Tarea 2

Evaluación de operadores para problema de QAP con Tabu Search

Gustavo Aguilar[1], Juan Barrerra[2], Juan Trujillo[3]

I. Descripción metaheurística implementada

La búsqueda tabú es un método de optimización matemática, perteneciente a la clase de técnicas de búsqueda local. La búsqueda tabú aumenta el rendimiento del método de búsqueda local mediante el uso de estructuras de memoria: una vez que una potencial solución es determinada, se la marca como "tabú"de modo que el algoritmo no vuelva a visitar esa posible solución.

La búsqueda Tabú surge, en un intento de dotar de "inteligencia" a los algoritmos de búsqueda local. Según Fred Glover, su primer definidor, "la búsqueda tabú guía un procedimiento de búsqueda local para explorar el espacio de soluciones más allá del óptimo local". La búsqueda tabú toma de la Inteligencia Artificial el concepto de memoria y lo implementa mediante estructuras simples con el objetivo de dirigir la búsqueda teniendo en cuenta la historia de ésta, es decir, el procedimiento trata de extraer información de lo sucedido y actuar en consecuencia. En este sentido puede decirse que hay un cierto aprendizaje y que la búsqueda es inteligente. La búsqueda tabú permite moverse a una solución aunque no sea tan buena como la actual, de modo que se pueda escapar de óptimos locales y continuar estratégicamente la búsqueda de soluciones aún mejores.

A continuación el algoritmo implementado para la metaheurística de TS (ver código completo en [1]).

Listing 1: Algoritmo Tabu Search

```
Funcion TS_QAP(params[]*)
    s* = solución_inicial
    fit* = fit_inicial
    // matrices cuadradas de largo según s*
    tabu_corto = matriz()
    tabu_largo = matriz()
    Repetir hasta tiempo_exploración_espacio
        ji = Seleccionar trabajo pivote i
        Iterar hasta intensidad_maxima
            Repetir Hasta tiempo_exploració
                n_local
                // búsqueda local, contiene el
                    tipo de operador
                vecinos = BusquedaLocal()
                // p contiene el movimiento que
                    fue permitido y
                // el valor fit que se obtuvo
                p = BuscarVecinosTabu(params2
                    []*)
                fit* = p.fit
                s* = p.sol
                tabu_corto[x,y] =
                    tiempo_prohibicion
            Fin repetir
    Fin repetir
Fin TS_QAP

funcion BuscarVecinosTabu(params[]*)
    min_sv = ordernarSolucionesDescendientes()
    Iterar Hasta min_sv
        // guardar movimiento en memoria de
            largo plazo
        tabu_largo[x,y)]=+ 1
    Iterar Hasta lmin_sv
        // si no está prohibido por memoria de
            corto plazo y largo plazo
        Si tabu_corto[x,y] == 0 & tabu_largo[x,y
            ] >= umbral_largo_plazo
            return {fit, sol, tabu_largo}
        Otro
            // si está prohibido por memoria de
                largo plazo
            Si tabu_largo[x,y] >=
                umbral_largo_plazo
                min *= penalizar
            Si min < fit
                temporal = min
    return {fit, sol, tabu_largo}
Fin BuscarVecinosTabu

Nota params[]* se hace referencia a todos los
    parámetros de entradas necesarios para
    ajustar la experimentación.
```

A. Parámetros de ajuste

A continuación, se describen los parámetros que utilizará la metaheurística implementada en la presente experiencia:

Variable	Descripción
Tiempo Local	Corresponde al número de iteraciones que va a realizar para buscar un conjunto de vecinos.
Tiempo Espacio	Corresponde al número de iteraciones para realizar una búsqueda diversificada.
Intensidad	Valor que indica que el tamaño o tasa de cambio del operador.
Memoria Corta	Tiempo de iteraciones no permitidas para considerar un movimiento.
Memoria Larga	Tiempo de iteraciones como umbral, en la que se comenzará a prohibir el movimiento, hasta el fin del algoritmo.
Penalización	Valor porcentual por la cual se va a decrementar el valor fit encontrado.
NVecinos	Corresponde al número de vecinos a considerar en la búsqueda local.

Tabla I: Descripción de parámetros de ajustes para el algoritmo de Tabu Search.

II. Descripción de operadores implementados

A. Swap

Para este operador se aplica un pequeño algoritmo, con el fin de generar una colección de soluciones vecinas, de forma ordenada e intensificada.

