Introducción

Los distintos test se han realizado en una máquina que tiene un i7 6700k como procesador y 16 Gb de RAM. Se ha establecido arbitrariamente estos parámetros para las pruebas en las que se necesita fijar un parámetro

Aristas	**1000 (100 para Backtracktracking puro)**
Vértices	**1000 (100 para Backtracktracking puro)**
Visitas	**5**

Debido a que no se podía estar demasiado tiempo realizando los test, y viendo que los tiempos que tenían las ejecuciones con Backtracking puro, se ha decidido reducir el tamaño de los grafos. Aun así, se han tenido que mostrar los gráficos aparte porque las escalas temporales eran muy distintas a la del resto de algoritmos.

En las preguntas 1, 2 y 3 se ha seguido el siguiente esquema:

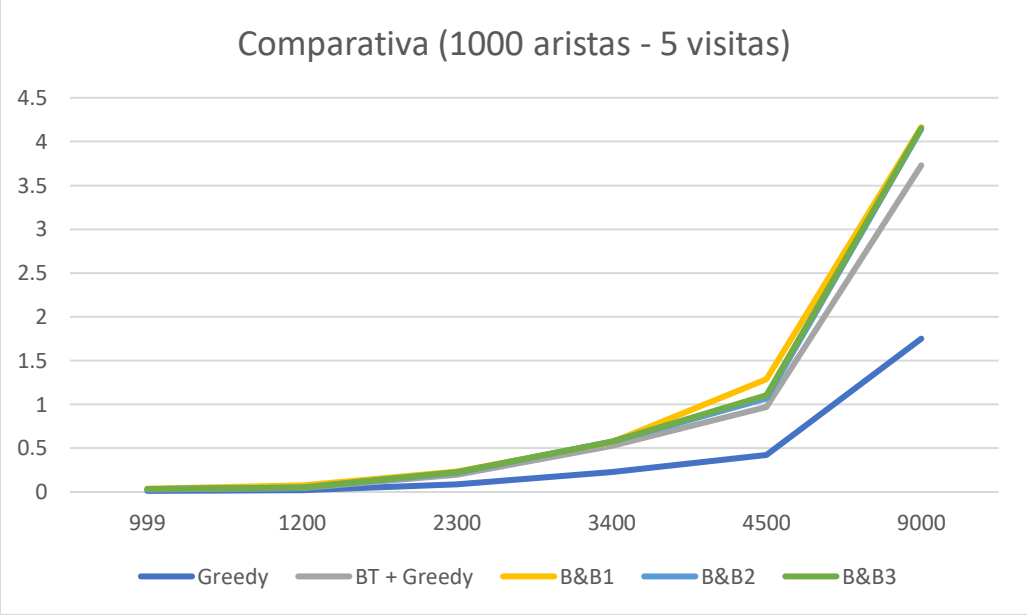
Grafica comparativa + grafica BT (para poder apreciar todos los resultados)

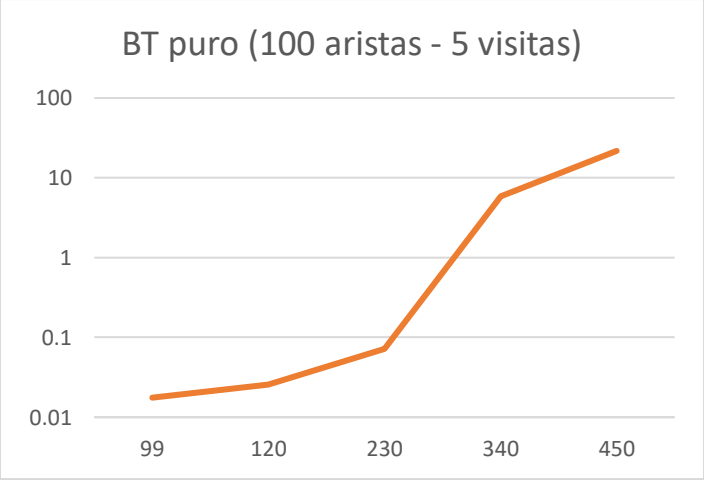
Justificación de los resultados obtenidos.

Tablas de datos y graficas individuales (las mismas usadas en la comparativa, pero con un único algoritmo)

Al analizar se ha vista que lo idóneo sería realizar varios test con cada iteración y luego calcular una media aritmética de estos, pero, por falta de tiempo no se ha realizado. Asumimos que los resultados no son completamente fieles a la realidad ya que solo se han probado con un único grafo para cada caso y se han "ploteado" los resultados obtenidos

1. **¿Cómo varía el tiempo de ejecución de los algoritmos al variar el número de vértices? Constante, logarítmico, lineal, exponencial, factorial, etc.**
   - **Una gráfica con las curvas de los 6 algoritmos. En el caso que no se pueda ver bien alguna curva en la primera gráfica, hacer otra gráfica donde sólo se vean las curvas que no se veían bien.**



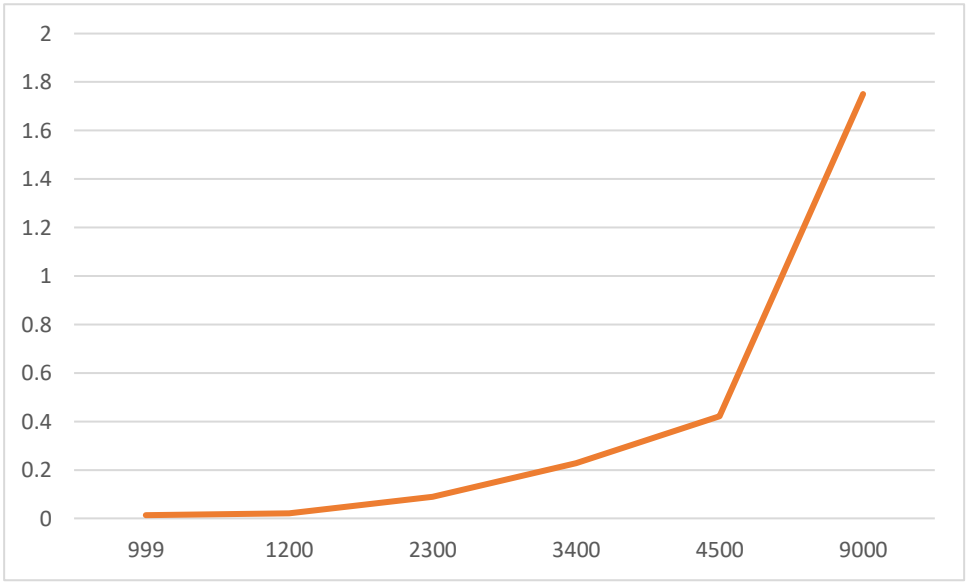


En todos los algoritmos vemos que la curva que se genera al aumentar el número de vértices, dejando fijado el número de aristas (1000 aristas) y el número de visitas (5 visitas) es de carácter **exponencial**.

El algoritmo que mejor soporta el crecimiento de vértices es el Backtracking Greedy, ya que greedy no nos da la solución óptima siempre y lo descartamos como mejor algoritmo.

