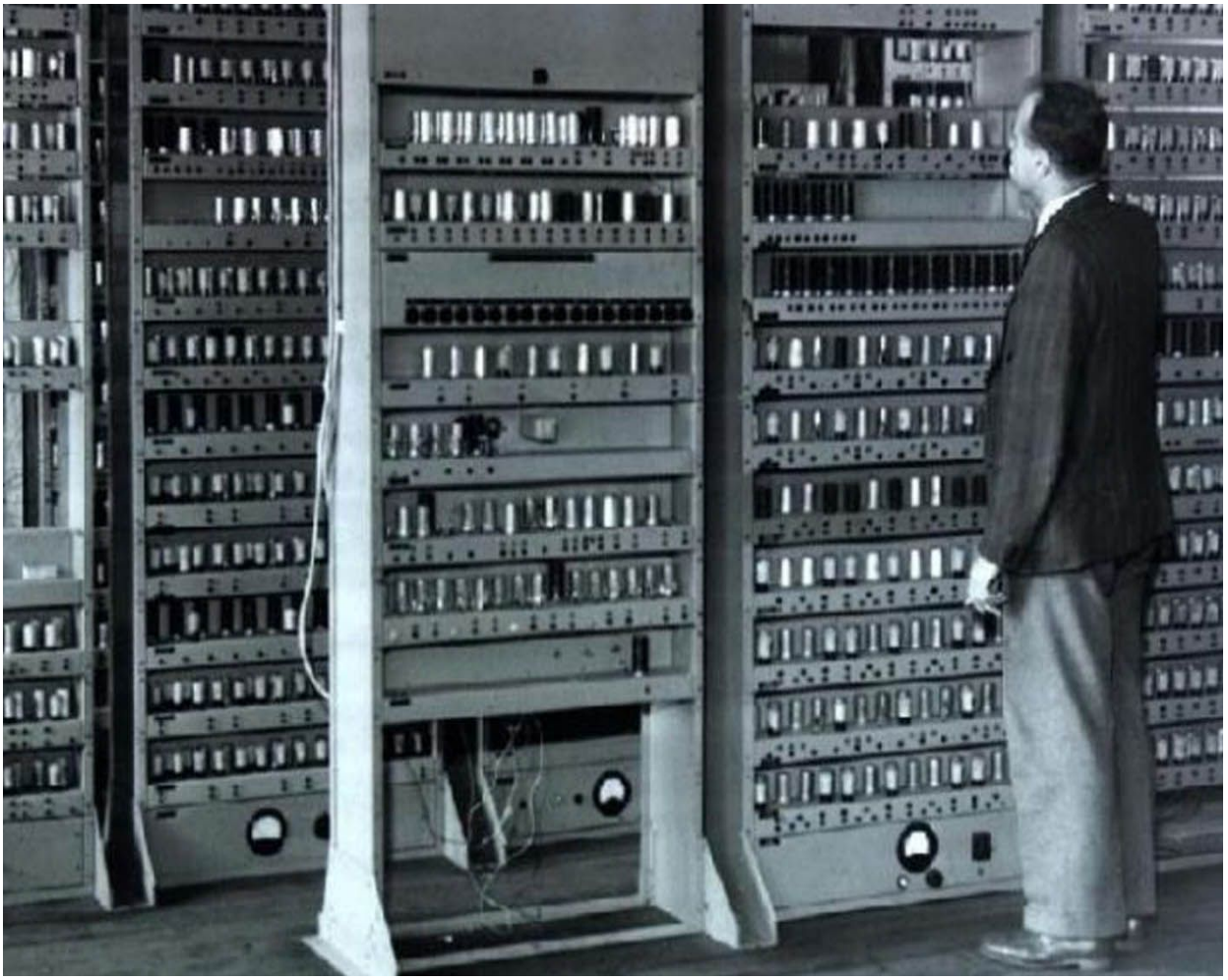
Informe Final

Algoritmos y Computabilidad



Grupo:
Doramas Báez Bernal
Kevin David Rosales Santana
Marcos Jesús Santana Perez

ÍNDICE

1. Introducción

Para entender el informe hay que tener en cuenta los siguientes elementos:

1. En primer lugar, la descripción de los problemas y algoritmos usados es **breve**. Esto es debido a que los problemas se han relatado en clase (con lo que no hace falta volver a explicarlos detalladamente) y los algoritmos son autoexplicativos en su código (algunos incluso contienen comentarios). Las secciones puntuales de estos últimos en caso de que varíen drásticamente de lo planteado en clase o sean poco entendibles a primera vista serán explicados detalladamente. Por ejemplo: cliques utilizados, restricciones complejas…
2. En segundo lugar, los tiempos se han calculado en base a un tiempo máximo de 3-5 minutos **sin cortar la ejecución tras una serie de minutos** (Como es posible hacer en MIP). **El algoritmo usado para la corrección** tendrá en cuenta un compromiso de **calidad-tiempo.** En concreto, si el algoritmo con un tamaño dado durara más de 10 minutos, se tratará de reducir la calidad para que el tiempo no sea demasiado alto.
    a. En algunos de los algoritmos de MIP se hace uso de parámetros como *timeLimit* y *MIPGap*. **Esto puede provocar que con fines temporales nos alejemos de la solución óptima.**
3. En tercer lugar, el informe contendrá los siguientes apartados:
    a. Descripción del Problema y sus Algoritmos.
    b. Algoritmo utilizado en su corrección.
    c. Análisis de diversos tiempos.
4. En cuarto lugar, se dispone de dos modos de funcionamiento:
    a. Funcionamiento Académico:
        i. Se llama al método con “python solver.py -m “Método” -p “Fichero”.
    b. Funcionamiento Profesional (Para evaluar la práctica):
        i. Se llama al método con “python solver.py “Fichero”.
5. **Todos los algoritmos han sido probados en el curso de Coursera para comprobar que funcionaban correctamente y como era de esperar.**
6. Por último, se dispone de ficheros separados en cada uno de los repositorios que **contienen los análisis temporales** que se encuentran en el presente informe también y un .md (Documento Markdown) donde **se encuentra cada uno de los métodos utilizados** y cómo llamarlos (Opción -m del **Funcionamiento Académico** mencionado previamente).

2. Knapsack

Knapsack o también conocido como **problema de la mochila(KP)**, consiste en buscar la mejor solución entre un conjunto finito de posibles soluciones a un problema. El problema es el siguiente: se parte de una mochila (con una capacidad) y existen unos items (que tienen un valor). El objetivo es maximizar el valor sin superar la capacidad de la mochila.

2.1 Optional Greedy

Se trata de una estrategia Greedy que mejora la estrategia Greedy que existe por defecto. Dicha estrategia usa una serie de coeficientes resultantes de dividir el peso entre el beneficio de coger dicho item.

2.2 Memoization

Se trata de una implementación hecha con Programación Dinámica que hace uso de un método recursivo con un algoritmo planteado en clase. **No** ofrece una explicación de la traza utilizada.

2.3 Tabulation

Al igual que la implementación realizada en Memoization, se hace uso de Programación Dinámica con un algoritmo también planteado en clase. A su vez, se realiza mediante un método la traza utilizada para conocer qué items se han cogido y cuáles no.

2.4 Branch & Bound Recursivo

Implementación básica y no recomendada de un algoritmo de ramificación y acotación en profundidad. Dicha implementación se basa en crear una serie de nodos siempre que satisfagan las restricciones de peso y no crearlos si se tiene que el valor máximo estimado no es superior a un valor ya encontrado (con lo que se procede a su correspondiente poda).

2.5 Branch & Bound Iterativo

Implementación recomendada de Branch & Bound en profundidad en la cual no se hace uso de la recursividad, haciendo que sea más estable.

2.6 Branch & Bound con Mejor Relajación Lineal

Implementación en la que se hace uso de una relajación lineal más precisa con coeficientes fraccionarios de los ítems. De esta manera, se puede proceder a las podas antes y se convierte en un algoritmo más eficaz.

2.7 Branch & Bound - Best First

Implementación que no recorre la pila en profundidad, sino que recoge el nodo que tiene mejor valor estimado en todo momento.

2.8 MIP

Implementación que hace uso de Programación Entera Mixta. El algoritmo escrito es bastante básico y fácil de entender.

2.9 Algoritmo Usado para la Corrección

El algoritmo usado para la Corrección es **MIP** debido a que, como se puede ver en los análisis temporales, llega a bastante tamaño de entrada y lo hace siempre en un tiempo óptimo.

2.10 Análisis Temporal

NOTA: La entrada **ks_ejemplo** contiene el caso de prueba de las Diapositivas de P3.