[1]Dpto Ingeniería Informática, Universidad Santiago de Chile, e-mail: gustavo.aguilar@usach.cl
[2]Dpto Ingeniería Informática, Universidad Santiago de Chile, e-mail: juan.barrerab@usach.cl
[3]Dpto Ingeniería Informática, Universidad Santiago de Chile, e-mail: juan.trujillo@usach.cl

Listing 2: Algoritmo Swap

```
    Iterar Hasta intensidad_maxima
        Iterar Hasta n_vecinos
            // i posición pivote y la intensidad
                , corresponde contra que posici
                ón.
            // se va a realizar el cambio de
                elemento.
            vecino = swap(i, i+intensidad)
        Fin Iterar
    Fin Iterar
```

Un ejemplo, es considerar un arreglo de soluciones como en (1)

$$s = [a, b, c, d, e, f] \tag{1}$$

Luego para la primera iteración, se tendrá una intensidad 1, por lo que se tendrán soluciones vecinas como en (2).

$$\begin{aligned} s1 &= [b, a, c, d, e, f] \\ s2 &= [a, c, b, d, e, f] \\ s3 &= [a, b, d, c, e, f] \\ &\dots \end{aligned} \tag{2}$$

Posteriormente para una intensidad mayor, se va a considerar un swap de largo 2 o más, como en el ejemplo (5).

$$\begin{aligned} s1 &= [c, b, a, d, e, f] \\ s2 &= [a, d, c, b, e, f] \\ s3 &= [a, b, e, d, c, f] \\ &\dots \end{aligned} \tag{3}$$

B. Insert

El segundo operador que será utilizado será el operador insert, el cual opera de la siguiente manera:

1. Se elige un elemento, el cual se mueve a otra posición cualquiera de la lista.
2. Dependiendo de en que sentido se mueva el elemento, el resto de los elementos se acomodan en el sentido contrario con el fin de ocupar el espacio vacío.

A continuación se muestran dos ejemplos de la utilización del operador insert, aplicados al arreglo (1):

1. Si se mueve el segundo elemento (b) a la quinta posición, este es el resultado:

$$s1 = [a, c, d, e, b, f] \tag{4}$$

2. Si se mueve el cuarto elemento (d) a la primera posición, este es el resultado:

$$s2 = [d, a, b, c, e, f] \tag{5}$$

C. Inverse

El tercer operador inverse, corresponde en principio a seleccionar una parte de la solución e invertir la posición de sus elementos y volver a colocar estos, en ese mismo espacio seleccionado. Esto se puede representar como en (7)

$$s1 = [a, b, c, d, e, f] \tag{6}$$

Al seleccionar los elementos $[b, c, d, e]$, entonces al invertir el orden, queda como $[e, d, c, b]$, por lo que la nueva solución queda como:

$$s2 = [a, e, d, c, b, f] \tag{7}$$

Para este caso, la intensidad va a variar el largo de la selección de los elementos, y así como también se varia la posición inicial de selección.

III. Descripción de las pruebas computacionales

A continuación se presenta un diagrama de flujo (ver figura 1) que representa las pruebas computacionales que se efectuaron

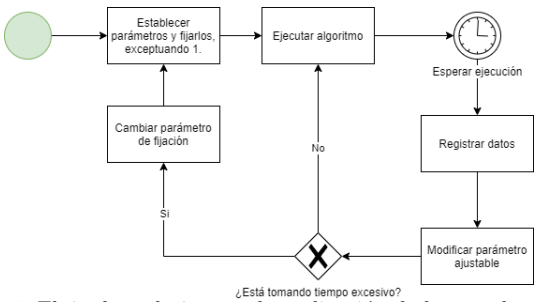

Fig. 1: Flujo de trabajo para la realización de las pruebas del algoritmo de TS, con diferentes operadores.

De acuerdo a lo descrito anteriormente, notamos que para tipo de operador, se debe ejecutar el flujo, con el fin de generar una colección de resultados.
Notar que para control de decisión ilustrado en la figura 1, se tiene como tope un cierto criterio de tiempo de ejecución, ya que para valores extremos de los parámetros de ajuste, este se extiende abruptamente y por lo tanto se considera el flujo alterno, en la que se modifican el/los parámetros fijos.

IV. Resultados y Discusión

A continuación se presenta una recopilación de los resultados obtenidos para cada uno de los operadores de la meta heurística de Tabu Search.
En donde: Tloc = Tiempo local, Tesp = Tiempo espacio, Int = Intensidad, MC = memoria corto plazo, ML = Memoria largo plazo, Pen = Penalización, Nvec = Número de vecinos, T. Ex. = Tiempo Excedido.

