Técnicas Algorítmicas

Trabajo Práctico Final

Integrantes

Alba Chicar, Agustín Darío

González, Joaquín

Fecha de entrega: 29 de noviembre de 2020

1.- Asignación mediante *backtracking*

El algoritmo consiste en recorrer el espacio de soluciones de forma recursiva. Se puede ver al algoritmo como la generación y recorrido de un árbol donde se encuentran las posibles soluciones al considerar los caminos desde el nodo raíz hasta las hojas más alejadas. Este algoritmo presenta una complejidad temporal $O(n!)$, donde n es el número de estudiantes y/o tópicos.
Para la implementación nos ayudamos de una función auxiliar (`asignar_backtracking_internal()`), que recibe una solución base sobre la que asignaremos los estudiantes y tópicos. Los tres pasos más importantes del algoritmo son los siguientes:

1. Asigno un estudiante *e* con un tópico según las preferencias.
2. Para ese estudiante *e* se resuelve recursivamente el resto de las asignaciones llamando a la misma función `asignar_backtracking_internal()` con la solución parcial anterior. Evalúamos si la solución que retorna `asignar_backtracking_internal()` es una mejor solución.
3. Desasignamos al estudiante *e* y proseguimos con el siguiente

La función retorna la mejor asiganción y realiza una búsqueda global en todo el espacio de soluciones por lo que explora todas las alternativas.

2.- Asignación mediante *heurística greedy*

Esta heurística se encarga de realizar la mejor asignación por orden de iteración de estudiantes, es decir, que para el primer estudiante listado, se buscará dar la mejor asignación, para el segundo estudiante la mejor asignación que se pueda siempre y cuando el primero no la haya tomado, y así sucesivamente.
Este algoritmo tiene una complejidad temporal igual a $O(n^2)$ donde n es el número de estudiantes y/o tópicos.

3.- Asignación mediante *heurística búsqueda local*

Se basa en una noción de vecindad que consiste en lo siguiente: dada una solución definida como el set de tuplas *Solucion = {(estudiante_i, tópico_j), ...}* una solución vecina es aquella solución que se diferencia de la original en *una* permutación de tópicos y estudiantes solamente. Entonces, dada una solución que consta de n pares estudiante - tópico, se buscará obtener una mejor solución en el subespacio de soluciones vecinas, en caso de encontrarse, se vuelve a reiterar este proceso de búsqueda en la vecindad.
El algoritmo tiene el inconveniente que no deriva siempre en un óptimo global, sino uno local por solo mirar la vecindad y avanzar en la dirección que optimiza la solución. No obstante, se recorre un conjunto mayor en el espacio de soluciones comparado con greedy. A pesar de esto, los resultados pueden variar negativamente respecto de greedy ya que se ha optado por utilizar una solución aleatoria como input del algoritmo de búsqueda local.

Para la implementación de esta técnica se realizaron modificaciones sobre la clase Planilla(). En particular, se crearon dos métodos que exponen la asignación (solución) de la

planilla y, también, el tamaño de la misma. Exponiendo estos métodos en la interfaz de la clase se facilitó la implementación de la heurística de búsqueda local.

```python
class Planilla:
    # Para comenzar una nueva planilla, necesitamos conocer las preferencias
    # declaradas por los estudiantes.
    def __init__(self, preferencias):
        self._prefs = preferencias
        self._e2t = dict()  # Mapeo de estudiante a tópico.
        self._t2e = dict()  # Mapeo de tópico a estudiante.
        self.size =  self._prefs._cantidad

    # Metodo para saber el tamano de la clase
    # para poder encontrar soluciones vecinas
    def cantidad(self):
        return self.size

    # Metodo para retornar la solucion de esta plantilla
    def solucion(self):
        return self._e2t
```

4.- Asignación mediante *heurística búsqueda local iterativa*

Como mejora al algoritmo de búsqueda local, se busca llegar a un mejor resultado mediante múltiples entradas aleatorias en el espacio de soluciones. Esto hace que en caso de tener más de un mínimo local, se incrementan las chances de caer en otro y poder evaluar las múltiples soluciones y haber tenido acceso al óptimo global.