Algoritmo	Entrada	Salida	Vector Elementos	Tiempo [ms]
Básico	ks_4_0	18	1 1 0 0	1
Básico	ks_19_0	11476	1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0	0
Básico	ks_30_0	90000	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2
Básico	ks_40_0	90001	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2
Básico	ks_45_0	22132	1 1 1 1 1 1 1 1 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1
Básico	ks_50_0	140034	1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	1
Básico	ks_50_1	4919	1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	15
Básico	ks_60_0	90000	Etc.	1
Básico	ks_82_0	104675449	Etc.	3
Básico	ks_100_0	90000	Etc.	2
Básico	ks_100_1	1324496	Etc.	1
Básico	ks_100_2	9869	Etc.	1
Básico	ks_106_0	106815225	Etc.	9
Básico	ks_200_0	90001	Etc.	1
Básico	ks_200_1	1093723	Etc.	1
Básico	ks_300_0	1677592	Etc.	12
Básico	ks_400_0	3936579	Etc.	8
Básico	ks_500_0	49877	Etc.	5
Básico	ks_1000_0	100891	Etc.	13
Básico	ks_10000_0	1012574	Etc.	50
Básico	ks_lecture_dp_1	11	1 1 0	2
Básico	ks_lecture_dp_2	35	1 1 0 0	5
Básico	ks_ejemplo.txt	7	1 1 0 0	5

Algoritmo	Entrada	Salida	Vector Elementos	Tiempo [ms]
Greedy	ks_4_0	18	1 1 0 0	2
Greedy	ks_19_0	11981	0 0 1 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0	10
Greedy	ks_30_0	90000	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1
Greedy	ks_40_0	96474	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0	1
Greedy	ks_45_0	23974	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0	1
Greedy	ks_50_0	141956	1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	3
Greedy	ks_50_1	5326	1 1 1 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0	1
Greedy	ks_60_0	90000	Etc.	1
Greedy	ks_82_0	104675449	Etc.	1
Greedy	ks_100_0	90000	Etc.	4
Greedy	ks_100_1	1333635	Etc.	8
Greedy	ks_100_2	10892	Etc.	1
Greedy	ks_106_0	106815225	Etc.	11
Greedy	ks_200_0	100062	Etc.	12
Greedy	ks_200_1	1103442	Etc.	1
Greedy	ks_300_0	1688584	Etc.	11
Greedy	ks_400_0	3966813	Etc.	13
Greedy	ks_500_0	54891	Etc.	2
Greedy	ks_1000_0	109869	Etc.	4
Greedy	ks_10000_0	1099870	Etc.	66
Greedy	ks_lecture_dp_1	8	1 0 1	1
Greedy	ks_lecture_dp_2	35	1 1 0 0	1
Greedy	ks_ejemplo.txt	7	1 1 0 0	0

Algoritmo	Entrada	Salida	Vector Elementos	Tiempo [ms]
Memoization	ks_4_0	19	-	1
Memoization	ks_19_0	12248	-	62
Memoization	ks_30_0	99798	-	9
Memoization	ks_40_0	99924	-	14
Memoization	ks_45_0	23974	-	1720
Memoization	ks_50_0	142156	-	14435
Memoization	ks_50_1	5345	-	401
Memoization	ks_60_0	99837	-	39
Memoization	ks_82_0	Duración demasiado larga [Más de 5 minutos]: - Puede que sea por problemas de la pila de ejecución del algoritmo recursivo - Hay ciertos tamaños que sí admiten aún así una respuesta, aun siendo "mayores" debido a su estructura con corto o largo tiempo. Un ejemplo es "ks_100_0", cuyo resultado es 99837 con tiempo de 99 ms o "ks_100_1", cuyo resultado es 1333930 con tiempo de 400576 ms.		
Memoization	ks_100_0			
Memoization	ks_100_1			
Memoization	ks_100_2			
Memoization	ks_106_0			
Memoization	ks_200_0			
Memoization	ks_200_1			
Memoization	ks_300_0			
Memoization	ks_400_0			
Memoization	ks_500_0			
Memoization	ks_1000_0			
Memoization	ks_10000_0			

Memoization	ks_lecture_dp_1	11	-	1
Memoization	ks_lecture_dp_2	44	-	0
Memoization	ks_ejemplo.txt	7	-	0

Algoritmo	Entrada	Salida	Vector Elementos	Tiempo [ms]
Tabulation	ks_4_0	19	[0, 0, 1, 1]	1
Tabulation	ks_19_0	12248	[0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0]	350
Tabulation	ks_30_0	99798	[0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]	1403
Tabulation	ks_40_0	99924	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]	1877
Tabulation	ks_45_0	23974	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	1501
Tabulation	ks_50_0	142156	[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]	11162
Tabulation	ks_50_1	5345	[1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	157
Tabulation	ks_60_0	99837	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	2731
Tabulation	ks_82_0	Duración demasiado larga		
Tabulation	ks_100_0	99837	Etc.	4265
Tabulation	ks_100_1	Duración demasiado larga [Más de 5 minutos]: - Puede que sea por problemas de la pila de ejecución. - Hay ciertos tamaños que si admiten aun así una respuesta, aun siendo "mayores" debido a su estructura con corto o largo tiempo.		
Tabulation	ks_100_2			
Tabulation	ks_106_0			
Tabulation	ks_200_0			
Tabulation	ks_200_1			
Tabulation	ks_300_0			
Tabulation	ks_400_0			
Tabulation	ks_500_0			
Tabulation	ks_1000_0			
Tabulation	ks_10000_0			
Tabulation	ks_lecture_dp_1	11	[1, 1, 0]	1
Tabulation	ks_lecture_dp_2	44	[1, 0, 0, 1]	1
Tabulation	ks_ejemplo.txt	7	[1, 1, 0, 0]	0

Algoritmo	Entrada	Salida	Vector Elementos	Tiempo [ms]
B&B Rec.	ks_4_0	19	[1, 0, 0, 1]	1
B&B Rec.	ks_19_0	12248	[0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0]	18589
B&B Rec.	ks_30_0	99798	[0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]	3825
B&B Rec.	ks_40_0	Duración demasiado larga [Más de 5 minutos]: - Puede que sea por problemas de la pila de ejecución del algoritmo recursivo.		
B&B Rec.	ks_45_0			
B&B Rec.	ks_50_0			
B&B Rec.	ks_50_1			
B&B Rec.	ks_60_0	- Haay ciertos tamaños que sí admiten aun así una respuesta, laun siendo "mayores" debido a su estructura con corto o go tiempo.		
B&B Rec.	ks_82_0			
B&B Rec.	ks_100_0			
B&B Rec.	ks_100_1			
B&B Rec.	ks_100_2			
B&B Rec.	ks_106_0			
B&B Rec.	ks_200_0			
B&B Rec.	ks_200_1			
B&B Rec.	ks_300_0			
B&B Rec.	ks_400_0			
B&B Rec.	ks_500_0			
B&B Rec.	ks_1000_0			
B&B Rec.	ks_10000_0			
B&B Rec.	ks_lecture_dp_1	11	[1, 1, 0]	1
B&B Rec.	ks_lecture_dp_2	44	[1, 0, 0, 1]	7
B&B Rec.	ks_ejemplo.txt	7	[1, 1, 0, 0]	2

Algoritmo	Entrada	Salida	Vector Elementos	Tiempo [ms]
B&B Iter.	ks_4_0	19	[0, 0, 1, 1]	3
B&B Iter.	ks_19_0	12248	[0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0]	230
B&B Iter.	ks_30_0	99798	[0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]	125
B&B Iter.	ks_40_0	99924	[0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0]	2491
B&B Iter.	ks_45_0	Duración demasiado larga [Más de 5 minutos]: - Puede que sea por problemas de la pila de ejecución. - Hay ciertos tamaños que sí admiten aun así una respuesta, aun siendo "mayores" debido a su estructura con corto o largo tiempo.		
B&B Iter.	ks_50_0			
B&B Iter.	ks_50_1			
B&B Iter.	ks_60_0			
B&B Iter.	ks_82_0			
B&B Iter.	ks_100_0			
B&B Iter.	ks_100_1			
B&B Iter.	ks_100_2			
B&B Iter.	ks_106_0			
B&B Iter.	ks_200_0			
B&B Iter.	ks_200_1			
B&B Iter.	ks_300_0			
B&B Iter.	ks_400_0			
B&B Iter.	ks_500_0			
B&B Iter.	ks_1000_0			
B&B Iter.	ks_10000_0			
B&B Iter.	ks_lecture_dp_1	11	[1, 1, 0]	2
B&B Iter.	ks_lecture_dp_2	44	[1, 0, 0, 1]	2
B&B Iter.	ks_ejemplo.txt	7	[1, 1, 0, 0]	3

Algoritmo	Entrada	Salida	Vector Elementos	Tiempo [ms]
B&B Iter. Coef. Frac.	ks_4_0	19	[0, 0, 1, 1]	7
B&B Iter. Coef. Frac.	ks_19_0	12248	[0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0]	1570

B&B Iter. Coef. Frac.	ks_30_0	99798	[0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]	1329
B&B Iter. Coef. Frac.	ks_40_0	99924	[0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0]	26212
B&B Iter. Coef. Frac.	ks_45_0	Duración demasiado larga [Más de 5 minutos]: - Puede que sea por problemas de la pila de ejecución. - Hay ciertos tamaños que sí admiten aun así una respuesta, aun siendo "mayores" debido a su estructura con corto o largo tiempo.		
B&B Iter. Coef. Frac.	ks_50_0			
B&B Iter. Coef. Frac.	ks_50_1			
B&B Iter. Coef. Frac.	ks_60_0			
B&B Iter. Coef. Frac.	ks_82_0			
B&B Iter. Coef. Frac.	ks_100_0			
B&B Iter. Coef. Frac.	ks_100_1			
B&B Iter. Coef. Frac.	ks_100_2			
B&B Iter. Coef. Frac.	ks_106_0			
B&B Iter. Coef. Frac.	ks_200_0			
B&B Iter. Coef. Frac.	ks_200_1			
B&B Iter. Coef. Frac.	ks_300_0			
B&B Iter. Coef. Frac.	ks_400_0			
B&B Iter. Coef. Frac.	ks_500_0			
B&B Iter. Coef. Frac.	ks_1000_0			
B&B Iter. Coef. Frac.	ks_10000_0			
B&B Iter. Coef. Frac.	ks_lecture_dp_1	11	[1, 1, 0]	1
B&B Iter. Coef. Frac.	ks_lecture_dp_2	44	[1, 0, 0, 1]	1
B&B Iter. Coef. Frac.	ks_ejemplo.txt	7	[1, 1, 0, 0]	1

