Universidad Autónoma Metropolitana Unidad Azcapotzalco
División de Ciencias Básicas e Ingeniería
Licenciatura en Ingeniería en Computación

Proyecto de investigación

Algoritmo y heurística para incrustar métricas en una línea

Saúl Martínez Juárez
2113000992
Asesores de proyecto terminal

Dr. Francisco Javier Zaragoza Martínez
Profesor Titular
Departamento de Sistemas

Trimestre 2017 Invierno

23 de marzo de 2017

Yo, Francisco Javier Zaragoza Martínez, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Yo, Saúl Martínez Juárez, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Resumen

En este proyecto se explora una solución para el problema de incrustación de una métrica en una línea recta, que ocupe el espacio más pequeño posible. Una métrica es una forma de representar espacios usualmente como una matriz.

La principal motivación para este trabajo es que el problema de incrustación es una constante en los ámbitos de la información hoy en día, tener virtualmente la misma información en un espacio menor es cada vez de mayor interés, un decremento dramático para el tamaño de la información como el que proponemos es valioso para cualquier aplicación tecnológica. De manera paralela se encuentra una solución para el problema del agente viajero cuando las distancias entre dos destinos son muy similares, tema que tiene gran peso en logística de transportes.

La solución determinística incluye una exploración completa aunque veloz de las soluciones en casos pequeños mientras que la solución heurística construye una solución cercana a la óptima en un tiempo razonable para todos los casos.

Abstract

In this project we explore a solution for the metric embedding on a straight line problem, which uses the smallest possible space. A metric is a way of representing any space, usually in a matrix.

The main motivation for this work is that the problem of embedding is quite common in the scope of information. To have virtually the same information in a smaller space (with smaller price) is increasingly interesting, a dramatic decrement in the size of information like the one we propose is valuable for any technological application. In parallel, there is a solution to the traveling salesman problem when the distances between any two vertex are very similar, topic that has big interest in transportation logistics.

The deterministic solution includes a complete, but quick exploration on small cases while the heuristic solution builds a near-optimal solution in a reasonable time for all cases.

Índice

Índice de figuras

Índice de tablas

1. Introducción

El presente trabajo es acerca de la resolución de un problema planteado de manera original como la incrustación de la métrica en la línea recta. Los temas que aborda son:

1. Incrustación: Cuando tenemos una estructura matemática contenida en otra estructura, tal que la primera estructura conserve su congruencia aún cuando esté incrustada en la segunda estructura.

2. Espacio métrico: Una métrica se compone de una función para medir distancias y un arreglo de puntos, estos dos se conjuntan en una matriz de distancias entre los puntos.

La característica principal de un espacio métrico representado en una matriz es que las distancias entre los puntos son uniformes, no existen puntos que entre ellos esté demasiado distantes, ni demasiado cercanos.

La razón por la cual este trabajo fue desarrollado, es porque se cree que es un problema NP-Completo, lo que quiere decir que incluso si la parte del cálculo de las soluciones ya está hecho, nos toma mucho tiempo diferir si son o no solución del problema.

La investigación de este proyecto se realizó incluso aunque los fines profesionales pueden ser motivos mucho más lucrativos, el interés inicial fue que despierta su resolución a nivel académico, ya que el problema planteado es un subproblema del problema del agente viajero.

La estrategia fue diseñar buscando eficacia tanto en número de operaciones como en la memoria a utilizar en algoritmo determinístico y rapidez y congruencia en el algoritmo heurístico. Para esto en el algoritmo determinístico nos apoyamos en el algoritmo de Heap, bueno para producir permutaciones en el menor número de operaciones posible. Para el algoritmo heurístico buscamos con apoyo de la probabilidad, encontrar secuencialmente los puntos consecutivos.

2. Antecedentes

Hay algunos problemas que se asemejan o con los que comparte algunas características:

El problema de distribución de suelos es un problema que enfrenta la ingeniería industrial a maximizar la producción por unidad de suelo [8].

El problema de la administración de recursos en un fondo común cercano para evitar requerirlos de un lugar más costoso, siendo estos lo más heterogéneos posible [7].

También se parece al problema del agente viajero [6], con la diferencia de que la gráfica inicial de la que buscaremos el recorrido no está inicial ni estáticamente ponderada.

El proyecto terminal Encajes primitivos de gráficas planares exteriores [1] busca incrustar grafos en espacios (gráficas) bidimensionales. Se parece al proyecto pues logra hacerlo en un espacio reducido, acercándose al mínimo en tiempo polinomial, lo cual es un buen precedente. Se diferencia en que las gráficas son estrictamente planares y en este trabajo son métricas.

El proyecto terminal Encajes primitivos de árboles planos [2] incrusta árboles, que son un subconjunto de las gráficas, se diferencia en que dado que son árboles, no siempre hay un camino directo de un vértice a otro sino que hay una trayectoria de por medio. Esto es un caso especial de nuestro problema en el que las trayectorias son de tamaño 1.