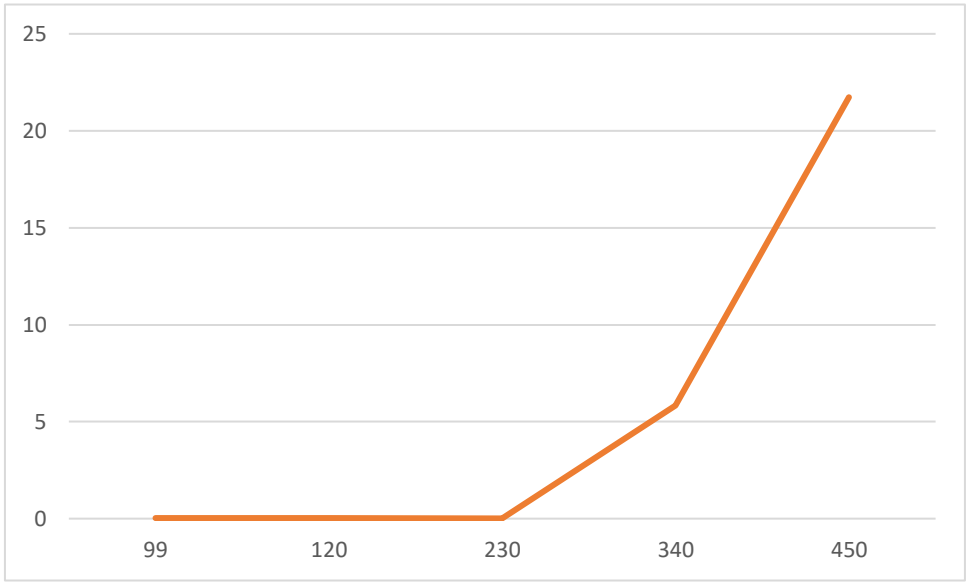
Greedy – 5 visits – 1000 aristas

999	0.0131942
1200	0.0207245
2300	0.0899859
3400	0.227703
4500	0.42209
9000	1.75053



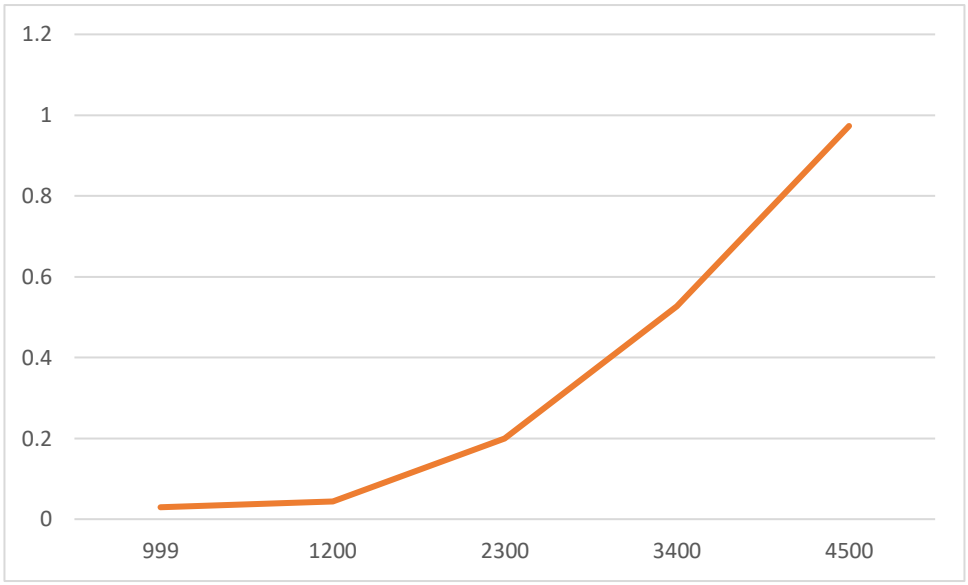
Backtracking puro – 5 visits – 100 aristas

99	0.0175715
120	0.025581
230	0.011484
340	5.84645
450	21.7296
900	1067.18



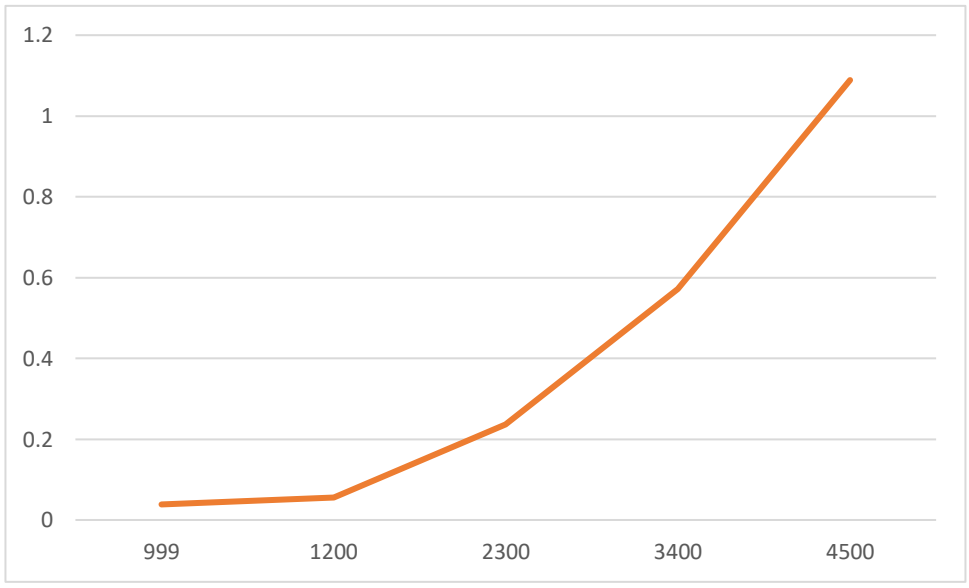
Backtracking greedy – 5 visits – 1000 aristas

999	0.0294972
1200	0.0437994
2300	0.198961
3400	0.52696
4500	0.973212
9000	3.72913



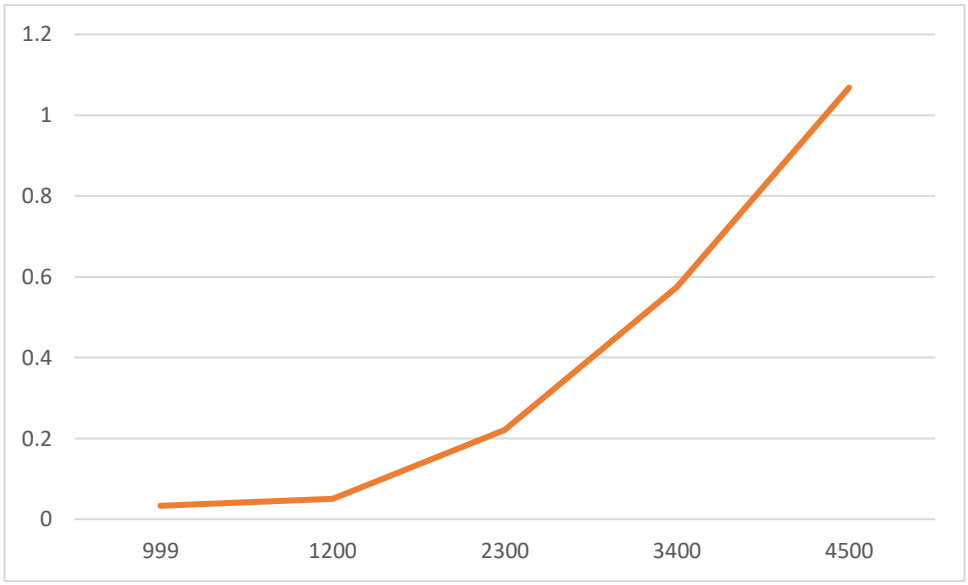
Brach and bound1 – 5 visits – 1000 aristas