Algoritmo	Entrada	Salida	Vector Elementos	Tiempo [ms]
B&B Mejor primero.	ks_4_0	19	[0, 0, 1, 1]	0
B&B Mejor primero.	ks_19_0	Duración demasiado larga [Más de 5 minutos]		
B&B Mejor primero.	ks_30_0	99798	[0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]	81758
B&B Mejor primero.	ks_40_0	Duración demasiado larga [Más de 5 minutos]		
B&B Mejor primero.	ks_45_0			
B&B Mejor primero.	ks_50_0			
B&B Mejor primero.	ks_50_1			
B&B Mejor primero.	ks_60_0			
B&B Mejor primero.	ks_82_0			
B&B Mejor primero.	ks_100_0			
B&B Mejor primero.	ks_100_1			
B&B Mejor primero.	ks_100_2			

B&B Mejor primero.	ks_106_0			
B&B Mejor primero.	ks_200_0			
B&B Mejor primero.	ks_200_1			
B&B Mejor primero.	ks_300_0			
B&B Mejor primero.	ks_400_0			
B&B Mejor primero.	ks_500_0			
B&B Mejor primero.	ks_1000_0			
B&B Mejor primero.	ks_10000_0			
B&B Mejor primero.	ks_lecture_dp_1	11	[1, 1, 0]	0
B&B Mejor primero.	ks_lecture_dp_2	44	[1, 0, 0, 1]	0

Algoritmo	Entrada	Salida	Vector	Tiempo
MIP	ks_4_0	19	0 0 1 1	10ms
	ks_19_0	12248	0 0 1 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0	37ms
	ks_30_0	99798	0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0	25ms
	ks_40_0	99924	0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 0 0 1 0 0 0	153ms
	ks_45_0	23974	[Etc]	13ms
	ks_50_0	142156		33ms
	ks_50_1	5345		20ms
	ks_60_0	99837		25ms
	ks_82_0	104716758		19ms

	ks_100_0	99837		31ms
	ks_100_1	1333930		135ms
	ks_100_2	10892		20ms
	ks_106_0	106919284		26ms
	ks_200_0	100236		53ms
	ks_200_1	1103604		432ms
	ks_300_0	1688692		2242ms
	ks_400_0	3967080		106ms
	ks_500_0	54939		45ms
	ks_1000_0	109899		39ms
	ks_10000_0	1099807		193ms
	ks_lecture_dp_1	11		13ms
	ks_lecture_dp_2	44		20

3. TSP

El **problema del vendedor viajero** (TSP), tiene como objetivo dada una lista de ciudades y las distancias entre cada par de ellas, obtener cual es la ruta más corta posible para visitar cada ciudad una sola vez y volviendo por último a la ciudad origen.

3.1 2-Approx

Para la reproducción de este algoritmo, nos basamos en el procedimiento definido en las diapositivas de la asignatura:

- El coste de las aristas debe cumplir la desigualdad triangular. Dados 3 vértices se debe cumplir:

$$c_{ij} \leq c_{ik} + c_{kj}$$

- Encontrar el Árbol de Expansión Mínima, T (Kruskal, Prim, Boruvka)
- Crear un multigrafo G* duplicando todas las aristas del árbol T
- Encontrar una cadena euleriana de G* y un circuito hamiltoniano, H*, embebido en ella

3.2 Christofides

Para la reproducción de este algoritmo, nos basamos en el procedimiento definido en las diapositivas de la asignatura:

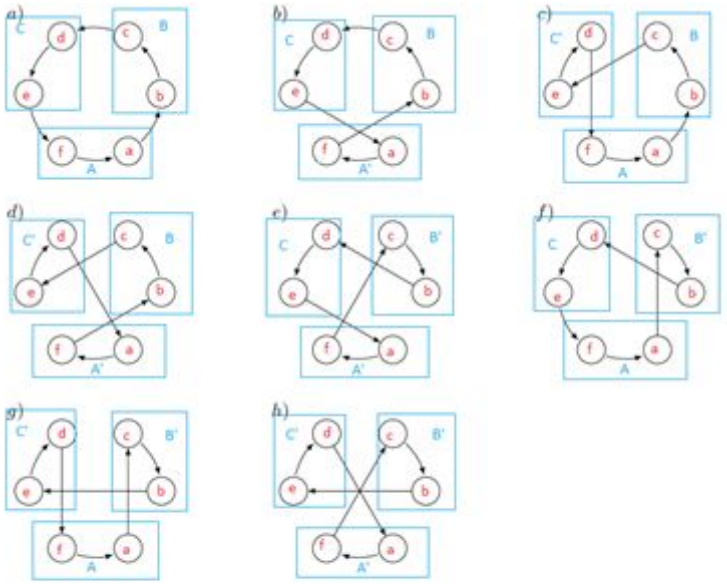
1. Obtener el Árbol de Expansión Mínima, $T^{(T \rightarrow Multigrafo)}$
2. Separar los vértices de T de grado impar, W
3. Obtener M, el emparejamiento perfecto de coste mínimo de W
4. Añadir las aristas de M a T
5. Obtener C, el camino euleriano de $M \cup T$
6. Eliminar las ocurrencias de vértices repetidos en C para obtener el camino hamiltoniano

3.3 OPT-2

Para la programación del opt2, lo primero que hacemos es tomar 4 puntos a,b,c,d (siendo ‘a’ el valor que precede a ‘b’ y ‘c’ el valor que precede a ‘d’) y calculamos el coste de la cadena si se invierte res([b:d]), es decir B’(d-b). Si esto mejora el coste de la cadena se aplica la inversión.

3.4 OPT-3

Para la programación del opt3, lo primero que hacíamos era tomar 6 puntos a,b,c,d,e,f, de la cadena que se le pasa por parámetro al algoritmo y a nivel conceptual, englobamos cada par de puntos, de la siguiente forma A(f--a) B(b--c) C(d--e), para poder calcular el coste de la cadena tras aplicarle todas las permutaciones posibles que no fueran redundantes. Un ejemplo de permutación sería A’ (a--f) B’ (c--b) C(d--e). En el siguiente esquema se verá más claro:



a) ABC, b) $A'BC$, c) ABC', d) $A'BC'$, e) $A'B'C$, f) $AB'C$, g) $AB'C'$,
h) $A'B'C'$ (Note: Symbol ' defines the reversed segment).

Una vez calculado el coste de todas las permutaciones, se procede a buscar cuál de las permutaciones dio mejor resultado, siendo esta la que finalmente aplicamos. Es importante destacar que una vez aplicada una permutación, se normalice la cadena (normalizar = desplazar la cadena hasta que su posición inicial contenga el valor 0), para que esta sea fiel al cálculo inicial que hicimos sobre cómo se quedaría la cadena si le aplicamos una permutación.

3.5 Simulated Annealing

Para la reproducción de este algoritmo, nos basamos en el siguiente esquema:

Sea $x \in S$ la configuración inicial.
Sea $T > 0$ la temperatura inicial.
Sea $N(T)$ el número máximo de iteraciones.
Repetir
{
    Repetir
    {
    Genera solución $y \in V(x) \subset S$
    Evalúa $\delta = C(y) - C(x)$
    o $\begin{cases} \delta < 0 \Rightarrow x = y \\ \delta \geq 0 \text{ y } u < exp(-\delta/T) \Rightarrow x = y \end{cases}$
    $n = n + 1$
    } mientras $n \leq N(T)$.
Disminuir T
} mientras no se haya alcanzado el criterio de parada.

3.6 Simulated Annealing + Tabu Search

El desarrollo de este algoritmo es idéntico al de Simulated Annealing con la particularidad, de que implementamos una lista tabú, que impide volver a tomar soluciones que se han tomado durante un determinado número de veces.

3.7 Genetic Algorithm (Cruce de ciclo)

Para la reproducción de este algoritmo, nos basamos en el procedimiento definido en las diapositivas de la asignatura:

$(h, k, c, e, f, d, b, l, a, i, g, j)$ Padre
$(a, b, c, d, e, f, g, h, i, j, k, l)$ Madre

Cruce de ciclo

Genera un hijo de forma que cualquiera de sus caracteres mantiene la posición del padre o de la madre, de acuerdo con sus posiciones en un ciclo.

En el ejemplo anterior, para la primera posición hay dos opciones para el hijo: *h* como el padre o *a* como la madre; se elige la h. Por lo tanto tendríamos (h,–,–,–,–,–,–,–,–,–,–,–)

Al hacer esta elección, *a* debe ir en la posición del padre obteniendo (h,–,–,–,–,–,–,–, a,–,–,–).