En la siguiente tabla, se presentan los resultados con el operador swap. Como se puede observar, los valores del fitness oscilan entre 5444024 y 5712063:

Tloc	Tesp	Int	MC	ML	Pen	Nvec	Fitness
3	10	10	5	10	0.5	25	5447705
3	10	10	5	10	0.5	20	5484343
3	10	10	5	10	0.5	15	5493118
3	10	10	5	10	0.5	10	5625487
5	10	10	5	10	0.5	25	5476591
5	10	3	5	10	0.5	25	5479093
5	10	5	5	10	0.5	25	5535741
5	10	7	5	10	0.5	25	5472241
5	10	12	5	10	0.5	25	5532472
5	10	15	5	10	0.5	25	5485875
5	10	20	5	10	0.5	25	5459086
3	10	3	5	5	0.5	25	5538160
3	10	3	5	8	0.5	25	5572179
3	10	3	5	10	0.5	25	5712063
3	10	3	5	15	0.5	25	5529541
3	3	10	5	10	0.5	25	5491389
3	5	10	5	10	0.5	25	5444024

En la siguiente tabla, se presentan los resultados con el operador insert. Como se puede observar, los valores del fitness oscilan entre 5497017 y 6072182:

Tloc	Tesp	Int	MC	ML	Pen	Nvec	Fitness
3	10	10	5	10	0.5	25	5497017
3	10	10	5	10	0.5	20	5597595
3	10	10	5	10	0.5	15	5727373
3	10	10	5	10	0.5	10	5734520
5	10	10	5	10	0.5	25	5864521
5	10	3	5	10	0.5	25	5704130
5	10	5	5	10	0.5	25	5878584
5	10	7	5	10	0.5	25	5761074
5	10	12	5	10	0.5	25	5739560
5	10	15	5	10	0.5	25	5872948
5	10	20	5	10	0.5	25	5685805
3	10	3	5	5	0.5	25	5725936
3	10	3	5	8	0.5	25	5894466
3	10	3	5	10	0.5	25	5695284
3	10	3	5	15	0.5	25	5658939
3	3	10	5	10	0.5	25	5858979
3	5	10	5	10	0.5	25	6072182

En la siguiente tabla, se presentan los resultados con el operador inverse. Como se puede observar, los valores del fitness oscilan entre 5568617 y 6146358:

Tloc	Tesp	Int	MC	ML	Pen	Nvec	Fitness
3	10	10	5	10	0.5	25	5999522
3	10	10	5	10	0.5	20	5989836
3	10	10	5	10	0.5	15	6013215
3	10	10	5	10	0.5	10	5923890
5	10	10	5	10	0.5	25	5883947
5	10	3	5	10	0.5	25	T. Ex.
5	10	5	5	10	0.5	25	6068359
5	10	7	5	10	0.5	25	6146358
5	10	12	5	10	0.5	25	5861372
5	10	15	5	10	0.5	25	5804495
5	10	20	5	10	0.5	25	5853406
3	10	3	5	5	0.5	25	5963582
3	10	3	5	8	0.5	25	5957331
3	10	3	5	10	0.5	25	6097720
3	10	3	5	15	0.5	25	5967265
3	3	10	5	10	0.5	25	5568617
3	5	10	5	10	0.5	25	5633072

De las tablas anteriores, tenemos que la mejor solución encontrada para las diferentes instancias de parámetros fue 5444024, lograda utilizando el operador swap. En general, se observa que las soluciones tienen a ser mejores usando este operador. También es posible plantear preliminarmente que las soluciones mejoran o empeoran variando ciertos parámetros. Por ejemplo, las soluciones parecen empeorar al disminuir el valor del parámetro Intensidad. Sin embargo, para determinar con certeza este comportamiento, sería necesaria la realización de un estudio más exhaustivo. De las pruebas realizadas, se logra evidenciar que para cada uno de los operadores, se tiene un perfil que difieren unos con otros de acuerdo a los valores de las soluciones encontradas, como se puede apreciar en las figuras (2) (3) (4).

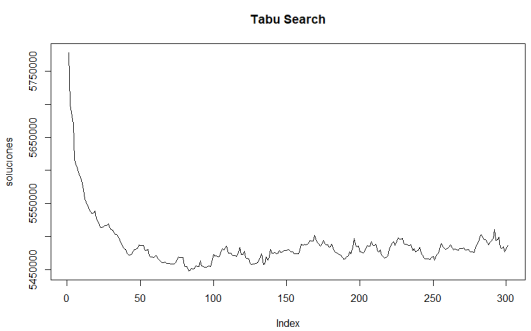

Fig. 2: Perfil de soluciones encontradas para meta heurística tabu search para QAP, con operador swap intensificado.