5.- Resultados

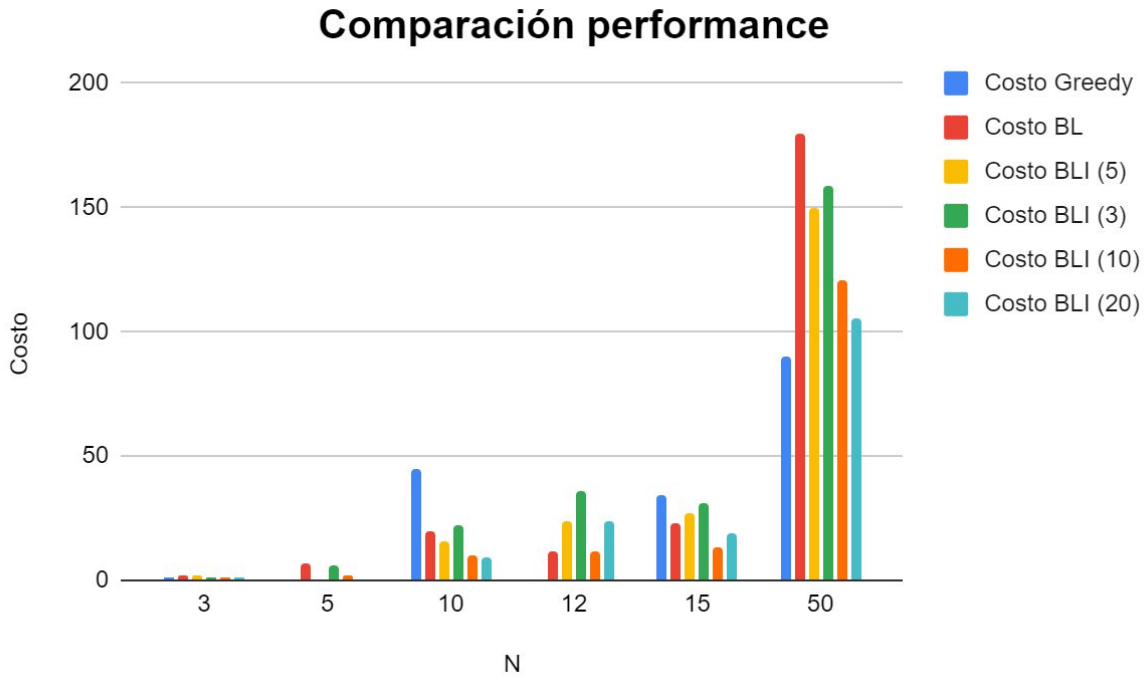
En el caso de BLI, podríamos decir que hay mucha variabilidad entre distintas corridas en términos de costo debido a que se parte de soluciones aleatorias, partiendo de una greedy se podría tener menos varianza pero seguramente mas sesgo. Debido a esto, podemos explicar que, a veces, se tengan soluciones de mayor costo con corridas de iteraciones más altas, ya que se podría haber partido de una solución inicial más “alejada”, en términos de la definición de vecindad desarrollada, de los mínimos en el espacio de soluciones.

