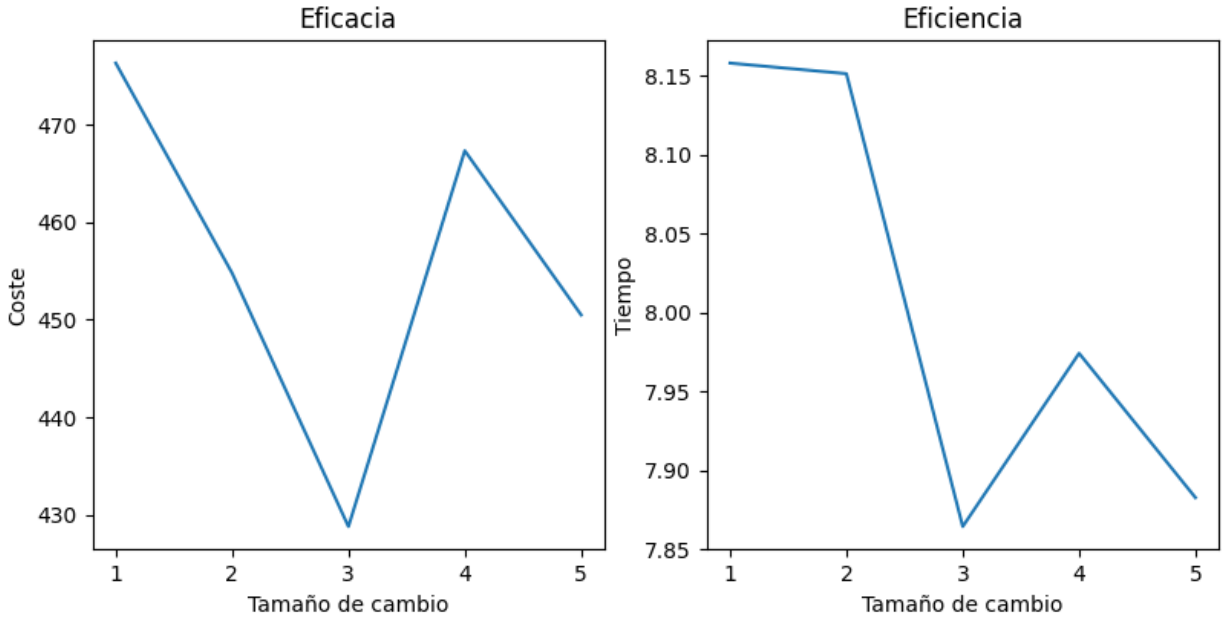
La mejor solución es con 150 iteraciones máximas, a una media de 8 segundos por semilla

Número de vecinos



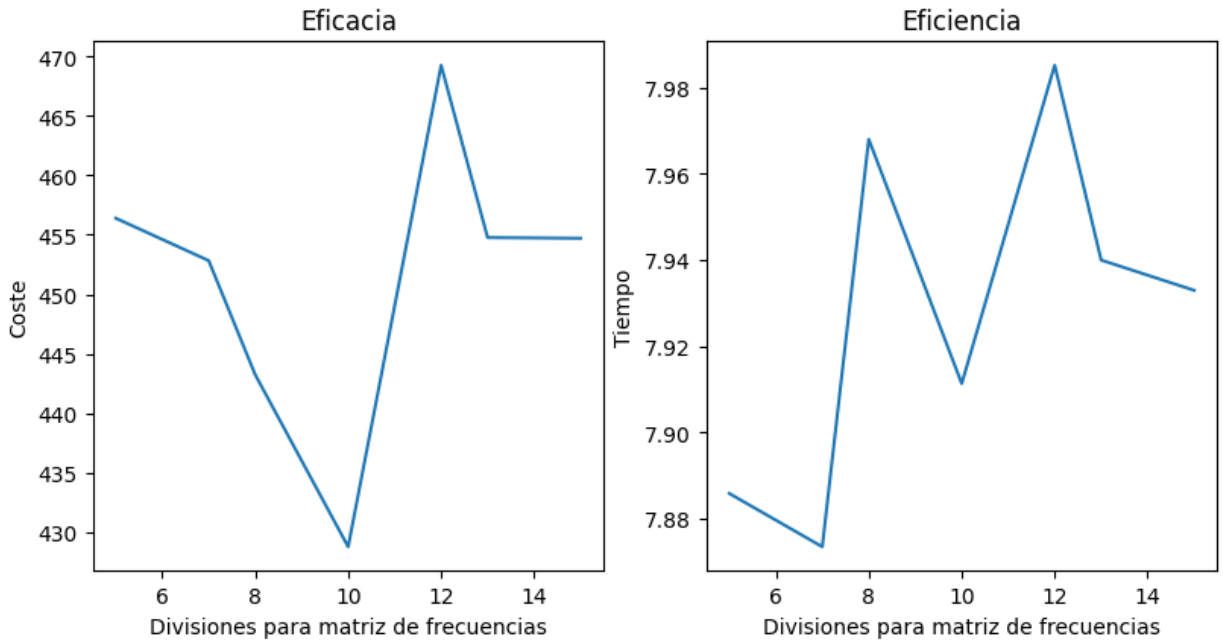
El mejor número de vecinos a generar por iteración es 30.