El proyecto terminal Diseño de reglas de Golomb óptimas [3] nos ayuda mucho en el diseño de algoritmos en la fase entera, ya que hay buenos resultados en este campo y especialmente en este proyecto, pues genera una serie que podemos reutilizar. Sin embargo, en nuestro proyecto no siempre se generan reglas de Golomb, cosa que es requisito en esta referencia.

Finalmente el problema del incrustamiento con mínima distorsión [4], es el más parecido pues sólo hay que declarar que es una métrica lo que se va a incrustar.

3. Justificación

El interés principal del problema de incrustación radica en la compresión de datos; los archivos al ser comprimidos pueden ser transportados y almacenados a menor costo. Mientras nuestros recursos sean limitados (no infinitos) nos interesará reducir cualquier tipo de costo. En la actualidad, el trabajo de compresión es muy valorado, y con justa razón; cualquier aplicación de tecnología tiende a utilizar cada vez más información lo cual se convierte en un reto técnico.

Para almacenar métricas lo más común es guardarlas como matrices, simétricas de diagonal 0, y por consiguiente de tamaño $\Theta(n^2)$, donde n representa el total de puntos a representar. Este valor polinomial es el que es interesante reducir: si se incrusta la métrica en un espacio tridimensional, la métrica total termina midiendo $\Theta(n)$, por lo que el tamaño se redujo drásticamente. Lo mismo sucede con espacios bidimensionales y unidimensionales, como es el caso actual. Cualquier incrustación de métricas en espacios euclideanos n-dimensionales reducen la complejidad del espacio necesario para representarlas, por lo que la incrustación es necesariamente un método de compresión eficiente.

En ocasiones, métricas específicas no son las adecuadas para ciertas operaciones, por lo que es necesario traducirlas o interpretarlas hacia otra métrica, la conversión no es completamente biyectiva, por lo que tiende a tener un error, que en lenguaje de métricas se le conoce como distorsión. Para términos prácticos de compresión de los datos, esto es pérdida de información.

El producto final de este proyecto contiene especificaciones sobre el modo de manejar problemas de optimización en los que el conjunto de restricciones está incluida directamente en el producto final, que en este caso es una matriz

incrustada en una línea recta.

La complejidad computacional de este problema permanece abierta, y los resultados de este proyecto pueden contribuir al estudio de los problemas NP y a su clasificación.

4. Objetivos

4.1. Objetivo general

- Diseñar, implementar y evaluar un algoritmo exacto y una heurística para el problema de incrustación de una métrica en una línea.

4.2. Objetivos específicos

1. Diseñar e implementar los algoritmos que generen métricas.
2. Diseñar e implementar un algoritmo exacto que resuelva el problema.
3. Diseñar e implementar un algoritmo heurístico que resuelva el problema.
4. Evaluar los resultados de ambos algoritmos.

5. Marco teórico

Una función $d : X \times X \to \mathbb{R}$ es una métrica en un conjunto no vacío X si se satisfacen las siguientes condiciones para todo $x, y, z \in X$:

- Positividad: $d(x, y) > 0$; si $x \neq y$, y $d(x, x) = 0$.
- Simetría: $d(x, y) = d(y, x)$.
- Desigualdad del triángulo: $d(x, z) \leq d(x, y) + d(y, z)$.

A $d(x, y)$ se le llama la distancia de x a y. Por poner dos ejemplos de métricas, si $X = \mathbb{R}^n$ y $x = (x_1, x_2, ..., x_n)$, $y = (y_1, y_2, \ldots, y_n)$, entonces la métrica euclideana se calcula como:

$$d_E(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_n - y_n)^2} \tag{1}$$

y la métrica Manhattan se calcula como:

$$d_M(x, y) = |x_1 - y_1| + |x_2 - y_2| + ... + |x_n - y_n|. \tag{2}$$

Sea d_a una métrica en A y d_b una métrica en B. Entonces una función $f : A \to B$ es una incrustación con contracción c_f y expansión e_f si para cada par de puntos $p, q \in A$ se cumple que

$$\frac{d_a(p, q)}{c_f} \leq d_b(f(p), f(q)) \leq e_f \cdot d_a(p, q) \tag{3}$$

Figura 1: Gráfica original de tamaño 3.

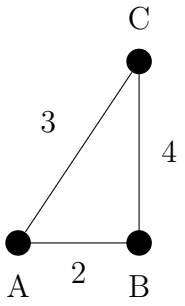


Figura 2: La longitud total de la línea es $z = 7$ ($\alpha = \frac{3+4}{2} = \frac{7}{2} = 3.5$).