999	0.0384781
1200	0.0556615
2300	0.236481
3400	0.571217
4500	1.08887
9000	4.16397



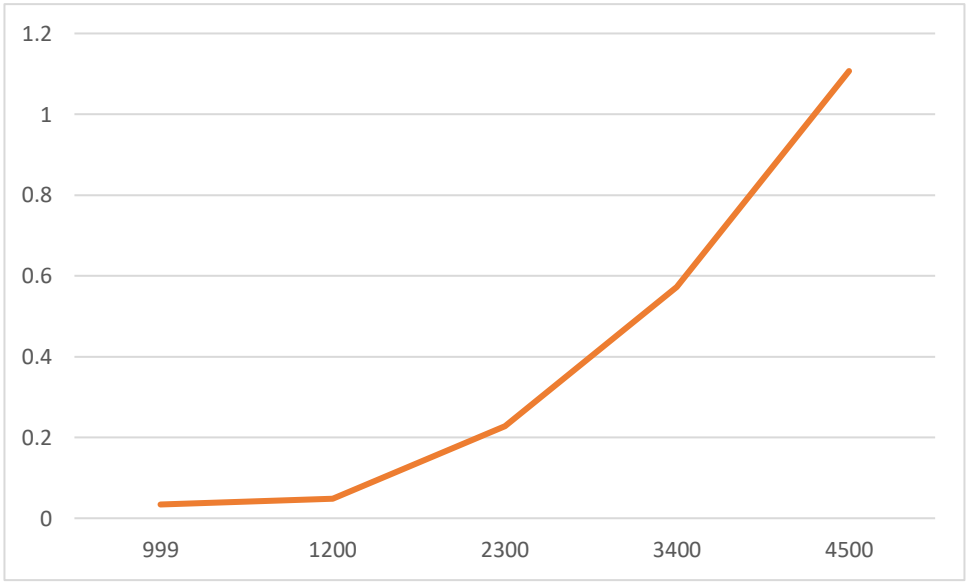
Brach and bound2 – 5 visits – 1000 aristas

999	0.0331563
1200	0.0503928
2300	0.220371
3400	0.574526
4500	1.06818
9000	4.1426



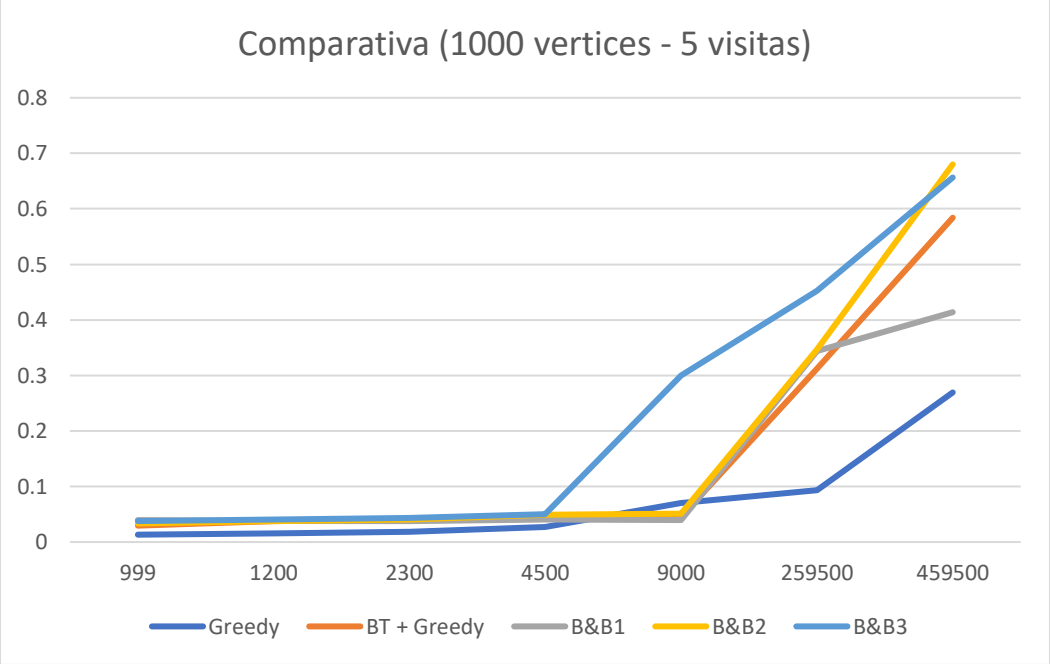
Brach and bound3 – 5 visits – 1000 aristas

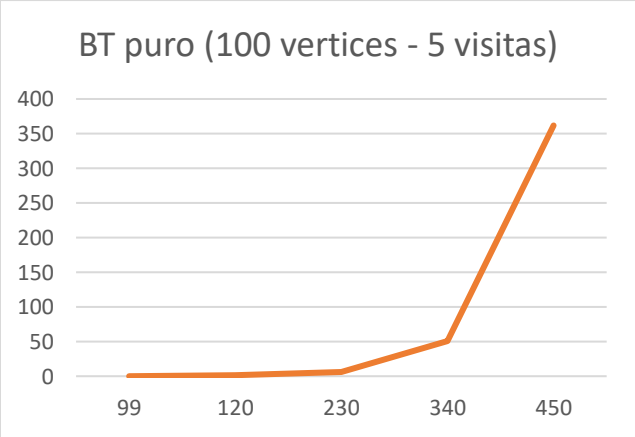
999	0.0340898
1200	0.048772
2300	0.227998
3400	0.572126
4500	1.10697
9000	4.15509



2. ¿Cómo varía el tiempo de ejecución de los algoritmos al variar el número de aristas? Constante, logarítmico, lineal, exponencial, factorial, etc.