Número de plazas por cambio



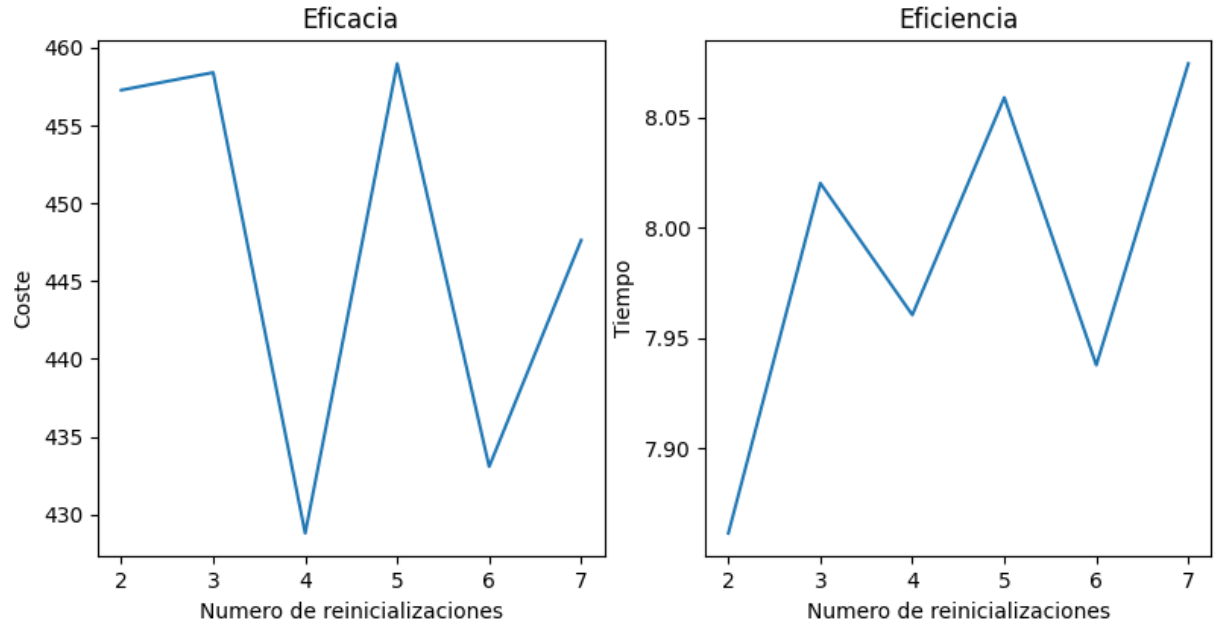
Como podemos ver el número óptimo de plazas cambiadas por generación de vecino es 3.

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

```
BUSQUEDA TABU

Parametros:

Numero de vecinos: 30
Tamaño de cambio: 3
Iteraciones maximas: 150
Division para matriz de frecuencias: 10
Numero de reiniciaciones: 4
```