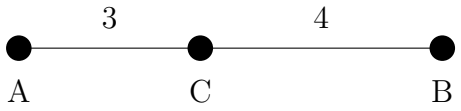


Figura 3: La longitud total de la línea es $z = 6$ ($\alpha = \frac{2+4}{3} = \frac{6}{3} = 2$).

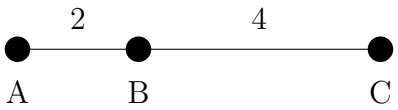
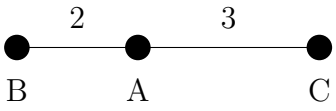


Figura 4: La longitud total de la línea es $z = 5$ ($\alpha = \frac{3+2}{4} = \frac{5}{4} = 1.25$)



Adicionalmente diremos que f no contrae si $c_f \leq 1$. Una función f que no contrae tiene distorsión α si $e_f \leq \alpha$.

La compresión de datos es la codificación de un cuerpo de datos D en un cuerpo de datos más pequeño D'. Una parte central en la compresión es la redundancia en los datos. Sólo los datos con redundancia pueden comprimirse aplicando un método o algoritmo de compresión que elimine o remueva de alguna forma dicha redundancia. La redundancia depende del tipo de datos (texto, imágenes, sonido, etc), por tanto, no existe un método de compresión universal que pueda ser óptimo para todos los tipos de datos. La compresión puede ser sin pérdida (reversible) o con pérdida (irreversible).

La expansión de una métrica es equivalente a su distorsión, y también equivalente a una compresión con pérdida. La razón por la que vale la pena tener una distorsión, es reducir la complejidad de un espacio. Dependiendo de su uso posterior, algunos espacios son más complejos que otros.

Ejemplo: Considere la gráfica de tamaño 3 con pesos en las aristas de la figura 1, que intentamos colocar en una línea recta, en el primer intento las colocamos en orden numérico, y la longitud total es 7 (ver figura 2). Podemos mejorarlo y en el siguiente intento, la longitud es 6 (ver figura 3). Sin embargo, este resultado aún puede mejorarse al incrustar en una línea de longitud 5 (ver figura 4). Observe que en estos ejemplos, a menor longitud, también menor distorsión α.

Entonces, el problema de incrustamiento de una métrica en una línea con distorsión mínima es un problema de optimización en el que incrustamos una

métrica en una línea recta con el menor espacio posible.

En este trabajo se implementará un método exacto y un método heurístico para resolver el problema de incrustación de una métrica en una línea con la menor longitud posible.

6. Desarrollo del proyecto

El desarrollo de este proyecto quedó dividido en tres partes:

- Generación de Métricas: Se utilizó una clase punto, en la que declaramos e implementamos el concepto de Punto, cada punto vive dentro de un espacio d-dimensional ($d \in [2, 10]$). Generamos puntos y tomamos sus coordenadas para luego generar una matriz que registre las distancias entre ellos.

- Algoritmo determinístico: Se utilizan varias técnicas tanto en el proceso como en el código final, todas con la misma estrategia de acción: generar una permutación, y compararla con respecto a las anteriores, determinar luego si era necesario recordarla o no.

- Algoritmo heurístico: Se utilizó principalmente la idea de un algoritmo glotón, que navega por el espacio de soluciones de manera que su solución mejora con cada paso que da.

6.1. Generación de Métricas

Tenemos tres generadores en los que a partir de la clase Punto podemos generar arreglos de puntos con la función $generaPuntos(int, int, short)$ y calcular las distancias entre ellos: el generador euclideano $genEuclideano(int, int, int)$ calcula la distancia con base en la distancia euclideana (1); el generador manhattan $genManhattan(int, int, int)$ calcula la distancia con base en la distancia manhattan (2); y el generador gráfico, $genGrafico(int, int)$ que sin la generación de puntos genera sólo distancias para luego con la función $floydWarshall(int, int**)$ seleccionar únicamente las distancias menores de un punto a otro.

6.2. Solución determinística

El algoritmo determinístico pasó por diferentes etapas antes de alcanzar su estado actual. Al principio para tener un mayor dominio del proceso de búsqueda de soluciones, se calculaban secuencialmente dependiendo de un número en el que estaban ordenadas y al final tenemos una solución que explora las soluciones posibles, mientras va evaluando su valor parcialmente, incrustando en el proceso de la búsqueda el cálculo y evitando volver a visitar las soluciones.