- **Una gráfica con las curvas de los 6 algoritmos. En el caso que no se pueda ver bien alguna curva en la primera gráfica, hacer otra gráfica donde sólo se vean las curvas que no se veían bien.**





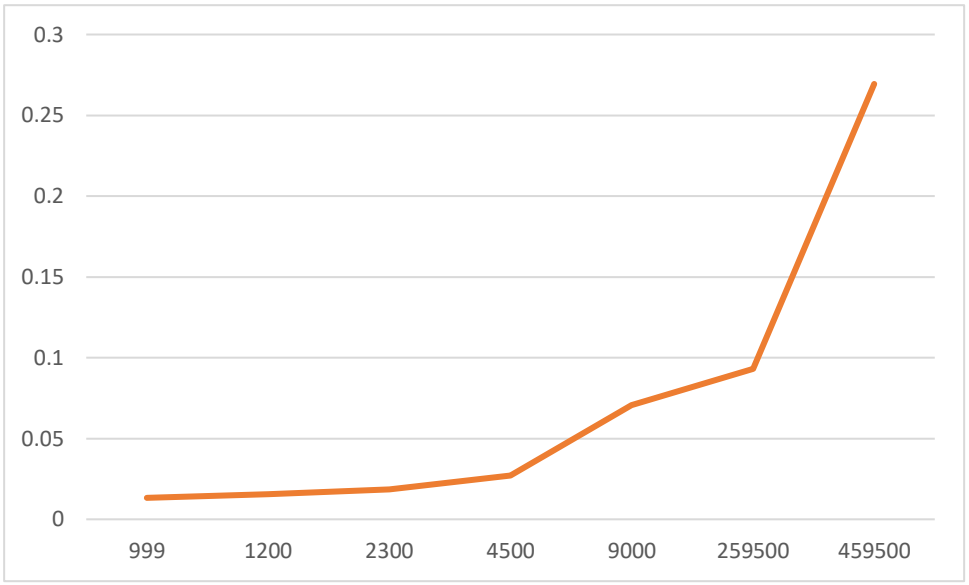
Al igual que la modificación de vértices, las aristas también producen un crecimiento **exponencial** del tiempo. Aun así, vemos que el crecimiento se produce al aumentar mucho el número de aristas (concretamente al pasar de la mitad de aristas máximas que puede tener el grafo)

$$aristas\ maximas = vertices * \frac{vertices - 1}{2}$$

Al superar la mitad de esa cifra, es cuando el crecimiento acelera ("codo" de la exponencial). Nos percatamos de ello al ver que los resultados no se apreciaban demasiado y en algunos casos daba la sensación de ser prácticamente lineales. Según se ha explicado en teoría, eso no es así y probamos a poner valores muy altos (por eso los valores probados no son equidistantes).

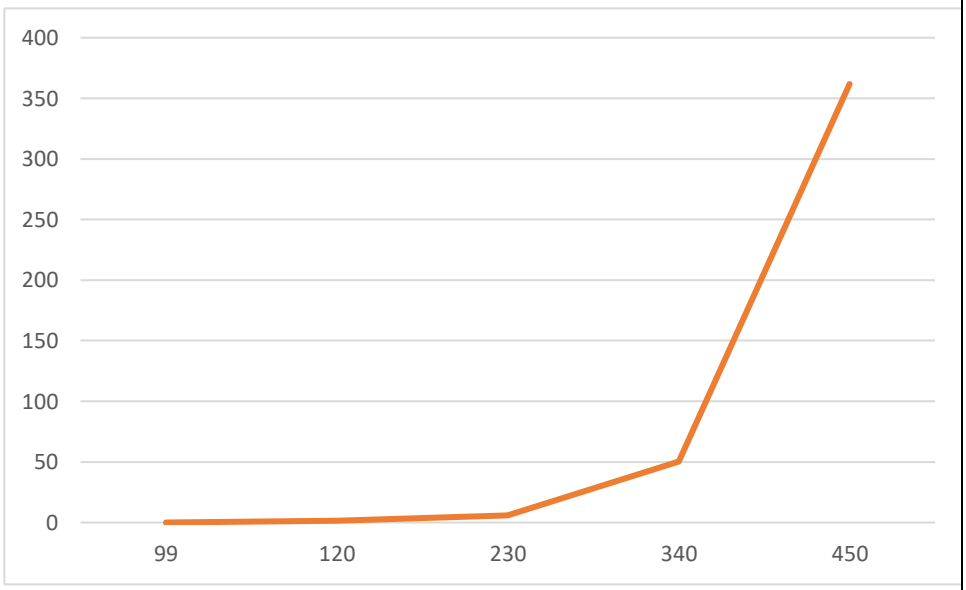
Greedy – 5 visits – 1000 vértices

999	0.0132729
1200	0.0156725
2300	0.0186423
4500	0.0272055
9000	0.07070076
259500	0.092938
459500	0.269391



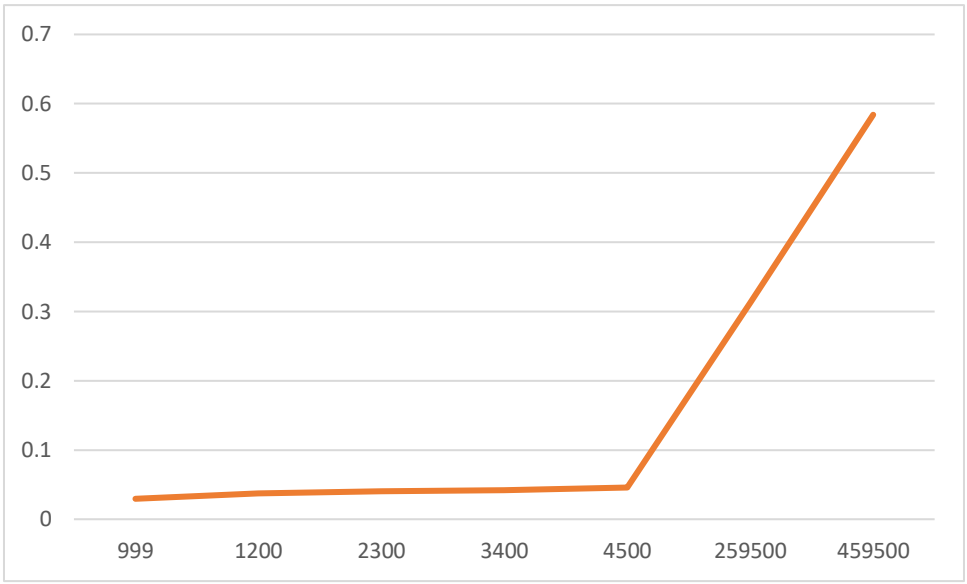
Backtracking puro – 5 visits – 100 vértices