Esto obliga a seleccionar la ciudad *i* del vector padre, ciudad bajo la a en el vector madre, obteniendo (h,–,–,–,–,–,–,–, a, i,–,–)

$(h, k, c, e, f, d, b, l, a, i, g, j)$ Padre
$(a, b, c, d, e, f, g, h, i, j, k, l)$ Madre

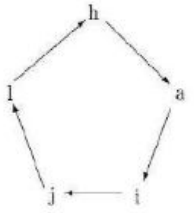
Cruce de ciclo

(h,–,–,–,–,–,–,–, a, i,–,–)

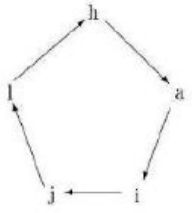
Por la misma razón, los caracteres *j* y *l* deben mantener la posición del padre, llegando a (h,–,–,–,–,–,–, l, a, i,–, j).

Se ha definido así un ciclo al elegir *h* en la primera posición:
h → a → i → j → l que denotaremos por ciclo 1.

(Si se hubiera elegido el carácter *a* para la primera posición, el ciclo sería el mismo con el orden inverso)



$(h, k, c, e, f, d, b, l, a, i, g, j)$ Padre
$(a, b, c, d, e, f, g, h, i, j, k, l)$ Madre



Cruce de ciclo

(h,–,–,–,–,–,–,–, l, a, i,–, j)

Se definen así varios ciclos, en el ejemplo son 3 más el ciclo unitario formado en la tercera posición al coincidir el carácter *c* en el padre y la madre

Denominando los ciclos 1, 2, 3 y U para el unitario, las posiciones de las 12 ciudades se asignan a estos ciclos

(1, 2,U, 3, 3, 3, 2, 1, 1, 1, 2, 1)

$(h, k, c, e, f, d, b, l, a, i, g, j)$ Padre
$(a, b, c, d, e, f, g, h, i, j, k, l)$ Madre

Cruce de ciclo (1, 2,U, 3, 3, 3, 2, 1, 1, 1, 2, 1)

La selección aleatoria fija uno de los ciclos con los caracteres del padre y el resto para la madre

En el ejemplo, se elige el ciclo 1 con los caracteres del padre y se completa el resto con lo de la madre quedando:

(h, b, c, d, e, f, g, l, a, i, k, j)

3.8 Ant Algorithm

[Clique para ver explicación del Algoritmo]

3.9 Algoritmo Usado para la Corrección

Para evitar el uso de grafos muy pesados, el algoritmo usado para la corrección depende del tamaño de la entrada:

- Si es <= a 400, se aplica **opt3** a **christofides**.
- Si es >400 y <2105, se aplica **simulated_annealing** sobre **opt2** sobre **christofides**.
- Si es >=2105 y <5000, se aplica **simulated_annealing** sobre **opt2** sobre **2approx**.
- Si es >=5000 y <25000, se aplica **simulated_annealing** sobre **opt2** sobre **greedy.**
- En el resto de casos (demasiado grandes), se aplica **simulated_annealing** sobre **greedy.**

3.10 Análisis temporal

Algoritmo	Entrada	Salida	Vector Elementos	Tiempo [Ms]
2-Aproximation	tsp_5_1	4.00	0 1 2 3 4 0	1ms
	tsp_51_1	627.08	0 33 5 2 28 45 9 10 22 1 25 20 37 21 29 42 11 40 18 16 44 15 38 14 19 7 43 26 6 36 12 30 34 24 41 27 8 4 46 3 23 35 13 47 31 39 50 48 32 17 49 0	10ms
	tsp_70_1	874.58	0 35 57 2 23 56 27 30 21 49 58 33 65 47 63 34 69 50 11 1 22 14 44 16 24 46 55 68 51 7 5 40 6 52 38 36 41 66 37 42 62 48 54 64 60 13 19 53 32 8 43 4 25 9 3 59 61 10 67 18 26 12 31 20 29 28 15 45 39 17 0	36ms
	tsp_76_1	694.22	0 46 69 5 40 11 55 62 16 33 19 47 75 26 48 14 51 20 72 44 45 7 71 13 64 6 66 9 8 58 73 32 21 31 57 74 30 67 35 65 23 34 70 24 53 10 61 60 17 38 4 2 3 25 59 22 39 42 52 18 50 15 28 37 36 29 43 12 54 1 49 41 63 56 27 68 0	60ms
	tsp_76_2	139777.08	Vector demasiado largo para ser representado en la tabla	33ms
	tsp_99_1	1699.57		29ms
	tsp_100_1	839.56		44ms
	tsp_100_2	27110.16		50ms
	tsp_100_3	27640.70		80ms
	tsp_100_4	28219.73		56ms
	tsp_100_5	30094.67		56ms
	tsp_100_6	10855.29		55ms
	tsp_101_1	839.56		63ms
	tsp_105_1	19394.96		56ms
	tsp_107_1	59051.97		36ms
	tsp_124_1	79026.33		84ms
	tsp_127_1	157244.31		82ms
	tsp_136_1	144423.17		100ms
	tsp_144_1	75415.05		172ms
	tsp_150_1	35558.62		134ms
	tsp_150_2	36180.43		178ms
	tsp_152_1	84689.07		183ms
	tsp_159_1	55118.91		190ms
	tsp_195_1	3383.40		233ms
	tsp_198_1	19288.25		335ms
	tsp_200_1	40810.81		343ms
	tsp_200_2	40773.31		301ms
	tsp_225_1	178426.83		218ms
	tsp_226_1	116134.58		218ms
	tsp_262_1	3287.63		480ms
	tsp_264_1	65105.15		511ms
	tsp_299_1	64312.86		568ms

	tsp_318_1	58291.86		640ms
	tsp_318_2	58291		764ms
	tsp_400_1	20222.33		867ms

Algoritmo	Entrada	Salida	Vector Elementos	Tiempo [Ms]
opt_2 (2-Aprox)	tsp_5_1	4.00	0 1 2 3 4 0	0ms
	tsp_51_1	532.65	0 10 9 45 28 2 5 47 27 3 46 8 13 35 4 34 24 41 23 30 12 36 6 26 33 22 1 20 37 7 19 40 18 16 14 44 15 38 29 42 11 21 43 50 39 25 31 48 32 17 49 0	8ms
	tsp_70_1	782.69	0 69 34 63 47 65 33 49 58 21 27 30 56 23 2 57 35 50 11 1 22 14 45 28 29 15 20 31 12 18 67 26 3 59 9 10 61 4 25 43 32 8 48 64 54 62 19 13 60 53 42 37 66 41 36 38 52 6 40 5 7 51 55 68 46 24 16 44 39 17 0	14ms
	tsp_76_1	630.47	Vector demasiado largo para ser representado en la tabla	64ms
	tsp_76_2	132895.42		24ms
	tsp_99_1	1471.29		31ms
	tsp_100_1	685.56		63ms
	tsp_100_2	25011.65		50ms
	tsp_100_3	24364.12		46ms
	tsp_100_4	24281.44		51ms
	tsp_100_5	26076.75		36ms
	tsp_100_6	9663.71		44ms
	tsp_101_1	685.56		98ms
	tsp_105_1	16759.20		45ms
	tsp_107_1	52123.23		31ms

	tsp_124_1	66699.24		73ms
	tsp_127_1	141316.25		95ms
	tsp_136_1	118397.37		194ms
	tsp_144_1	70326.34		80ms
	tsp_150_1	32360.55		123ms
	tsp_150_2	32036.10		91ms
	tsp_152_1	79709.32		171ms
	tsp_159_1	46964.70		151ms
	tsp_195_1	2728.23		202ms
	tsp_198_1	17684.99		164ms
	tsp_200_1	34757.32		213ms
	tsp_200_2	35258.10		411ms
	tsp_225_1	150136.07		146ms
	tsp_226_1	95557.19		162ms
	tsp_262_1	2836.42		325ms
	tsp_264_1	56177.41		704ms
	tsp_299_1	57888.64		471ms
	tsp_318_1	50398.26		504ms
	tsp_318_2	50398.26		597ms
	tsp_400_1	17550.49		718ms

Algoritmo	Entrada	Salida	Vector Elementos	Tiempo [Ms]
opt_3 (2-Aprox)	tsp_5_1	4.00	0 1 2 3 4 0	0ms
	tsp_51_1	448.35	0 48 32 17 49 39 50 43 21 37 20 25 36 6 26 1 31 22 33 47 27 41 24 34 23 12 30 11 42 29 38 15 14 44 16 18 40 19 7 13 35 4 8 46 3 45 9 10 28 2 5 0	957ms
	tsp_70_1	712.52	0 1 22 14 15 20 29 28 45 12 31 67 18 26 3 59 9 25 4 10 61 43 32 8 64 54 48 37 66 42 62 13 19 60 53 41 36 38 52 6 40 5 7 51 55 68 46 24 16 44 39 17 11 50 35 57 2 23 56 30 27 21 58 49 69 34 33 65 47 63 0	2188ms
	tsp_76_1	585.00	Vector demasiado largo para ser representado en la tabla	3668ms
	tsp_76_2	113624.10		3068ms
	tsp_99_1	1293.09		6131ms
	tsp_100_1	668.05		7493ms
	tsp_100_2	24022.67		6580ms
	tsp_100_3	22478.31		8688ms
	tsp_100_4	22931.64		7285ms
	tsp_100_5	24378.64		7514ms
	tsp_100_6	8493.56		8198ms
	tsp_101_1	668.05		10169ms
	tsp_105_1	15397.25		8703ms
	tsp_107_1	46973.15		7082ms
	tsp_124_1	60896.73		14569ms
	tsp_127_1	130829.10		15278ms
	tsp_136_1	103266.35		25264ms
	tsp_144_1	62812.08		25261ms
	tsp_150_1	28938.36		29564ms
	tsp_150_2	27744.74		28095ms
	tsp_152_1	74778.75		31751ms
	tsp_159_1	44287.30		40687ms
	tsp_195_1	2448.53		63667ms
	tsp_198_1	16639.78		63465ms
	tsp_200_1	32847.49		65437ms
	tsp_200_2	31287.66		46417ms
	tsp_225_1	135809.77		74035ms
	tsp_226_1	87291.77		85352ms
	tsp_262_1	2601.06		163168ms
	tsp_264_1	54810.00		280208ms
	tsp_299_1	52191.67		248971ms
	tsp_318_1	46113.27		303813ms
	tsp_318_2	46113.27		262879ms
	tsp_400_1	16612.33		446299ms