El algoritmo final consta de 5 funciones:

Figura 5: Pseudocódigo que ilustra el funcionamiento del algoritmo de Heap.

```
procedimiento genera(n : entero, A : Arreglo de algo):
    si n = 1 entonces
          imprime(A)
    si no
        para i := 0; i < n - 1; i += 1:
            genera(n - 1, A)
            si n es par entonces
                swap(A[i], A[n-1])
            si no
                swap(A[0], A[n-1])
            termina si
        termina para
        genera(n - 1, A)
    termina si
```

- $calcSol(short, int*, int**)$: Sirve para evaluar una solución dada en un vector con el orden de los puntos, requiere la longitud del vector, un vector y una matriz donde se encuentran las distancias.
- $evaluaMinimal(short, int**)$: Es una función preparatoria para empezar el proceso de permutación.
- $evenPermute(int, int, int*, int**, int)$: evenPermute es la rama de la solución que utilizamos cuando el número del procedimiento hacia la permutación que tenemos es un número par, dado que un número par nunca sera 1, esta rama siempre será no terminal, y nos llevará a un número impar luego de hacer la suma acumulada correspondiente.
- $oddPermute(int, int, int*, int**, int)$: oddPermute es la rama de la solución que utilizamos cuando el número que tenemos es un número impar, en cuanto el número sea 1, procedemos a terminar la suma acumulada.
- $permute(int, int, int*, int**)$: minimalPermute es un método que permite, mediante el uso del algoritmo de Heap como estructura, incrustar las operaciones de suma acumulada que necesitamos en el proyecto.

El algoritmo de Heap es un algoritmo recursivo que empieza con una variable contador en 0 y repetimos hasta que sea igual a N y mientras avanzamos, invocamos las $(N-1)$ permutaciones que tienen el último elemento en común. El siguiente pseudocódigo ejemplifica la dinámica del contador y la decisión sobre si el número es par o impar:

6.3. Solución heurística

En la solución heurística requerimos tener conocimiento de cual aproximadamente es el siguiente mejor punto a escoger. Por lo que sumamos los valores de todas las filas (o columnas) y los comparamos entre sí, de modo que el que tenga el menor valor inmediatamente tiene si no al valor más pequeño, a la suma de dos bastante pequeños. Luego restamos aquellos valores que ya hemos tomado en cuenta para seguir comparando a los demás y encontrar los puntos siguientes.

La solución heurística para el problema consta de 7 funciones:

- $copia(int, int**)$: A diferencia del algoritmo determinístico, en esta solución se requiere editar la matriz de distancias, por lo que hacemos una copia de seguridad y cambiamos los valores de la matriz, cambiando el objetivo de minimizar por maximizar de modo que los valores de cero en posiciones de la diagonal no afecten al desarrollo sin hacer comparaciones extra.

- $suma(int, int**)$: Genera la N columna de la matriz A, siendo esta la suma de la fila correspondiente.

- $compara(int, pair < int, int >, int **)$: Comparamos todos los valores de la columna de las sumas, y habiendo encontrado el mayor, comparamos los valores de la fila escogida, escogido el mayor tenemos ahora un par de puntos que nos indican las coordenadas en la matriz del valor mínimo.

- $aparta(int, int**, pair < int, int >, pair < int, int >)$: Ya que hemos elegido el valor mínimo que necesitamos, hacemos cero los valores que no deben seguir en las comparaciones, anulamos los valores que cierren la permutación, y los intermedios que ya no pueden ser escogidos.

- $compara\_primera(int, int**)$: Es el caso independiente de la función *compara* para la primera iteración, donde no tenemos en cuenta ninguna comparación anterior.

- $aparta\_primera(int, int **, pair < int, int >)$: Es el caso independiente de la función *aparta* para la primera iteración, donde no tenemos en cuenta ninguna comparación anterior y el número de anulaciones es mínimo.

- $gloton(int, int **)$: Es la función que engloba todas las demás, contiene la inicialización y la pauta de las iteraciones sobre *compara* y *aparta*.

7. Resultados

Para la solución determinística tenemos que dado que el algoritmo recorre todas las soluciones, el número de operaciones para recorrer es $n!$, mientras que en la suma, las operaciones son 4 veces esa cantidad menos cuatro veces (dos accesos, la suma entre ellos y la suma acumulada) que ya tenemos el resultado y

sólo es comparar $4 \cdot n! - 4$. Esto nos lleva a un total de $5n! - 4$ operaciones. En el peor de los casos ($n = 15$) el número de operaciones a realizar es $6.53837184 \cdot 10^{12}$ que tarda menos de 18 horas en terminar.

Para la solución heurística ocupamos las siguientes operaciones:

- *copia*: Necesitamos de $1 + 3n + 3n^2$ operaciones.
- *suma*: $2n + 3n^2$ operaciones.
- *compara*: $3n + 12$ operaciones.
- *aparta*: $6n + 4$ operaciones.
- *compara_primera*: $6n + 9$ operaciones.
- *aparta_primera*: 4 operaciones.

Efectuando la suma de todas las operaciones anteriores, queda un total de $15n^2 + 20n - 18$ operaciones, tomamos únicamente la parte cuadrática y ahora las operaciones son $15n^2$, de orden $\theta(n^2)$. Y para el peor de los casos en el que $n = 1000$ tenemos que el número de operaciones total es de 15000000, cifra pequeña comparada con la cantidad a la que se pudiera llegar en en algoritmo determinístico.