99	0.00857675
120	1.48295
230	5.90311
340	50.4171
450	361.79
4500	



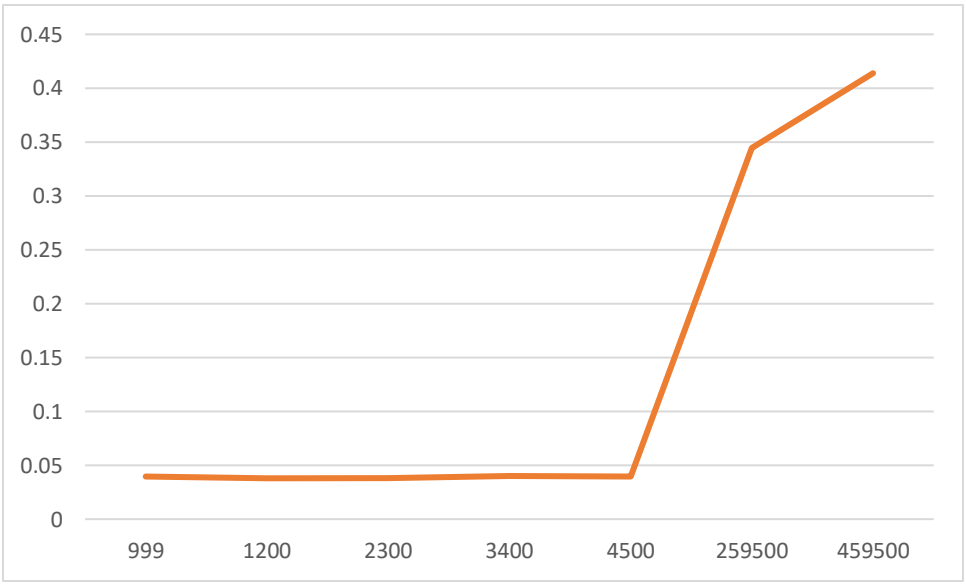
Backtracking greedy – 5 visits – 1000 vértices

999	0.0295356
1200	0.0372596
2300	0.0403796
3400	0.0422631
4500	0.0457587
259500	0.312806
459500	0.584017



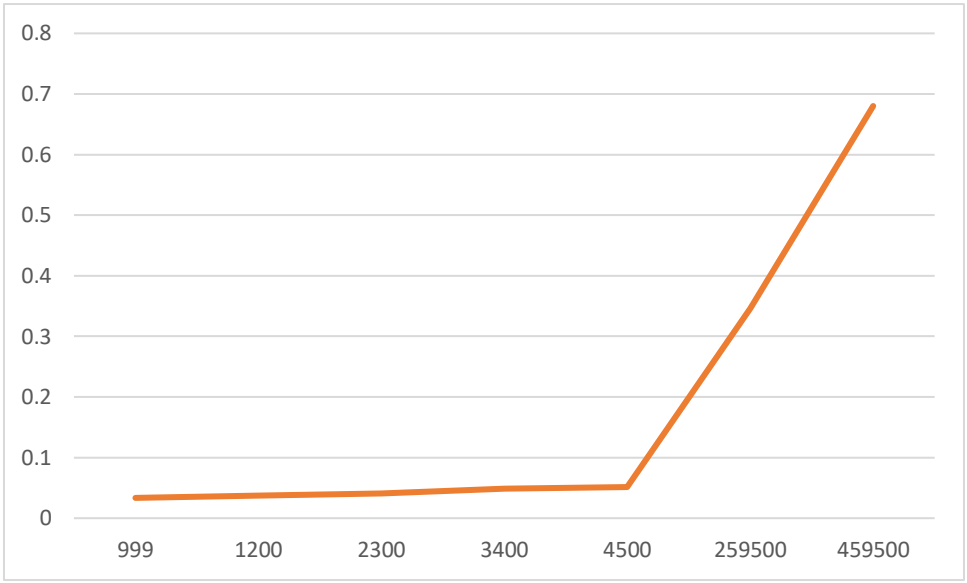
Brach and bound1

999	0.0398008
1200	0.0379224
2300	0.0379963
3400	0.040067
4500	0.0396707
259500	0.344425
459500	0.413915



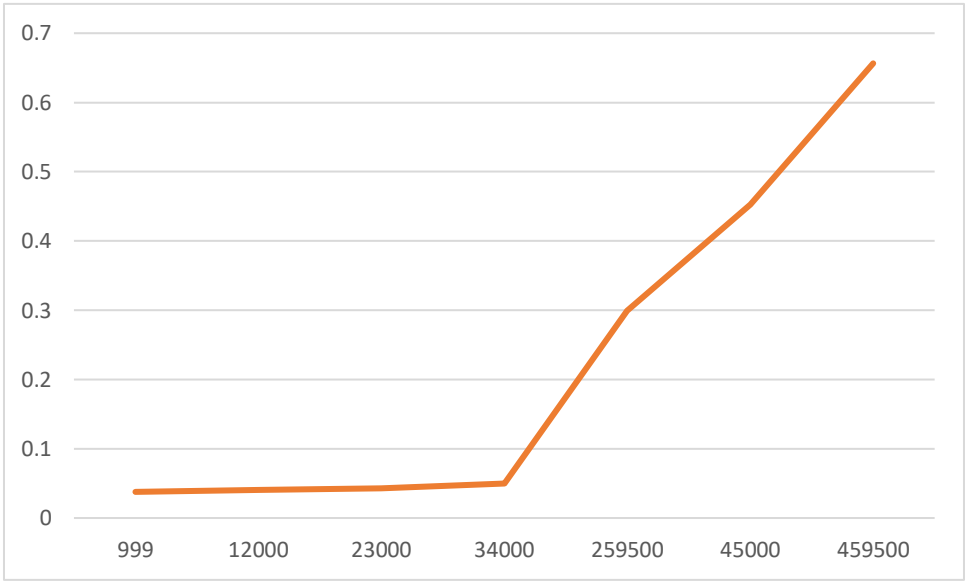
Brach and bound2

999	0.0334806
1200	0.0371998
2300	0.0409498
3400	0.0488846
4500	0.0509065
259500	0.346473
459500	0.680101