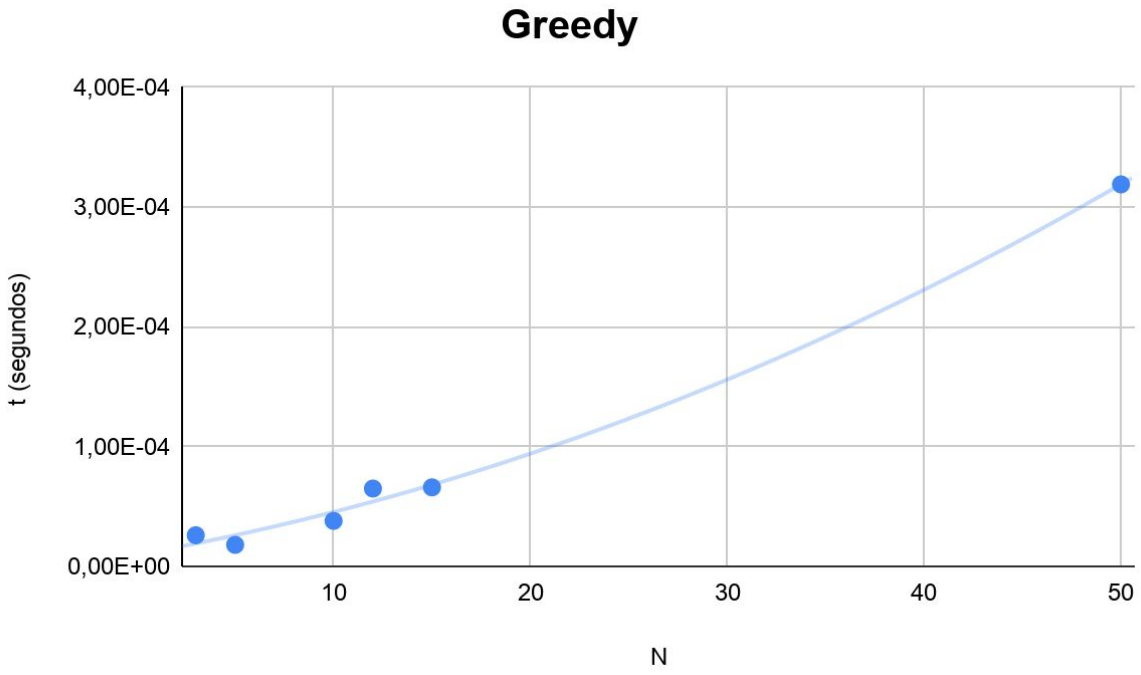
A continuación, se adjuntan tablas con resultados de las simulaciones realizadas para cada algoritmo. Puede observarse en el gráfico comparativo de performance, como BLI es el que ha terminado performando mejor en promedio. Esto es esperable ya que busca en un subconjunto del espacio de soluciones mucho mayor que BL y Greedy. Por otro lado, también se observa una mejora progresiva al ir aumentando el número de iteraciones, aunque esto no es estrictamente lineal por lo explicado anteriormente de que el input es una

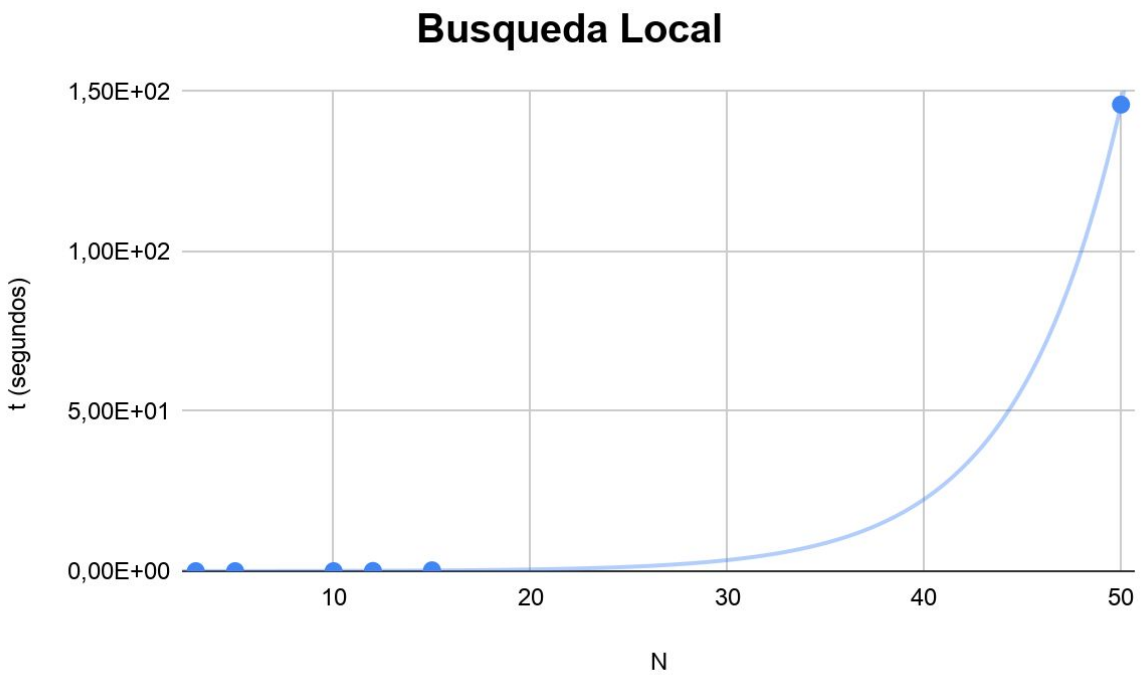
solución aleatoria. Otra variable a analizar respecto de BLI, es que el tiempo de ejecución a medida que aumenta el número de iteraciones crece linealmente.