Como resultados, presentamos 70 valores de pruebas con puntos $n = 12$, dimensiones $d = 10$, rango de valores $l = 100$ y métrica euclideana.

Para las Tablas 1, 2 y las tablas que siguen podemos ver algunas de las siguientes 8 columnas:

- min: La distancia mínima global, encontrada por el algoritmo determinístico.
- t_D: El tiempo en milisegundos que tarda el algoritmo determinístico en encontrar min. Cabe mencionar que para el código que se encuentra en los entregables de donde obtenemos estos resultados, utilizamos la función $clock()$ de la biblioteca $time.h$ que regresa $ticks$, por lo que había que hacer una conversión dividiendo el resultado arrojado entre $TICKS\_PER\_SECOND$ que en la computadora usada para la evaluación era igual a 1000000. Por lo que un $ticks$ equivale a $1\mu s$, y $1000 ticks$ a $1ms$ que es la unidad de medida de tiempo que utilizamos.
- min_H: La distancia mínima encontrada por el algoritmo heurístico.
- t_H: El tiempo en milisegundos que tarda el algoritmo heurístico en encontrar min_H.
- ϵ: El error empírico, la diferencia entre min_H y min.
- $\epsilon_{\%}$: El error porcentual tomado de dividir ϵ entre min.

- $t_{A\%}$: Tiempo ahorrado, producto de dividir t_D menos t_H, entre t_D
- n: El número de puntos que conforman la métrica evaluada.

Tabla 1: Pruebas de comparación entre algoritmo determinístico contra heurístico.

min	t_D	min_H	t_H	ϵ	$\epsilon_{\%}$	$t_{A\%}$
908	26201.107	947	0.018	39	4.118270	99.991050
1054	26406.950	1136	0.020	82	7.218310	99.999924
1106	26528.356	1139	0.021	33	2.897280	99.999921
1050	26398.966	1114	0.023	64	5.745060	99.999913
1050	26468.729	1101	0.021	51	4.632150	99.999921
1051	26356.467	1103	0.019	52	4.714420	99.999928
1076	26390.431	1147	0.028	71	6.190060	99.999894
1041	26398.110	1139	0.024	98	8.604040	99.999909
931	27291.115	956	0.021	25	2.615060	99.999923
990	26813.083	1020	0.021	30	2.941180	99.999922
1009	26512.312	1026	0.027	17	1.656920	99.999898
1075	26497.678	1120	0.021	45	4.017860	99.999921
1291	26528.611	1349	0.022	58	4.299480	99.999917
981	26473.864	1013	0.021	32	3.158930	99.999921
1123	26434.045	1152	0.019	29	2.517360	99.999928
1006	25926.410	1050	0.019	44	4.190480	99.999927
1068	26555.939	1171	0.019	103	8.795900	99.999928
958	26128.766	1015	0.020	57	5.615760	99.999923
1078	26131.564	1126	0.022	48	4.262880	99.999916
1063	26212.811	1095	0.018	32	2.922370	99.999931
1060	25989.989	1115	0.020	55	4.932740	99.999923
1011	26247.894	1055	0.020	44	4.170620	99.999924
1071	26297.095	1133	0.020	62	5.472200	99.999924
1031	26171.812	1060	0.025	29	2.735850	99.999904
1160	26326.245	1224	0.022	64	5.228760	99.999916
999	26302.620	1054	0.021	55	5.218220	99.999920
1061	26237.790	1106	0.020	45	4.068720	99.999924
916	26161.955	1033	0.019	117	11.326200	99.999927
973	26926.302	1044	0.020	71	6.800770	99.999926
1059	26190.008	1114	0.021	55	4.937160	99.999920
1026	26373.266	1118	0.019	92	8.228980	99.999928
1036	26170.959	1073	0.019	37	3.448280	99.999927
1061	26201.441	1121	0.019	60	5.352360	99.999927
1174	26326.411	1218	0.020	44	3.612480	99.999924
1032	26136.981	1133	0.018	101	8.914390	99.999931

† .

Tabla 2: Continuación de la Tabla 1.