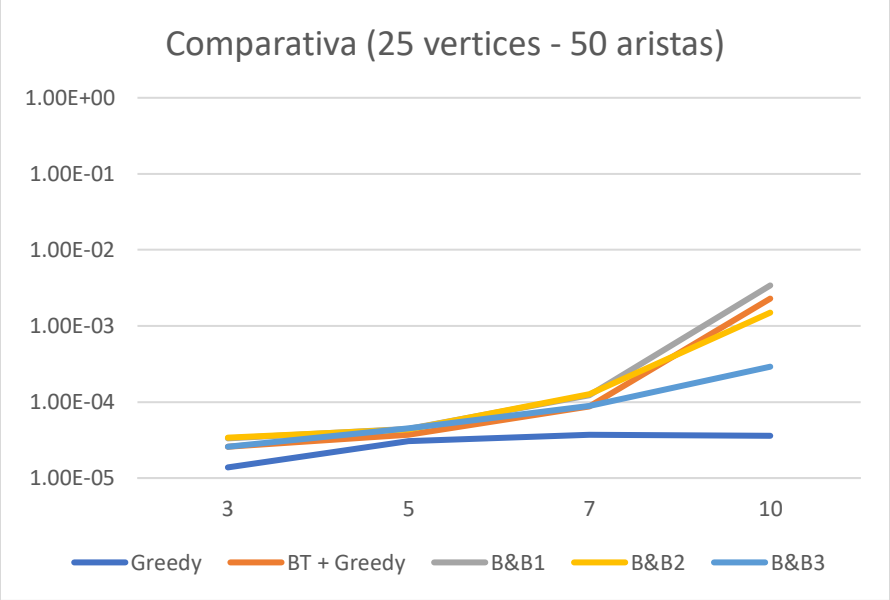


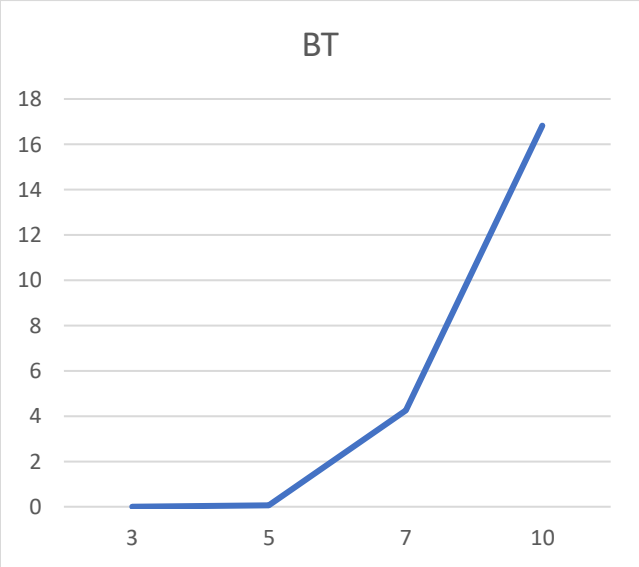
Brach and bound3 – 5 visits – 1000 vértices

999	0.0377587
12000	0.0408993
23000	0.0432056
34000	0.0497106
259500	0.299502
45000	0.452394
459500	0.656469



3. **¿Cómo varía el tiempo de ejecución de los algoritmos al variar el número de visitas? Constante, logarítmico, lineal, exponencial, factorial, etc.**
   - **Una gráfica con las curvas de los 6 algoritmos. En el caso que no se pueda ver bien alguna curva en la primera gráfica, hacer otra gráfica donde sólo se vean las curvas que no se veían bien.**

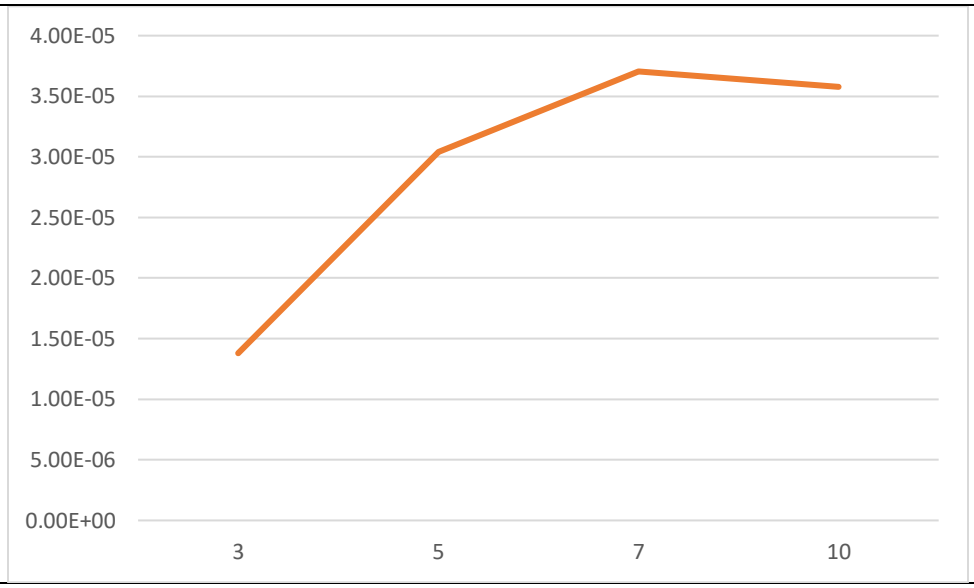




Vemos como greedy tiene (o aparenta tener) un crecimiento **logarítmico** mientras que el resto (los algoritmos de backtracking y Branch and bound) lo tienen de carácter **exponencial**.

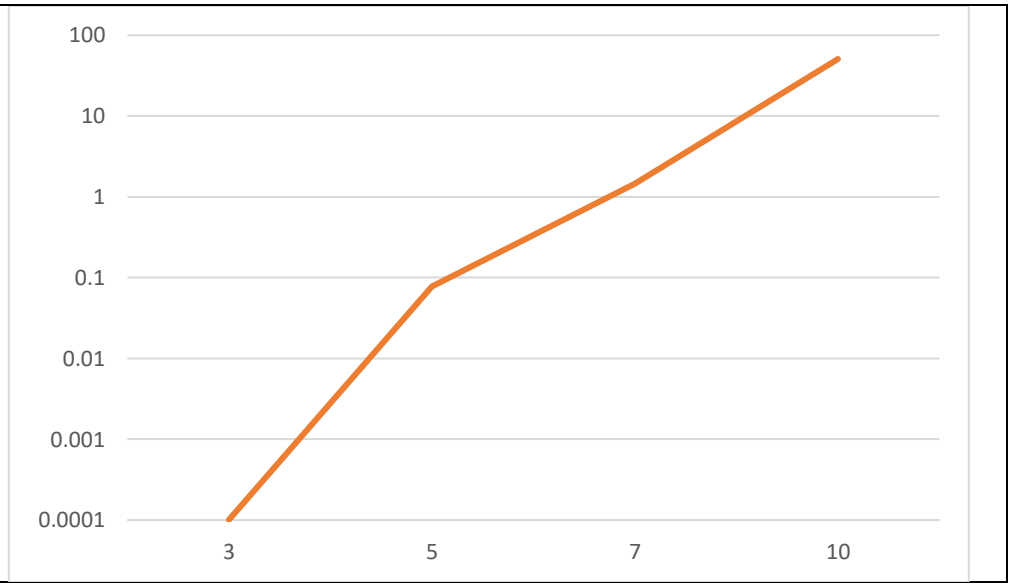
Greedy – 25 vértices – 50 aristas

3	1.37964e-05
5	3.04032e-05
7	3.70459e-05
10	3.57684e-05



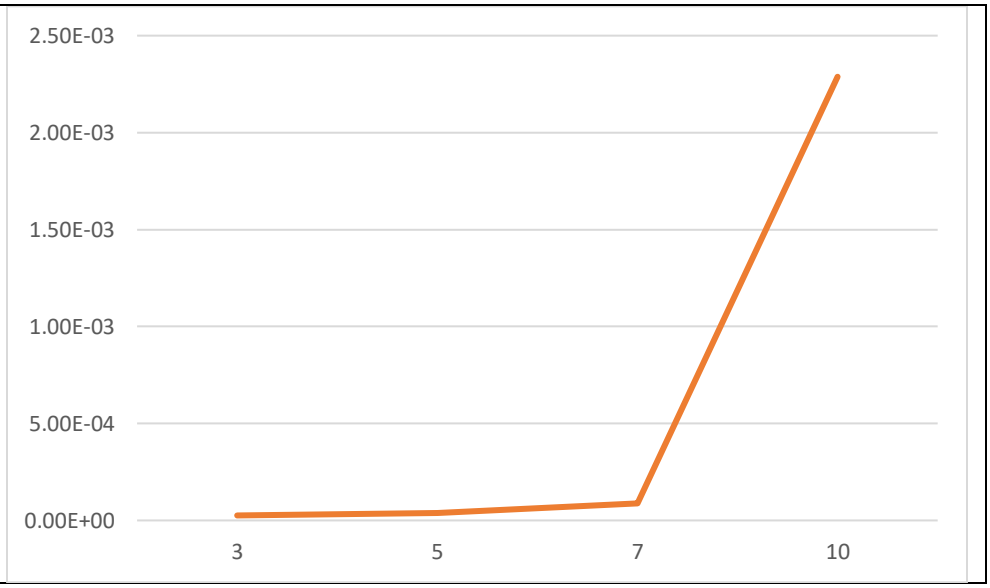
Backtracking puro – 25 vértices – 50 aristas