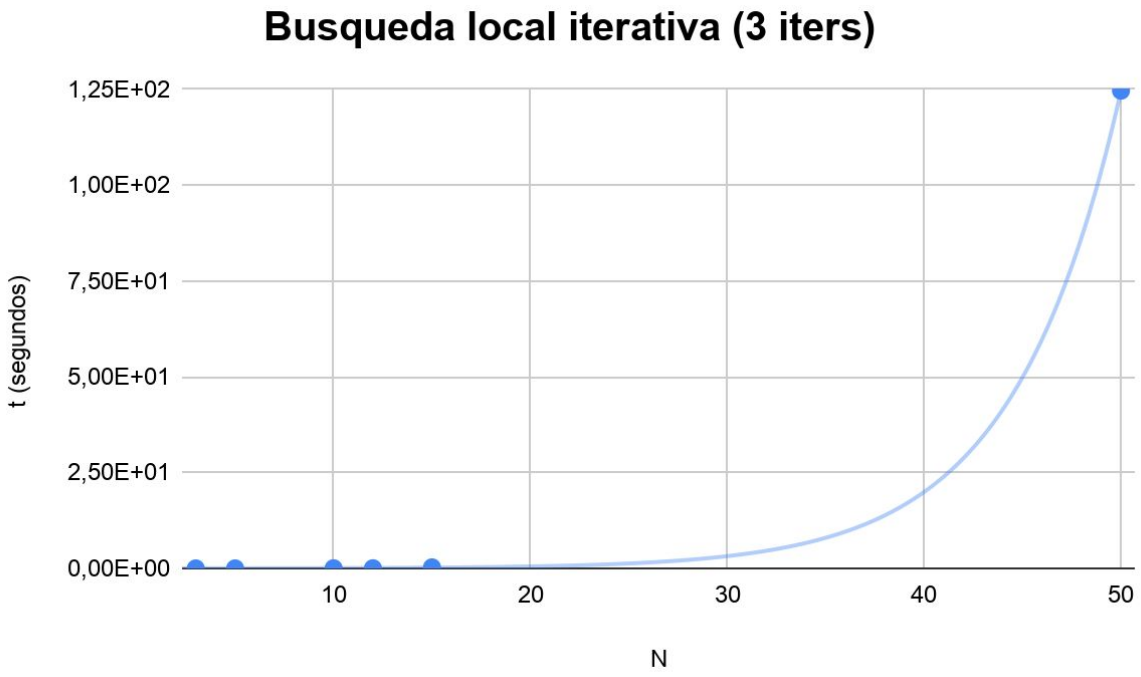
N	Backtracking		Greedy		Búsqueda Local	
	Tiempo	Costo	Tiempo	Costo Greedy	Tiempo	Costo BL
3	0	1	2,60E-05	1	8,00E-06	2
5	1,57E-02	0	1,80E-05	0	6,88E-04	7
10	1,94E+03	1	3,80E-05	45	9,83E-03	20
12	9999	9999	6,50E-05	0	6,89E-02	12
15	9999	9999	6,60E-05	34	2,71E-01	23
50	9999	9999	0,000319	90	1,46E+02	180