Algoritmo	Entrada	Salida	Vector Elementos	Tiempo [Ms]
Christofides	tsp_5_1	4.00	0 1 2 3 4 0	0ms
	tsp_51_1	475.41	0 33 22 31 39 50 38 15 44 14 16 18 40 19 7 11 42 29 43 21 37 20 25 1 48 49 17 32 26 47 6 36 12 30 34 23 35 13 4 8 24 41 27 46 3 10 9 45 28 2 5 0	51ms
	tsp_70_1	775.68	0 35 50 11 1 22 14 44 16 24 46 67 18 12 31 45 28 29 20 15 26 25 4 61 10 9 59 3 43 51 7 32 8 64 54 48 62 42 60 13 19 37 66 41 53 36 38 52 6 40 5 55 68	146ms

			39 17 30 27 21 49 58 56 23 2 33 69 34 63 47 65 57 0	
	tsp_76_1	601.59	Vector demasiado largo para ser representado en la tabla	201ms
	tsp_76_2	117154.28		82ms
	tsp_99_1	1327.17		203ms
	tsp_100_1	701.42		351ms
	tsp_100_2	23649.26		213ms
	tsp_100_3	23164.69		351ms
	tsp_100_4	23546.66		201ms
	tsp_100_5	23532.58		362ms
	tsp_100_6	9138.68		472ms
	tsp_101_1	701.42		548ms
	tsp_105_1	16585.16		249ms
	tsp_107_1	48143.87		158ms
	tsp_124_1	63059.14		323ms
	tsp_127_1	130391.68 1		578ms
	tsp_136_1	105832.48		268ms
	tsp_144_1	69076.15		273ms
	tsp_150_1	28840.82		1033ms
	tsp_150_2	29905.29		913ms
	tsp_152_1	76212.99		193ms
	tsp_159_1	45699.29		668ms
	tsp_195_1	2543.65		899ms
	tsp_198_1	17294.49		1101ms
	tsp_200_1	32184.15		1963ms
	tsp_200_2	32524.23		957ms
	tsp_225_1	133556.35		235ms
	tsp_226_1	91203.57		894ms
	tsp_262_1	2656.08		2417ms
	tsp_264_1	52492.73		1575ms
	tsp_299_1	52999.93		3823ms
	tsp_318_1	47281.04		4404ms
	tsp_318_2	47281.04		3250ms
	tsp_400_1	17007.51		9948ms

Algoritmo	Entrada	Salida	Vector Elementos	Tiempo [Ms]
Opt_2 (Christofides)	tsp_5_1	4.00	0 1 2 3 4 0	0ms
	tsp_51_1	452.37	0 32 17 49 48 39 50 38 15 14 44 16 18 7 19 40 11 42 29 43 21 37 20 25 1 31 22 33 47 26 6 36 30 12 34 23 35 13 4 8 24 41 27 3 46 45 9 10 28 2 5 0	14ms
	tsp_70_1	749.41	0 35 50 11 1 22 14 44 16 68 55 5 40 6 52 38 36 53 41 66 37 42 60 19 13 62 48 54 64 8 32 7 51 43 61 10 9 59 3 4 25 26 46 24 67 18 12 31 45 28 29 20 15 39 17 30 58 49 21 27 56 23 65 33 69 34 63 47 2 57 0	18ms
	tsp_76_1	575.00	Vector demasiado largo para ser representado en la tabla	28ms
	tsp_76_2	112353.70		18ms
	tsp_99_1	1279.05		37ms
	tsp_100_1	677.74		45ms
	tsp_100_2	22607.23		48ms
	tsp_100_3	21843.69		76ms

	tsp_100_4	22474.47		39ms
	tsp_100_5	22374.83		40ms
	tsp_100_6	8541.34		52ms
	tsp_101_1	677.74		44ms
	tsp_105_1	16013.89		44ms
	tsp_107_1	45531.83		46ms
	tsp_124_1	59281.77		115ms
	tsp_127_1	123258.54		50ms
	tsp_136_1	102316.58		100ms
	tsp_144_1	66517.21		106ms
	tsp_150_1	27206.95		102ms
	tsp_150_2	27963.31		118ms
	tsp_152_1	74999.71		80ms
	tsp_159_1	44377.56		170ms
	tsp_195_1	2466.24		127ms
	tsp_198_1	16507.24		127ms
	tsp_200_1	30934.60		176ms
	tsp_200_2	30947.91		124ms
	tsp_225_1	130182.85		184ms
	tsp_226_1	83859.21		201ms
	tsp_262_1	2548.85		204ms
	tsp_264_1	51407.82		459ms
	tsp_299_1	50607.72		446ms
	tsp_318_1	44937.30		490ms
	tsp_318_2	44937.30		397ms

	tsp_400_1	16309.11		595ms

Algoritmo	Entrada	Salida	Vector Elementos	Tiempo [Ms]
Opt_3 (Christofides)	tsp_5_1	4.00	0 1 2 3 4 0	0ms
	tsp_51_1	439.29	0 5 2 28 10 9 45 27 3 46 41 24 8 4 13 35 23 34 12 30 36 6 47 26 1 25 20 37 21 43 29 42 11 40 7 19 18 16 44 14 15 38 50 39 49 17 32 48 31 22 33 0	1203ms
	tsp_70_1	691.26	0 57 2 47 63 34 69 65 33 23 56 27 30 17 39 44 16 24 67 46 68 55 51 7 5 40 6 52 38 36 21 49 58 53 41 66 37 42 60 19 13 62 48 54 64 8 32 43 61 10 4 25 9 59 3 26 18 12 31 45 28 29 20 15 14 22 1 11 50 35 0	2952ms
	tsp_76_1	568.12	Vector demasiado largo para ser representado en la tabla	4055ms
	tsp_76_2	112312.64		2907ms
	tsp_99_1	1264.61		6164ms
	tsp_100_1	666.14		9816ms
	tsp_100_2	22607.23		7923ms
	tsp_100_3	21201.40		8966ms
	tsp_100_4	21728.25		9158ms
	tsp_100_5	22245.11		7302ms
	tsp_100_6	8257.95		8027ms
	tsp_101_1	666.14		8705ms
	tsp_105_1	15600.80		8195ms
	tsp_107_1	45077.09		6899ms
	tsp_124_1	59569.19		14948ms
	tsp_127_1	124487.92		15928ms

	tsp_136_1	101573.53		18631ms
	tsp_144_1	62374.05		26993ms
	tsp_150_1	26987.61		25412ms
	tsp_150_2	27056.34		29500ms
	tsp_152_1	74340.48		30658ms
	tsp_159_1	43588.96		36850ms
	tsp_195_1	2453.24		60681ms
	tsp_198_1	16243.86		66852ms
	tsp_200_1	30559.05		72894ms
	tsp_200_2	30357.53		44809ms
	tsp_225_1	128535.01		73730ms
	tsp_226_1	82877.22		100595ms
	tsp_262_1	2515.09		151952ms
	tsp_264_1	50729.72		223307ms
	tsp_299_1	49442.70		234674ms
	tsp_318_1	44148.98		293647ms
	tsp_318_2	44148.98		220471ms
	tsp_400_1	15914.99		537220ms

Algoritmo	Entrada	Salida	Vector Elementos	Tiempo [Ms]
SA (Christofides)	tsp_5_1	4.00	0 1 2 3 4 0	2777ms
	tsp_51_1	447.22	0 33 32 48 17 49 39 50 38 15 14 44 16 18 19 7 40 11 42 29 43 21 37 20 25 1 31 22 47 26 6 36 30 12 34 23 35 13 4 8 24 46 3 41 27 45 9 10 28 2 5 0	3554ms

	tsp_70_1	713.97	0 35 50 11 1 22 14 15 20 29 28 45 31 12 18 26 25 3 59 9 10 61 4 43 51 7 32 8 64 54 48 62 42 13 19 60 53 36 41 66 37 38 52 6 40 5 55 68 46 67 24 16 44 39 17 30 27 56 23 21 58 49 69 34 63 47 65 33 2 57 0	3455ms
	tsp_76_1	577.59	Vector demasiado largo para ser representado en la tabla	4265ms
	tsp_76_2	111978.24		2810ms
	tsp_99_1	1274.25		3067ms
	tsp_100_1	726.20		1ms
	tsp_100_2	22744.78		2609ms
	tsp_100_3	21577.49		3508ms
	tsp_100_4	22480.49		3322ms
	tsp_100_5	22418.96		3183ms
	tsp_100_6	8292.26		3223ms
	tsp_101_1	705.11		1ms
	tsp_105_1	15568.80		3221ms
	tsp_107_1	46659.36		28ms
	tsp_124_1	59998.71		3228ms
	tsp_127_1	127481.05		74ms
	tsp_136_1	103575.40		2761ms
	tsp_144_1	64474.27		3768ms
	tsp_150_1	26806.84		3159ms
	tsp_150_2	27506.13		3535ms
	tsp_152_1	74596.96		4099ms
	tsp_159_1	43749.71		603ms
	tsp_195_1	2455.65		3365ms

	tsp_198_1	16574.49		433ms
	tsp_200_1	30313.31		3451ms
	tsp_200_2	30528.24		2276ms
	tsp_225_1	130509.58		72ms
	tsp_226_1	90538.68		9ms
	tsp_262_1	2511.01		5051ms
	tsp_264_1	51380.20		3664ms
	tsp_299_1	51453.43		506ms
	tsp_318_1	44933.03		3295ms
	tsp_318_2	44567.34		2429ms
	tsp_400_1	16163.35		2921ms