3	0.000100663
5	0.158073
7	1.55105
10	50.8212



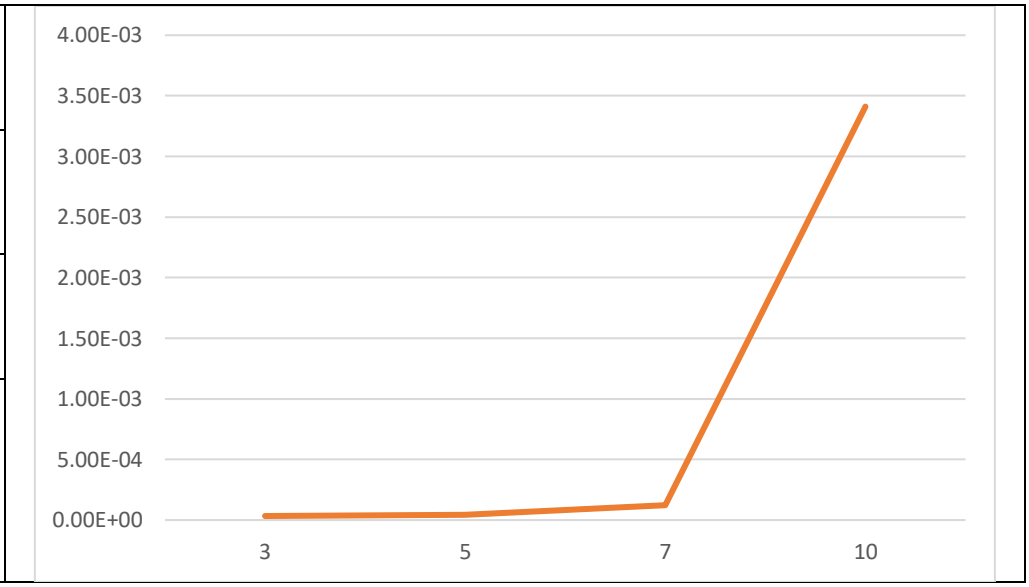
Backtracking greedy – 25 vértices – 50 aristas

3	2.58044e-05
5	3.73014e-05
7	8.76326e-05
10	0.0022879



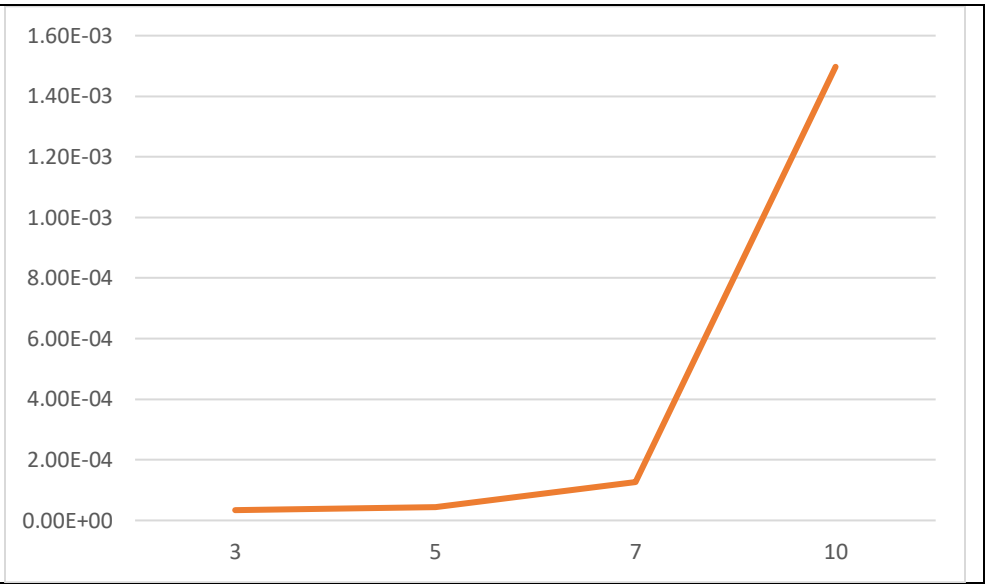
Brach and bound1 - 25 vértices – 50 aristas

3	3.32135e-05
5	4.4455e-05
7	0.000121868
10	0.00341154



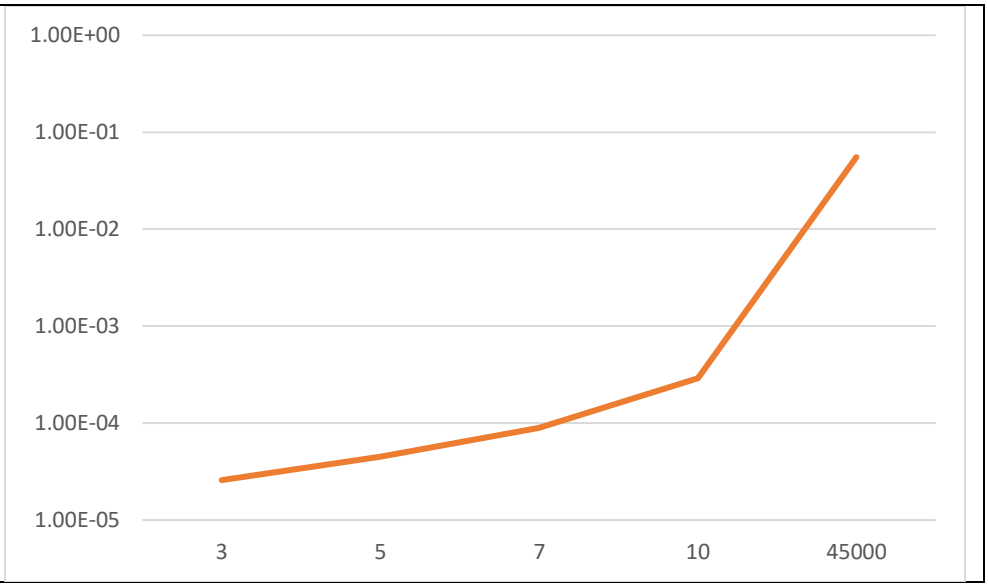
Brach and bound2 - 25 vértices – 50 aristas