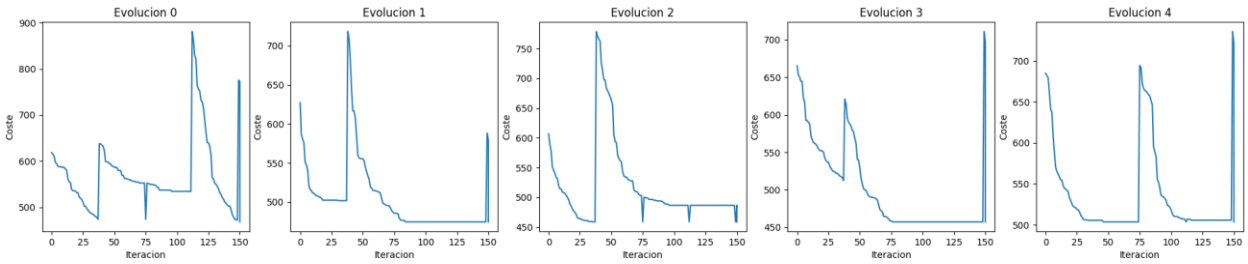
```
-----------------
Semilla: -276
-----------------

Inicial
Solucion: [16 18 18 13  9  9 22 22  9  4 13 13  4 11 11 20]     Coste: 618.6244393934463

Iteracion: 37, Decrementamos lista tabu (2), Reinicializamos con memoria a largo plazo
Iteracion: 74, Incrementamos lista tabu (4), Reinicializamos desde la mejor solucion
Iteracion: 111, Incrementamos lista tabu (8), Reinicializamos con memoria a largo plazo
Iteracion: 148, Incrementamos lista tabu (16), Reinicializamos con memoria a largo plazo

Final
Solucion: [15 12 14 22 20  9 12 10 12 11 11 21  5 17  7 15]     Coste: 468.1916666782713
```

```
-----------------
Semilla: -866
-----------------

Inicial
Solucion: [12 12 14 20 18 12 16 14 12 14 12  4 10 16  4 20]     Coste: 627.2257214081047

Iteracion: 37, Incrementamos lista tabu (8), Reinicializamos con solucion aleatoria
Iteracion: 74, Incrementamos lista tabu (16), Reinicializamos desde la mejor solucion
Iteracion: 111, Decrementamos lista tabu (8), Reinicializamos desde la mejor solucion
Iteracion: 148, Decrementamos lista tabu (4), Reinicializamos con solucion aleatoria

Final
Solucion: [ 9 12 21 19 18 13 15  8 11 14 13 23  3 14  8 11]     Coste: 474.7481794568191
```

```
-----------------
Semilla: 739
-----------------

Inicial
Solucion: [11 15 20  9  9  4  6 18 18 11  9 18 22 15 15 13]     Coste: 606.8372900234123

Iteracion: 37, Decrementamos lista tabu (2), Reinicializamos con memoria a largo plazo
Iteracion: 74, Incrementamos lista tabu (4), Reinicializamos desde la mejor solucion
Iteracion: 111, Decrementamos lista tabu (2), Reinicializamos desde la mejor solucion
Iteracion: 148, Incrementamos lista tabu (4), Reinicializamos desde la mejor solucion

Final
Solucion: [14 12 23 15 21 13 12  9 12 14 12 18  4  9 15 10]     Coste: 458.97575282629555
```

```
------------------
Semilla: 112
------------------

Inicial
Solucion: [17 11 17 15  7 13  9 19 17  3 15  9 17 17 11 13]     Coste: 665.4523496579687

Iteracion: 37, Decrementamos lista tabu (2), Reinicializamos con solucion aleatoria
Iteracion: 74, Decrementamos lista tabu (1), Reinicializamos desde la mejor solucion
Iteracion: 111, Incrementamos lista tabu (2), Reinicializamos desde la mejor solucion
Iteracion: 148, Decrementamos lista tabu (1), Reinicializamos con memoria a largo plazo

Final
Solucion: [13 12 20 18 19 15 12 11 14 14 12 18  2 14  9 10]     Coste: 457.2329068430243
```

```
------------------
Semilla: -246
------------------

Inicial
Solucion: [ 8  8  4 16 18 10 12 10 14 14 20  6 20 16 14 20]     Coste: 684.8570869796081

Iteracion: 37, Decrementamos lista tabu (2), Reinicializamos desde la mejor solucion
Iteracion: 74, Decrementamos lista tabu (1), Reinicializamos con memoria a largo plazo
Iteracion: 111, Reinicializamos desde la mejor solucion
Iteracion: 148, Incrementamos lista tabu (2), Reinicializamos con memoria a largo plazo

Final
Solucion: [17 11 22 22 18 13 12 10 11 14 11 18  2 13  5 11]     Coste: 503.59930646138747
```



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.