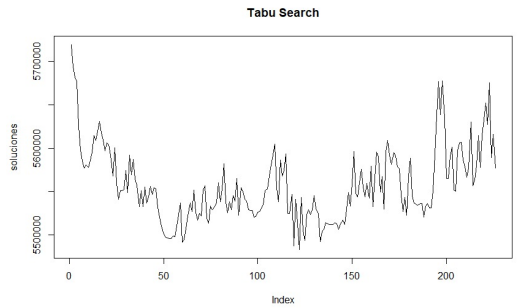

Fig. 3: Perfil de soluciones encontradas para meta heurística tabu search para QAP, con operador insert intensificado.

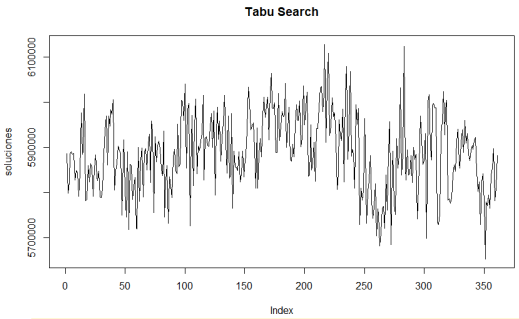

Fig. 4: Perfil de soluciones encontradas para meta heurística tabu search para QAP, con operador invert intensificado.

De acuerdo a la figura (2), podemos ver que rápidamente encuentran soluciones de buena calidad, de acuerdo a los parámetros de ajustes evidenciados en la tablas anteriores. Sin embargo el espacio de diversificación, no se logra apreciar a diferencia de los otros operadores, que se observan picos con diferencias sustanciales en los perfiles.
Respecto a los parámetros de ajuste, podemos destacar que la memoria de corto plazo y la intensificación, mejoran significativamente los resultados a medida que aumentan sus medidas, sin embargo, el costo computacional, también se incrementa.
Por otro lado, la memoria de largo plazo y el tiempo de espacio, sus variaciones no se logran evidenciar mejoras importantes en la búsqueda de soluciones de mejor calidad, e incluso el costo computacional para tiempo de espacio amplio, no garantiza mejores resultados o mejoras significativas.
Ahora bien, dentro del algoritmo de tabu search, es importante considerar el tamaño de los vecinos a evaluar (si es reducido o no), y el conjunto de prohibiciones por la memoria de corto o largo plazo, ya que es probable que se descarten soluciones que permitan explorar otros escenarios. De hecho, en las pruebas realizadas, la cantidad de vecinos fue sustancialmente relevante para encontrar soluciones de mejorar calidad. Sin embargo, el extender este parámetros (por sobre los 25 a 30), se evidencia que el tiempo de procesamiento se incrementa a puntos de cancelar el experimento.
Respecto a la penalización, no se registra cambios importantes par ninguno de los operador.

V. Conclusiones

A partir de los resultados obtenidos, podemos extraer las siguientes conclusiones:

1. Respecto a la exploración del espacio de búsqueda, el operador insert es el que parece mostrar el mejor balance entre intensificación y diversificación. En el swap predomina la intensificación y el inverse parece realizar casi únicamente diversificación.
2. Respecto a la calidad de los resultados, el swap es el operador que entrega un mejor fitness para las diferentes combinaciones de parámetros analizadas. Por otra parte, el inverse es el operador que produce los peores resultados. Esto podría indicar preliminarmente que el problema se resuelve mejor con un algoritmo en que predomine la intensificación.
3. Respecto al rendimiento computacional, la variación del valor algunos parámetros parece afectar notablemente el tiempo de procesamiento para cada operador. Por ejemplo, al aumentar el número de vecinos, los algoritmos aumenta considerablemente su tiempo de procesamiento.
4. Respecto al trabajo en general, se hace muy útil e interesante contar con la posibilidad de encontrar soluciones viables para problemas complejos, en un corto tiempo, utilizando algoritmos relativamente sencillos de implementar y con un lenguaje de programación de alto nivel como R.

Referencias

[1] Gustavo Aguilar, *Tabu search desarrollado en R. https://github.com/naotoam/magister/tree/master/Optimizacion en Ingenieria/tarea2*, 2021.