QAP mediante algoritmos de búsqueda local

Grupo1: Naroa Iparraguirre, Xiomara Cáceres e Iraida Abad

January 2025

En este documento se muestran las respuestas obtenidas de los distintos algoritmos implementados para resolver el problema QAP.

1 Pseudocódigo

1.1 Random Search

```
randomSearch(solucion_ini, valor_ini, max_iter, size):

    optimo = solucion_ini
    val_optimo = valor_ini

    mientras i < max_iter:
        vecino = random.permitation(size)
        val_vecino = funcionObjetivo(vecino)
        if val_vecino <= val_optimo:
            val_optimo = val_vecino
            optimo = vecino

    return optimo, val_optimo
```

1.2 Best First

```
bestFirst(solucion_ini, valor_ini, max_iter):

    vecindad = calcularVecinos(solucion_ini)
    #devuelve una lista con los vecinos de solucion_ini
    #que se pueden crear con inserts de un solo movimiento
    por vecino en vecindad:
        val_vecino = funcionObjetivo(vecino)
        if val_vecino <= valor_ini:
            return vecino, val_vecino

    return solucion_ini, valor_ini
```

1.3 Greedy

```
greedy(solucion_ini, valor_ini, max_iter):
    optimo = solucion_ini
    val_optimo = valor_ini
    vecindad = calcularVecinos(solucion_ini)

    por vecino en vecindad:
        val_vecino = funcionObjetivo(vecino)
        if val_vecino <= val_optimo:
            val_optimo = val_vecino
            optimo = vecino

    return optimo, val_optimo
```

1.4 Random BL

```
randomBL(solucion_ini, valor_ini, max_iter):
    optimo = solucion_ini
    val_optimo = valor_ini
    vecindad = calcularVecinos(solucion_ini)

    mientras vecindad no este vacia:
        vecino = vecindad.random()
        vecindad.borrar(vecino)
        val_vecino = funcionObjetivo(vecino)
        if val_vecino <= val_optimo:
            val_optimo = val_vecino
            optimo = vecino

    return optimo, val_optimo
```

2 Respuestas

NOTA: para poder evaluar los algoritmos de manera más "justa", se han limitado las iteraciones del algoritmo random search a 361, que es la cantidad de vecinos que tienen que recorrer los algoritmos greedy y random BL.

2.0.1 Tabla de respuestas

La media de las respuestas se ha obtenido ejecutando 30 iteraciones.

Random Search	steps time best	361 0.17789 s 828721.334
Best First	Avg. steps Avg. time Avg. best	10.13 0.00399 s 880909.267
Greedy	Avg. steps Avg. time Avg. best	361 0.14225 s 844288.4
Random BL	Avg. steps Avg. time Avg. best	361 0.15422 s 844288.4

2.0.2 Representación gráfica

En los siguientes gráficos podemos observar el comportamiento de los distintos algoritmos. Como se puede ver el algoritmo greedy y el random BL acaban convergiendo, ya que ambos buscan entre todas las posibilidades del vecindario, mientras que el best first siempre se queda con la primera respuesta que mejore la inicial.

En la mayoría de los casos, el greedy y el random BL no llegan a encontrar un óptimo tan "bueno" como el random search, ya que este último no está limitado por la vecindad de la solución inicial.
Aun así, hemos encontrado un caso en el que los algoritmos greedy y random BL consiguen una respuesta mejor que la que consigue el random search *(Figura2)*.