Algoritmo	Entrada	Salida	Vector Elementos	Tiempo [Ms]
tabu_annealing (Christofides)	tsp_5_1	4.00	0 1 2 3 4 0	3082ms
	tsp_51_1	446.53	0 33 32 17 49 48 22 31 39 50 38 15 14 44 16 18 7 19 40 11 42 29 43 21 37 20 25 1 26 47 6 36 30 12 34 23 35 13 4 8 24 41 46 3 27 45 9 10 28 2 5 0	4634ms
	tsp_70_1	734.85	0 35 50 11 1 22 14 44 16 55 68 46 24 15 20 29 28 45 12 31 67 18 26 25 3 59 9 10 61 4 43 51 7 32 8 64 54 48 13 19 60 42 62 37 66 41 53 36 38 52 6 5 40 39 17 30 27 21 58 49 23 56 2 65 33 69 34 63 47 57 0	7013ms
	tsp_76_1	567.14	Vector demasiado largo para ser representado en la tabla	5172ms
	tsp_76_2	111685.36		3450ms
	tsp_99_1	1268.91		3476ms
	tsp_100_1	670.88		5034ms

	tsp_100_2	22827.61		3695ms
	tsp_100_3	21647.86		4913ms
	tsp_100_4	22307.55		4416ms
	tsp_100_5	22281.21		4190ms
	tsp_100_6	8399.49		4514ms
	tsp_101_1	670.19		4614ms
	tsp_105_1	15626.15		3554ms
	tsp_107_1	45148.58		72ms
	tsp_124_1	59663.23		4546ms
	tsp_127_1	123346.03		4634ms
	tsp_136_1	103313.18		3891ms
	tsp_144_1	67736.63		178ms
	tsp_150_1	27011.42		3935ms
	tsp_150_2	28051.26		4608ms
	tsp_152_1	74863.76		3500ms
	tsp_159_1	43590.55		1047ms
	tsp_195_1	2458.88		4520ms
	tsp_198_1	16710.35		589ms
	tsp_200_1	30614.04		4719ms
	tsp_200_2	30360.41		3421ms
	tsp_225_1	129368.41		722ms
	tsp_226_1	90739.29		3ms
	tsp_262_1	2501.29		6072ms
	tsp_264_1	51008.66		5102ms
	tsp_299_1	50599.47		1169ms

	tsp_318_1	44741.21		4659ms
	tsp_318_2	44548.26		3431ms
	tsp_400_1	16105.31		4890ms

Algoritmo	Entrada	Salida	Vector Elementos	Tiempo [Ms]
genetic_cycle (Christofides)	tsp_5_1	4.00	0 1 2 3 4 0	0ms
	tsp_51_1	455.20	0 33 32 17 49 48 39 50 38 15 14 44 16 18 19 7 40 11 42 29 43 21 37 20 25 1 31 22 26 47 6 36 30 12 34 23 35 13 4 8 24 46 3 41 27 45 9 10 28 2 5 0	0ms
	tsp_70_1	1710.29	0 35 50 11 1 22 14 15 16 29 68 45 24 12 20 26 28 3 59 31 10 18 4 25 51 7 9 8 61 54 43 62 42 32 19 64 53 48 13 66 60 38 52 37 40 41 55 36 46 67 6 5 44 39 17 30 27 21 58 49 23 56 69 34 63 47 65 33 2 57 0	0ms
	tsp_76_1	819.20	Vector demasiado largo para ser representado en la tabla	1ms
	tsp_76_2	111859.32		1ms
	tsp_99_1	1406.63		1ms
	tsp_100_1	794.29		0ms
	tsp_100_2	22930.69		2ms
	tsp_100_3	22790.13		0ms
	tsp_100_4	32396.83		0ms
	tsp_100_5	22281.21		1ms
	tsp_100_6	8453.18		1ms
	tsp_101_1	876.25		1ms
	tsp_105_1	15626.15		1ms

	tsp_107_1	47063.62		1ms
	tsp_124_1	72621.52		1ms
	tsp_127_1	373157.47		1ms
	tsp_136_1	387155.74		0ms
	tsp_144_1	115659.05		1ms
	tsp_150_1	34912.16		2ms
	tsp_150_2	27582.67		3ms
	tsp_152_1	74979.26		1ms
	tsp_159_1	43749.71		1ms
	tsp_195_1	2495.68		1ms
	tsp_198_1	43313.67		1ms
	tsp_200_1	39292.32		0ms
	tsp_200_2	30562.94		1ms
	tsp_225_1	431373.37		3ms
	tsp_226_1	91934.48		1ms
	tsp_262_1	5346.67		4ms
	tsp_264_1	52885.60		3ms
	tsp_299_1	55142.52		4ms
	tsp_318_1	55483.94		8ms
	tsp_318_2	44894.83		2ms
	tsp_400_1	40854.06		11ms

Algoritmo	Entrada	Salida	Vector Elementos	Tiempo [Ms]
ant_algorithm	tsp_5_1	4.00	0 1 2 3 4 0	2ms
	tsp_51_1	740.38	0 5 2 20 25 50 11 37 21 43 39 49 48 31 22 17 32 33 47 26 6 1 42 29 44 15 16 18 38 14 7 23 34 12 36 4 24 35 40 19 41 27 28 45 9 10 46 3 8 13 30 0	42409ms
	tsp_70_1	1085.40	0 34 63 47 65 33 23 57 11 1 22 50 69 27 56 30 21 49 58 42 62 37 32 8 54 64 48 60 66 41 36 53 6 52 38 40 5 7 51 25 4 10 61 9 59 3 43 19 13 2 14 16 20 15 67 18 26 12 28 31 29 45 55 68 46 24 44 39 17 35 0	176204ms
	tsp_76_1	919.82	Vector demasiado largo para ser representado en la tabla	248684ms
	tsp_76_2	338422.68		77547ms
	tsp_99_1			Más de 5 minutos

4. Facility Location

Facility Location tiene como objetivo obtener la ubicación óptima de unas ciertas instalaciones para minimizar los costos de transporte y de apertura.

4.1 MIP

```
"""
    Variables de Decisión:
    ci = cliente
    aj = almacén
        Ci -> Aj
        Aj -> 1/0 [1 si se abre, 0 si no se abre]
    Restricciones:
        sum(Ci)<=1
        sum(demandas(Ci))<=Capacidad(Aj)
        Ci -> Aj si Aj es 1
    Función Objetivo:
        Minimizar C(t)
        C(t) = C(Aperturas) + C(Trayectos)
        C(Aperturas) = sum(Capertura(Aj)) si Aj es 1
        C(Trayectos) = sum(Cdistancia(Ci -> Aj))
"""
```

4.2 Algoritmo Usado para la Corrección

Evidentemente se utilizará MIP para resolver el problema. Estará capado a unos 10 minutos su ejecución. (La tabla que se muestra a continuación es sin límite temporal)

4.3 Análisis temporal

Algoritmo	Entrada	Salida	VectorElementos	Tiempo[ms]
MIP	fl_3_1	2545.77	0 0 1 2	13ms
	fl_16_1	3523677.87	6 6 15 14 6 9 11 15 6 6 10 15 14 6 9 9 6 13 6 6 6 15 9 6 15 6 12 6 9 6 6 6 15 1 6 6 3 11 9 6 14 7 9 9 7 9 6 6 13 6	391ms
	fl_16_2	889538.38	0 4 2 3 14 2 12 4 3 14 14 0 7 14 0 10 12 11 7 10 2 11 2 3 4 1 0 2 2 10 3 10 10 1 12 10 10 0 4 12 0 4 2 11 4 2 1 4 11 14	75ms
	fl_25_1	3406660.76	Etc.	84ms
	fl_25_2	3269821.32		58ms
	fl_25_3	3308526.93		64ms
	fl_25_4	1103338.51		86ms
	fl_25_5	1446560.76		78ms
	fl_50_1	2825999.22		148ms
	fl_50_2	2879410.36		166ms
	fl_50_3	2854410.36		132ms
	fl_50_4	2916910.36		195ms
	fl_50_5	1157235.26		373ms
	fl_50_6	3732793.43		416ms
	fl_100_1	Más de 3 min		Más de 3 min
	fl_100_2	Más de 3 min		Más de 3 min
	fl_200_1	3807.32		1757ms
	fl_200_2	3964.14		1850ms
	fl_200_3	3694.74		1967ms
	fl_200_4	3916.50		1894ms
	fl_200_5	4022.52		1962ms
	fl_200_6	Más de 3 min		Más de 3 min
	fl_500_6	Más de 3 min		Más de 3 min

5. Graph Coloring

Graph Coloring o coloreado de grafos, consiste en colorear el grafo con el mínimo número de colores tal que ningún vértice adyacente comparta el mismo color.