min	t_D	min_H	t_H	ϵ	$\epsilon_{\%}$	$t_{A\%}$
908	26197.016	947	0.019	39	4.118270	99.999927
1054	25840.734	1136	0.018	82	7.218310	99.999930
1106	26184.798	1139	0.018	33	2.897280	99.999931
1050	26235.115	1114	0.018	64	5.745060	99.999931
1050	26307.467	1101	0.027	51	4.632150	99.999897
1051	26316.716	1103	0.019	52	4.714420	99.999928
1076	26279.884	1147	0.019	71	6.190060	99.999928
1041	26282.789	1139	0.020	98	8.604040	99.999924
931	26263.607	956	0.019	25	2.615060	99.999928
990	26200.653	1020	0.020	30	2.941180	99.999924
1009	26212.855	1026	0.019	17	1.656920	99.999928
1075	25907.503	1120	0.019	45	4.017860	99.999927
1291	26166.378	1349	0.019	58	4.299480	99.999927
981	26243.127	1013	0.018	32	3.158930	99.999931
1123	27790.315	1152	0.019	29	2.517360	99.999932
1006	26296.059	1050	0.022	44	4.190480	99.999916
1068	26154.375	1171	0.019	103	8.795900	99.999927
958	26161.978	1015	0.021	57	5.615760	99.999920
1078	25868.246	1126	0.024	48	4.262880	99.999907
1063	26226.243	1095	0.018	32	2.922370	99.999931
1060	26322.103	1115	0.021	55	4.932740	99.999920
1011	26500.573	1055	0.023	44	4.170620	99.999913
1071	26347.304	1133	0.021	62	5.472200	99.999920
1031	26369.075	1060	0.023	29	2.735850	99.999913
1160	26267.478	1224	0.019	64	5.228760	99.999928
999	26216.909	1054	0.018	55	5.218220	99.999931
1061	25827.836	1106	0.019	45	4.068720	99.999926
916	26121.311	1033	0.018	117	11.326200	99.999931
973	26147.78	1044	0.019	71	6.800770	99.999927
1059	26180.339	1114	0.019	55	4.937160	99.999927
1026	26195.457	1118	0.021	92	8.228980	99.999920
1036	26306.031	1073	0.019	37	3.448280	99.999928
1061	26482.994	1121	0.019	60	5.352360	99.999928
1174	26162.097	1218	0.019	44	3.612480	99.999927
1032	26228.709	1133	0.024	101	8.914390	99.999908

† En estas obervaciones El promedio del error porcentual es 5.233394 %, el error mínimo porcentual es de 0.999092 % mientras que el máximo alcanza 11.3262 %, sin embargo puede ocurrir que el error sea igual a cero.

Otra muestra de resultados los tenemos en la Tabla 3, que arroja medidas sobre la evolución con respecto al número de puntos, para poder comparar la progresión de los tiempos, de las soluciones, del error, error porcentual y de la varianza.

Tabla 3: Pruebas de comparación entre algoritmo determinístico contra heurístico.

n	min	t_D	min_H	t_H	ϵ	$\epsilon_{\%}$	Δ_t
3	226.4	12.1	226.4	7.9	0	0	4.398106
4	328.8	13.6	343.4	8.3	14.6	4.102831	0.295275
5	424.1	19.4	461.9	9.2	37.8	7.999095	3.600989
6	521.7	55.3	544.9	10.2	23.2	4.055595	0.342511
7	613.1	296.3	636.3	10.7	23.2	3.644673	0.753433
8	716.7	2254.2	743.4	12.2	26.7	3.541959	0.856146
9	820.7	20085.8	869.5	16.3	48.8	5.603017	1.204911
10	903.2	200308.9	944.4	15.7	41.2	4.299304	0.098802
11	966.4	2205748	1020.7	17.4	54.3	5.278352	0.880246
12	1048.5	26597991.2	1109	30.5	60.5	5.456231	1.058125

† Esta tabla es un producto derivado, promedia los valores de las tablas que se encuentran en el Apéndice A. Tenemos un promedio de error porcentual de 4.398105 %.

En la Tabla 4 observamos el comportamiento de los tiempos y distancias mínimas encontradas en la heurística para $n > 12$. No hay datos de la prueba determinística en estos casos pues se tardaría demasiado, el último caso efectuado con el algoritmo determinístico arrojó el resultado después de casi 18 horas para $n = 15$.

Tabla 4: Pruebas de desempeño heurístico.

n	min	t_D
15	1340.3	42.7
20	1894.3	44.3
25	2290.3	59.7
30	2684.7	64.0
35	3013	88.3
40	3528	100.3
45	3792	117.3
50	4066.7	137
100	7759.7	436.7
200	14203	1688
300	20436	3718
400	26357.3	6259.3
500	32063	9759.7
600	37732.7	14647.7
700	42861	20033
800	48285	26611
900	52907.3	34490.3
1000	58648.3	44246

† Esta tabla es un producto derivado, promedia los valores de las tablas que se encuentran en el Apéndice B.

Por último verificamos los datos para la solución determinística en la Figura 10 y la Figura 11 con los tiempos en los que se ejecuta la solución determinística y el valor mínimo de la distancia encontrado.

8. Análisis y discusión de resultados

Podemos observar de los resultados en las Tablas 1 y 2 que aunque las cifras en cuanto a minimización de la distancia objetivo sean relativamente similares, los tiempos cambian mucho, mientras que el error se mantiene cerca del 5 % aproximadamente. El error mínimo porcentual es del 1 % aproximadamente mientras que el máximo se encuentra por el 11 %.