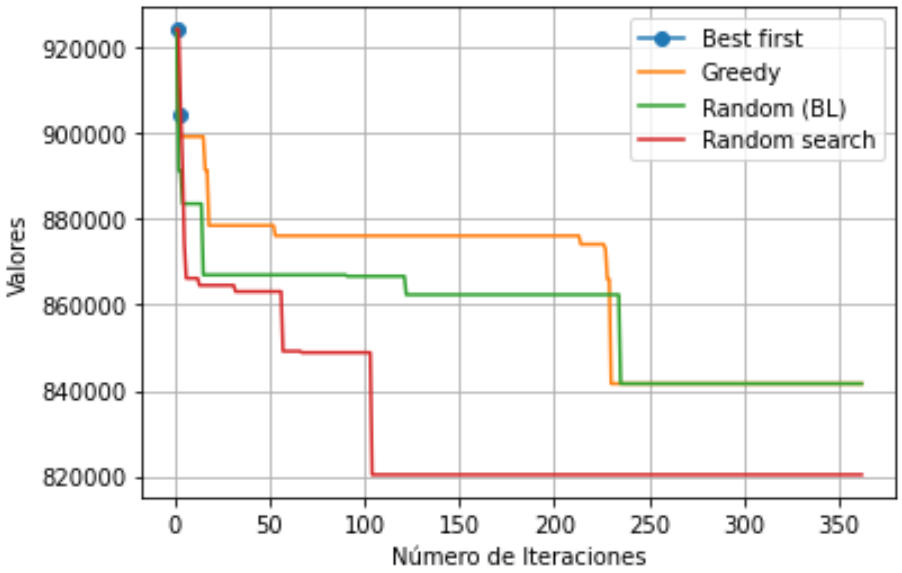

Figure 1: imagen 1

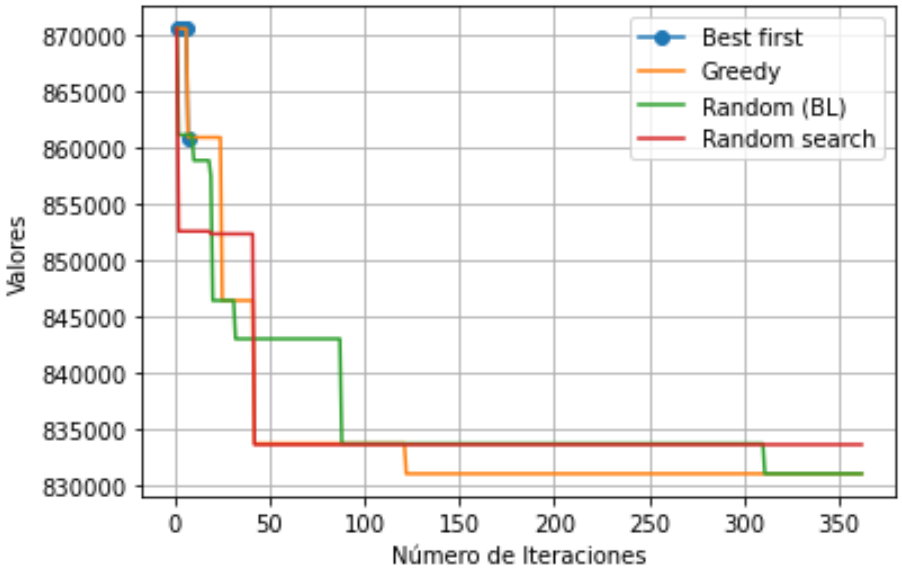

Figure 2: imagen 2

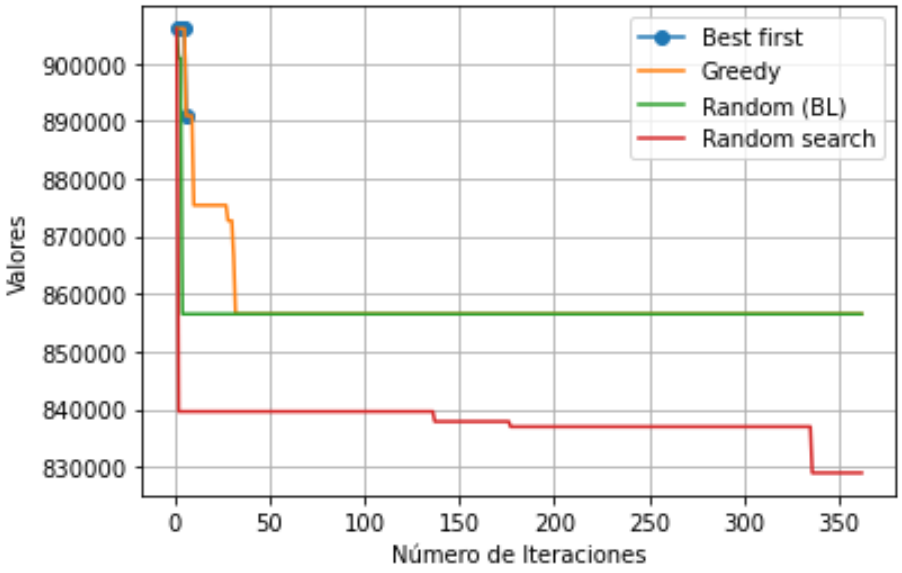

Figure 3: imagen 3