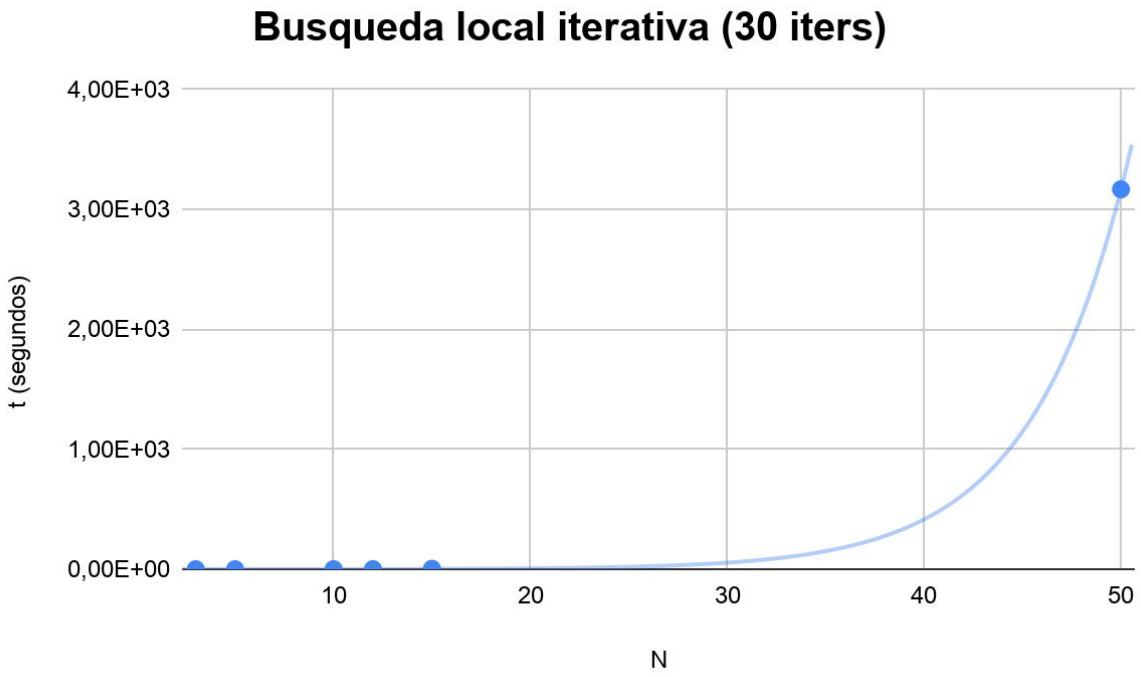
N	BLI (5 iters)		BLI (3 iters)		BLI (10 iters)		BLI (20 iters)		BLI (30 iters)	
	Tiempo	Costo	Tiempo	Costo	Tiempo	Costo	Tiempo	Costo	Tiempo	Costo
3	2,12E-04	2	1,60E-04	1	1,14E-03	1	1,52E-03	1	9,03E-04	1
5	0,001452	0	0,0008787	6	0,003243	2	0,007237	0	1,31E-02	0
10	0,068015	16	0,0409	22	0,14276	10	0,312692	9	5,18E-01	10
12	0,09155	24	0,0386	36	0,147105	12	0,344933	24	6,54E-01	12
15	0,494942	27	0,27708	31	1,01032	13	1,93903	19	3,42E+00	21
50	209,554	150	124,66	159	442,281	121	989,443	105	3,17E+03	127




Greedy
4,00E-04
3,00E-04
2,00E-04
1,00E-04
0,00E+00
t (segundos)
10
20
30
40
50
N


Busqueda Local
1,50E+02
1,00E+02
5,00E+01
0,00E+00
t (segundos)
10
20
30
40
50
N





Nota: las líneas de tendencia se han realizado utilizando funciones cuadráticas para la asignación greedy y exponencial para las asignaciones iterativas de búsqueda local.