En la Tabla 3 podemos observar la evolución de las pruebas conforme aumentamos el número de puntos a evaluar, evidentemente el valor a optimizar crece, el tiempo en el algoritmo determinístico crece con mucha rapidez, conforme a su orden factorial; el tiempo en el algoritmo heurístico también crece pero es menor, pareciese constante pero de igual modo es polinomial. El error y el error porcen-

Figura 6: Gráfica de puntos contra tiempo para $n > 12$.
Relación entre número de puntos y tiempo en el que resuelve.

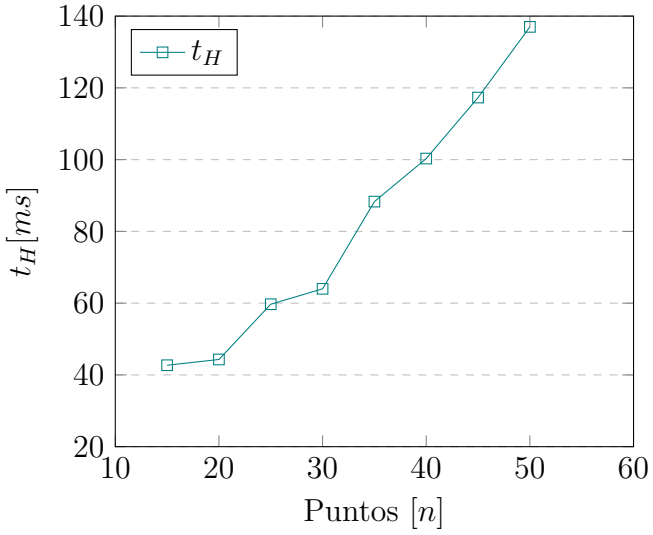


Figura 7: Gráfica de puntos contra tiempo para n muy grandes.
Relación entre número de puntos y tiempo en el que resuelve.

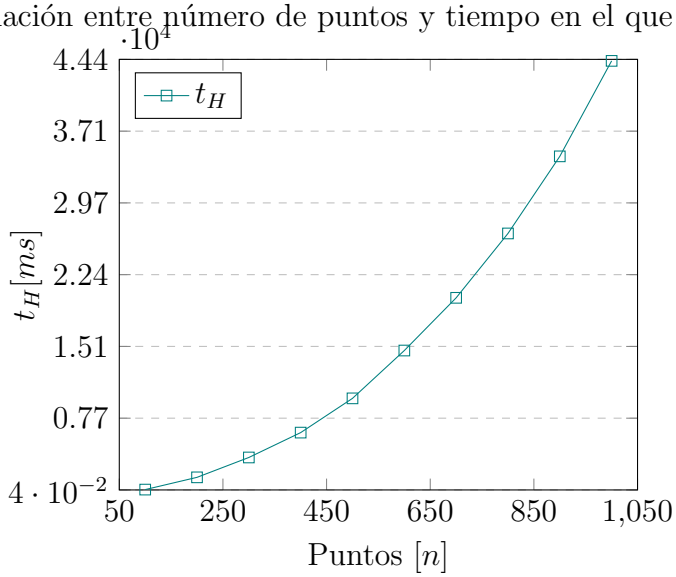


Figura 8: Gráfica de puntos contra distancia para $n > 12$.
Relación entre número de puntos y distancia mínima encontrada.

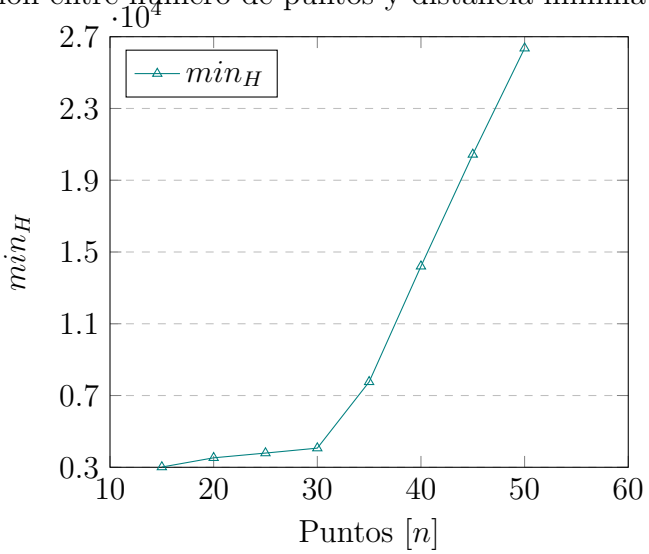
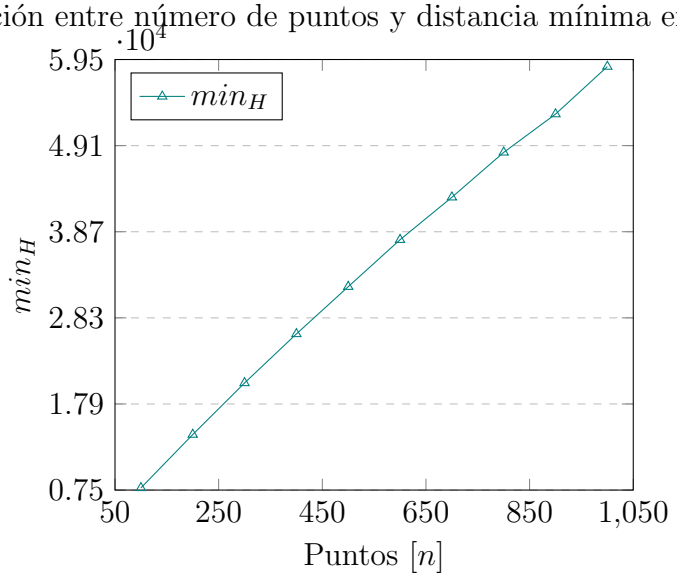