5.1 Minizinc

Input:

```
int: nSize;
int : nRestrictions;
```

Constantes:

```
set of int: size = 1..nSize;
set of int: nrestrictions = 1..nRestrictions;
```

Variables de decisión:

```
array[size] of int : countries;
array[size] of var 1..nSize : coloresPaises;
array[nrestrictions] of int : A;
array[nrestrictions] of int : B;
```

Restricciones:

```
% Usamos value_precede como restricción lexicográfica, obligando a que dentro de 'coloresPaises',
% antes de aparecer un nuevo color deben haber aparecido todos los de detrás.

constraint forall(i in 1..nSize-1)(value_precede(i,i+1,coloresPaises));
constraint forall(i in nrestrictions) (coloresPaises[A[i]] != coloresPaises[B[i]]);
```

Objetivo:

```
solve minimize(max(coloresPaises));
```

5.2 Local Search

La búsqueda local es un método heurístico para resolver problemas de optimización computacionalmente difíciles. Consiste en recorrer el espacio de soluciones, hasta que se encuentra una solución que se considera óptima o hasta que se ha transcurrido un límite de tiempo.

5.3 MIP

'''

Variables de Decisión:

Colores de los Países

Restricciones:

1 país con 1 conexión tiene distintos colores

Función Objetivo:

Minimizar el número de colores

'''

Un **clique** en un grafo es un conjunto de vértices, $C \subseteq V$, tal que todo par de vértices distintos son adyacentes, es decir, existe una arista que los conecta. En otras palabras, un clique es un subgrafo en el que cada vértice está conectado a todos los demás vértices del subgrafo. Para la resolución del problema hemos aplicado cortes con cliques de **tamaño 3 y 4.**

5.4 Greedy Networkx

Networkx **suministra diversos algoritmos que permiten resolver el problema de graph coloring.** En el caso de problemas con muchos edges, MIP es inviable por lo tanto hemos realizado un greedy con **Networkx** con todos estos algoritmos proporcionados con networkx y tomamos el mejor valor de dichos algoritmos.

5.5 Algoritmo Usado para la Corrección

En vista a los tiempos que se exponen a continuación, se ha optado por:

1. Usar **MIP** hasta un tamaño de 200.
2. Usar **Greedy networkx** desde un tamaño de 200.

5.5 Análisis temporal

Algoritmo	Entrada	Salida	Vector Elementos	Tiempo [ms]
Básico	gc_4_1	2	1 2 1 1	509
Básico	gc_20_1	3	1 1 2 1 1 3 2 2 2 2 1 1 1 2 2 1 3 3 1 1	489
Básico	gc_20_3	5	1 2 1 3 2 2 1 4 2 1 3 4 4 5 4 4 2 2 3 1	513
Básico	gc_20_5	5	1 2 3 2 4 1 5 2 4 3 4 3 3 4 2 1 5 5 1 3	771
Básico	gc_20_7	8	1 2 3 2 4 4 1 5 1 4 6 3 7 6 5 8 5 7 8 5	738
Básico	gc_20_9	11	1 2 3 4 1 5 6 7 5 4 8 9 10 8 7 10 2 2 3 11	650
Básico	gc_50_1	4	1 2 1 1 1 2 2 2 1 2 2 2 2 3 3 2 4 4 1 1 2 3 3 1 3 4 2 1 1 3 3 2 4 4 1 1 3 2 2 2 1 4 2 3 3 2 3 4 1 3	631
Básico	gc_50_3	6	1 1 1 2 2 3 1 4 3 5 3 4 3 5 1 6 5 4 6 4 6 2 3 3 3 4 6 4 2 2 1 5 1 3 5 3 4 5 1 2 1 6 2 6 1 6 2 5 2 5	998
Básico	gc_50_5	9	1 1 1 2 3 4 5 6 3 7 6 6 8 3 3 3 1 6 8 4 9 9 7 6 6 1 9 2 2 4 7 7 2 5 3 1 4 8 8 9 5 7 5 9 3 2 4 2 5 4	2147
Básico	gc_50_7	14	1 2 2 3 4 5 6 2 7 8 7 5 9 3 6 7 3 10 10 11 8 12 4 9 10 11 5 9 13 7 8 4 2 12 14 13 12 10 11 2 6 1 14 3 8 1 7 14 13 1	25794
Básico	gc_50_9	Más 2 minutos		
Básico	gc_70_1	4	Etc.	470
Básico	gc_70_3	8	Etc.	14654
Básico	gc_70_5	Más 2 minutos		
Básico	gc_70_7			
Básico	gc_70_9			
Básico	gc_100_1	5	Etc.	704
Básico	gc_100_3			
Básico	gc_100_5			
Básico	gc_100_7			
Básico	gc_100_9			
Básico	gc_250_1			
Básico	gc_250_3			
Básico	gc_250_5			
Básico	gc_250_7			
Básico	gc_250_9			
Básico	gc_500_1			
Básico	gc_500_3			
Básico	gc_500_5			
Básico	gc_500_7			
Básico	gc_500_9			
Básico	gc_1000_1			
Básico	gc_1000_3	Más 2 minutos		

Básico	gc_1000_5			
Básico	gc_1000_7			
Básico	gc_1000_9			
MIP	gc_4_1	2	0 1 0 0	28ms
	gc_20_1	3	1 2 0 0 0 0 1 1 1 1 0 2 0 2 2 0 2 1 0 0	54ms
	gc_20_3	5	0 0 2 1 2 0 1 2 0 3 1 4 0 2 0 0 0 0 1 2	241ms
	gc_20_5	5	3 2 4 2 1 3 0 2 1 4 1 4 4 1 2 3 0 0 3 4	270ms
	gc_20_7	8	6 1 3 1 4 4 6 5 6 4 2 3 2 7 5 0 5 2 0 5	118ms
	gc_20_9	11	0 3 7 5 0 2 1 9 2 1 6 1 0 4 6 9 4 5 3 7 8	113ms
	gc_50_1	4	0 3 2 3 1 3 0 0 2 1 3 1 0 0 1 2 0 0 2 3 1 1 1 2 2 1 0 3 0 1 3 0 1 1 0 2 0 0 0 3 0 1 1 1 2 0 2 1 3 2	527ms
	gc_50_3	6	Etc.	11709ms
	gc_50_5	[...]		Más de 3 min
	gc_50_7	[...]		
	gc_50_9	[...]		
	gc_70_1	4		739ms
	gc_70_3	[...]		Más de 3 min
	gc_70_5			
	gc_70_7			Más de 3 min
	gc_70_9			Más de 3 min
	gc_100_1	5		9455ms
	gc_100_3	[...]		Más de 3 min

Algoritmo	Entrada	Salida	Vector Elementos	Tiempo [ms]
Búsqueda Local	gc_4_1	2	0 1 0 0	0ms
Búsqueda Local	gc_20_1	3	0 0 2 2 1 0 1 2 1 1 1 1 0 0 1 1 1 0 0 0	0ms
Búsqueda Local	gc_20_3	5	3 1 2 0 1 1 2 3 4 2 0 3 3 1 3 3 1 1 0 2	1ms
Búsqueda Local	gc_20_5	7	0 1 2 3 4 0 3 6 4 2 4 1 1 4 1 0 3 5 0 2	1ms
Búsqueda Local	gc_20_7	9	0 1 2 1 4 4 0 7 3 4 2 11 7 5 6 0 7 3 11 5	1ms
Búsqueda Local	gc_20_9	11	0 1 2 3 0 5 6 7 5 6 9 11 4 8 7 4 1 1 2 8	5ms
Búsqueda Local	gc_50_1	5	0 1 2 1 0 3 2 0 1 2 0 2 0 1 0 1 2 2 3 0 0 3 0 1 1 2 3 1 0 0 1 0 2 0 2 2 3 1 3 0 2 2 0 4 1 2 3 2 0 0	2ms
Búsqueda Local	gc_50_3	8	1 1 2 3 1 5 3 2 4 1 5 2 5 2 7 7 2 7 7 0 0 1 0 6 4 0 3 5 1 1 6 3 1 0 5 0 5 4 7 2 7 1 4 2 3 2 4 3 1 6	8ms
Búsqueda Local	gc_50_5	12	0 1 2 1 1 5 6 9 8 2 4 4 7 4 9 4 0 4 6 7 8 3 11 10 4 0 3 6 1 5 2 7 4 3 0 3 3 2 8 3 9 3 5 6 1 9 9 6 5 5	15ms
Búsqueda Local	gc_50_7	17	0 1 2 3 4 5 6 5 8 9 7 5 12 2 6 15 3 15 11 1 9 11 4 12 11 13 6 12 10 10 9 0 8 2 14 10 3 9 1 12 6 14 14 7 17 14 7 13 10 17	26ms
Búsqueda Local	gc_50_9	25	Etc.	49ms
Búsqueda Local	gc_70_1	6	Etc.	5ms
Búsqueda Local	gc_70_3	10	Etc.	27ms
Búsqueda Local	gc_70_5	15	Etc.	44ms