3	3.398e-05
5	4.34331e-05
7	0.000125956
10	0.00149716

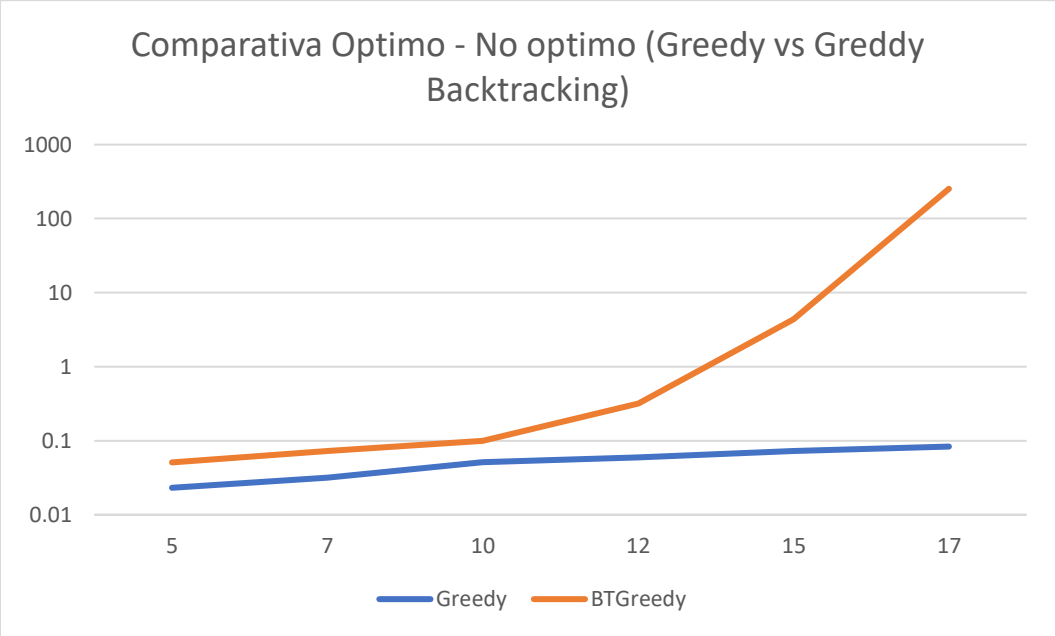


Brach and bound3 – 25 vértices – 50 aristas

3	2.58044e-05
5	4.4966e-05
7	8.89101e-05
10	0.000291002



4. **¿En cuánto empeora el resultado del algoritmo greedy respecto la solución óptima al variar número de vértices, aristas y visitas?**
   - **Una gráfica donde se han fijado el número de vértices y aristas. Sólo se varía el número de visitas.**
   - **El empeoramiento de un resultado se puede medir como el tanto por cierto de longitud de más que tiene una solución respecto la óptima.**



	Greedy	BTGreedy	"SpeedUp"
5	0,0231695	0,0510018	220,1247329%
7	0,0319739	0,0733092	229,2782551%
10	0,0509202	0,0997063	195,8089324%
12	0,0598077	0,318099	531,8696422%
15	0,07267	4,237	6009,632586%
17	0,0834945	254,8173	303628,3827%

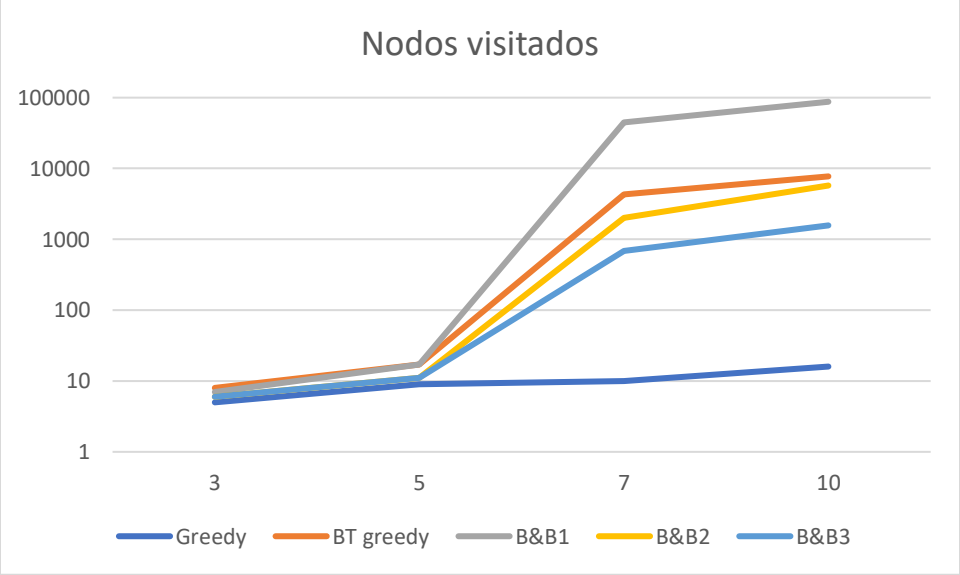
Los “SpeedUp” representan el porcentaje de mejora del greedy al backtracking greedy. A medida que aumentamos la cantidad de visitas a realizar también se nota más la diferencia de tiempo entre los dos algoritmos.

Con estos resultados podemos decir que, si solo nos interesa obtener una solución, es mucho mejor realizar los cálculos usando el algoritmo Greedy, ya que la mejora es muy grande incluso cuando se deben hacer pocas visitas

5. **¿Cuánto ha de podar un algoritmo de Branch & bound (1,2,3) para ser mejor que un algoritmo de backtracking greedy? Considerar el número de nodos del árbol de búsqueda que recorre cada algoritmo y cuando tiempo dedican a cada nodo. Ver gráficamente cuando se consigue una mejora de tiempo considerando los nodos del árbol de búsqueda visitados.**
   - **Para responder a esta pregunta fijar el número de vértices y aristas del grafo. Sólo variar el número de visitas.**
   - **Considerar gráficas con tiempo y gráficas con nodos del árbol de solución visitados. Los nodos del árbol de búsqueda serán las soluciones parciales generadas por los algoritmos de Branch&bound y las llamadas recursivas del algoritmo de backtracking**.

La siguiente tabla contiene los resultados de ejecuciones usando 25 vértices y 50 aristas. Se muestra la cantidad de nodos abiertos

	Greedy	BT greedy	B&B1	B&B2	B&B3
3	5	8	7	6	6
5	9	17	17	11	11
7	10	4323	44821	2021	683
10	16	7732	87483	5749	1573



Greedy solo abrirá nodos necesarios para llegar a la próxima visita hasta llegar al final, por lo tanto, el número que nos muestra la interfaz es el número de nodos visitados.

BT puro se ha omitido ya que siempre recorre todo y sus resultados no varían

B&B3 es el que realiza la mejor poda (ya que escogemos el camino más corto y más avanzado (tiene más nodos)).

B&B2 realiza lo mismo, pero sin tener en cuenta si está avanzado o no.

B&B1 usa el camino más largo siempre, lo que hace que pode muy poco y visite tantos nodos siempre