Búsqueda Local	gc_70_7	21	Etc.	68ms
Búsqueda Local	gc_70_9	33	Etc.	171ms
Búsqueda Local	gc_100_1	6	Etc.	18ms
Búsqueda Local	gc_100_3	13	Etc.	68ms
Búsqueda Local	gc_100_5	19	Etc.	130ms
Búsqueda Local	gc_100_7	29	Etc.	260ms
Búsqueda Local	gc_100_9	45	Etc.	556ms
Búsqueda Local	gc_250_1	12	Etc.	330ms
Búsqueda Local	gc_250_3	25	Etc.	1478ms
Búsqueda Local	gc_250_5	40	Etc.	3536ms
Búsqueda Local	gc_250_7	60	Etc.	6652ms
Búsqueda Local	gc_250_9	99	15360ms	
Búsqueda Local	gc_500_1	18	3437ms	
Búsqueda Local	gc_500_3	40	17079ms	
Búsqueda Local	gc_500_5	71	Etc.	44491ms
Búsqueda Local	gc_500_7	106	Etc.	94020ms
Búsqueda Local	gc_500_9	171	Etc.	190506ms
Búsqueda Local	gc_1000_1	30	Etc.	39869ms
Búsqueda Local	gc_1000_3	73	Etc.	217774ms
Búsqueda Local	gc_1000_5	121	Etc.	514176ms
Búsqueda Local	gc_1000_7	Más de 5 minutos		
Búsqueda Local	gc_1000_9			

Greedy networkx	gc_250_1	10	Etc.	460ms
Greedy networkx	gc_250_3	22	Etc.	940ms
Greedy networkx	gc_250_5	37	Etc.	1281ms
Greedy networkx	gc_250_7	53	Etc.	1647ms
Greedy networkx	gc_250_9	91	2671	
Greedy networkx	gc_500_1	16	2466ms	
Greedy networkx	gc_500_3	38	4335ms	
Greedy networkx	gc_500_5	63	Etc.	7268ms
Greedy networkx	gc_500_7	98	Etc.	9782
Greedy networkx	gc_500_9	157	Etc.	15357ms
Greedy networkx	gc_1000_1	26	Etc.	13557ms
Greedy networkx	gc_1000_3	64	Etc.	37210ms
Greedy networkx	gc_1000_5	108	Etc.	43043ms
Greedy networkx	gc_1000_7	166	70680ms	
Greedy networkx	gc_1000_9	277	112628ms	

6. Set Covering

Set Covering Problem (SCP) o conjunto de cobertura es un problema clásico, que pertenece a la clase NPCompleto, donde la entrada está dada por varios conjuntos de elementos o datos que tienen algún elemento en común. En general, estos problemas consisten en encontrar un conjunto de soluciones que permitan cubrir en forma total o parcial un conjunto de necesidades al menor costo posible.

6.1 Minizinc

Input:

```
int: ITEM_COUNT;
int: NSET_COUNT;
array[SET_COUNT] of int: cost;
```

Constantes:

```
set of int: NODES = 0..ITEM_COUNT-1;
set of int: SET_COUNT = 1..NSET_COUNT;
```

Variables de decisión:

```
array[SET_COUNT] of var set of NODES: sets;
array[SET_COUNT] of var 0..1: selected_ones;
```

Restricciones:

```
constraint forall(node in NODES)(sum(nset in SET_COUNT)
                    (selected_ones[nset]*(node in sets[nset])) >= 1);
```

Objetivo:

```
var int: totalCost = sum(i in SET_COUNT)(cost[i] * selected_ones[i]);
solve minimize(totalCost);
```

6.2 MIP

Se ha realizado una implementación en **Gurobi** que llega a ejecutar entradas del doble de tamaño que Minizinc. Las variables de decisión, objetivo y restricciones son las mismas que las usadas por Minizinc, así que no se explicarán en detalle. Además, se le ha puesto un tiempo límite de 10 minutos como al resto de implementaciones de Gurobi para facilitar la corrección.

6.3 Algoritmo Usado para la Corrección

Se usará Minizinc para tamaños <= 75 y MIP para el resto de casos.

6.3 Análisis temporal

Algoritmo	Entrada	Salida	Vector Elementos	Tiempo [ms]
Básico	sc_6_1	2	0 0 0 0 1 1	557
Básico	sc_9_0	5	1 1 1 1 1 0 0 0 0	452
Básico	sc_15_0	9	1 1 1 1 1 1 0 1 1 1 0 0 0 0 0	507
Básico	sc_25_0	6	0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0	515
Básico	sc_27_0	18	1 1 1 1 0 1 1 1 1 1 1 1 0 1 0 1 1 0 0 1 1 1 1 1 0 0 0	605
Básico	sc_45_0	30	0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0	17165
Básico	sc_81_0	Minizinc no es lo suficientemente potente para poder llegar a una solución en un tiempo aceptable.		

Algoritmo	Entrada	Salida	Vector Elementos	Tiempo [ms]
MIP	sc_6_1	2	0 0 0 0 1 1	7
	sc_9_0	5	0 0 0 1 1 1 0 1 1	16
	sc_15_0	9	0 0 0 0 0 1 1 1 1 1 0 1 1 1 1	37
	sc_25_0	6	0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1	9
	sc_27_0	18	1 1 1 1 1 1 0 0 0 1 1 0 1 0 1 0 1 1 0 1 1 1 0 1 1 1 0	61
	sc_45_0	30	1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1	1112
	sc_81_0	61	[Etc.]	1581
	sc_135_0	[Más de 10 min]		

7. Vehicle Routing

Vehicle Routing o problema de enrutamiento de vehículos, consiste en obtener cual es la ruta más óptima para una flota de vehículos que debe satisfacer las demandas de un conjunto de clientes. Es una Variación del conocido Problema del Viajante con unas ciertas restricciones añadidas.

7.1 MIP

Se ha utilizado MIP para resolver el VRP de la siguiente manera (explicada también en el código):

- **Variables de Decisión:**
  - **Recorridos usados** => Si una traza es utilizada, estará puesta a uno. Ejemplo: Traza (1,3) a 1 significa que se va del cliente 1 al 3.
  - **Capacidad de un Vehículo a la hora de tratar con un Cliente** => Una lista del tamaño de los clientes totales que marca la capacidad de un vehículo (lo ocupado que se encuentra concretamente) después de tratar con un cliente. Ejemplo: capacidad_vehiculo[1] = 30 indica que el vehículo al atender al cliente 1 tuvo 30 de capacidad lleno. Evidentemente, este número no puede ser mayor a la capacidad total del vehículo. Dicha variable tiene un lower bound de la demanda de dicho cliente (Valor mínimo que debería tener) y un upper bound de la capacidad del vehículo (Valor máximo que debería tener).
- **Restricciones:**
  - Solo puede haber una traza [i, cualquierColumna] y una traza [cualquierFila, j]. De esta manera se evita que exista (1,3) y (1,4) o (2, 4) y (3,4). Sin embargo no se evita que pueda existir (1,3) y (3,2), que es el caso que se puede dar en el algoritmo.
  - Además, se tienen dos restricciones para la capacidad-demanda. Estas restricciones se diferencian en que la primera se aplica principalmente cuando sale un vehículo del almacén y la segunda cuando el vehículo se encuentra circulando.
    - La primera sostiene en resumen que la capacidad ocupada de un vehículo cuando atiende a un cliente en caso de salir del almacén tendrá que ser menor o igual la demanda de dicho cliente.
    - La segunda sostiene en resumen que la capacidad ocupada de un vehículo cuando atendió al cliente anterior menos la capacidad de un vehículo cuando atendió al nuevo cliente si existe dicha traza (debería dar negativo evidentemente) deberá ser menor o igual a la demanda de dicho cliente nuevo en negativo. Con esto se quiere decir que en caso de que el coche no pase por un almacén, la capacidad de ocupación del vehículo tras pasar por un nuevo cliente deberá incrementarse en la demanda de dicho cliente.

    - Por último, se tiene la restricción de que del almacén solo pueden salir como máximo el número de vehículos disponibles.
- **Función Objetivo**
    - La función objetivo será minimizar el coste de las aristas escogidas (su distancia).

El algoritmo pasa para representar la salida por un método llamado *subtour* para indicar el trayecto de cada vehículo en el formato que se pedía.

7.2 Algoritmo Usado para la Corrección

Evidentemente se utilizará MIP para resolver el problema. Estará capado a unos 10 minutos su ejecución. (La tabla que se muestra a continuación es sin límite temporal)

7.3 Análisis temporal

Algoritmo	Entrada	Salida	Vector Elementos	Tiempo [Ms]
MIP	Vrp_5_4_1	68.28	68.28 1 0 2 1 0 0 4 3 0 0 0 0 0	28ms
	Vrp_16_3_1	278.73	278.73 1 0 1 3 8 7 6 0 0 11 2 9 12 0 0 14 13 4 15 10 5 0	31053ms
	Vrp_16_5_1	334.96	334.96 1 0 2 3 1 0 0 6 11 5 0 0 7 8 0 0 9 10 15 12 0 0 14 13 4 0	29948ms
	Vrp_21_4_1	362.41	362.41 1 0 4 15 3 9 0 0 8 7 5 2 0 0 12 10 18 16 13 14 0 0 17 1 6 19 11 20 0	300317ms
	Vrp_21_6_1	[Más de 5 min]		

8. Optativos

a. **8 Queens Problem**

Consiste en una pequeña implementación en MiniZinc del problema de las 8 reinas con dos versiones:

- Una versión comentada y más larga.
- Una versión más precisa.

b. **Magic Square**

Consiste en una implementación en MiniZinc donde **se rompe la simetría** para facilitar la búsqueda del resultado, mejorando la versión mostrada en clase.