Figura 9: Gráfica de puntos contra distancia para n muy grandes.
Relación entre número de puntos y distancia mínima encontrada.



tual parecen no tener una tasa de cambio fija, incluso el promedio del error y error porcentual son congruentes con los datos anteriores, por lo que podríamos pensar que el error se comporta de la misma manera aunque el número de puntos sea mayor.

En la Figura 10 podemos verificar que el tiempo necesario para determinar la solución óptima es mayor que polinomial, mientras que en la Figura 11 notamos el parecido con las soluciones heurísticas en las que además en la pendiente notamos apenas diferencia con las pruebas heurísticas.

Figura 10: Gráfica de puntos contra tiempo para $n < 12$.
Relación entre número de puntos y tiempo en el que resuelve.

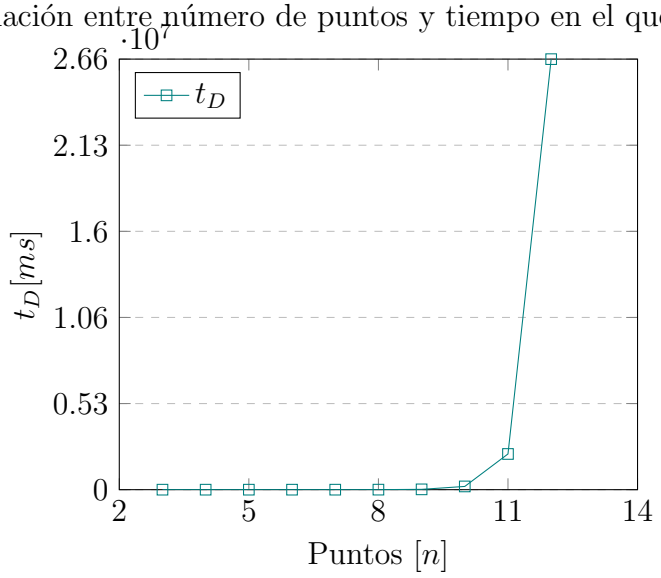
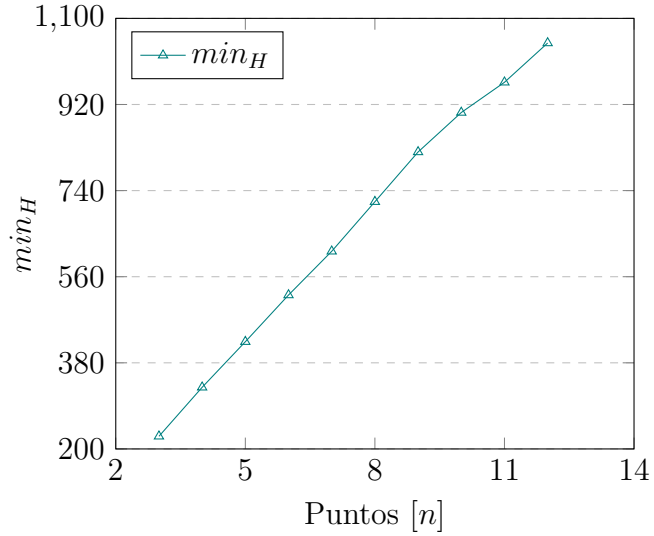


Figura 11: Gráfica de puntos contra distancia para n pequeñas.
Relación entre número de puntos y distancia mínima.



Por último en la Tabla 4 encontramos una sucesión de promedios en los que podemos notar el crecimiento lineal en el valor mínimo y uno caudrático en el tiempo, lo cual es congruente con el orden de complejidad de ambos procedimientos. Estos comportamientos los podemos observar mejor en las Figuras 6, 7, 8 y 9.

9. Conclusiones

Los resultados son muy buenos, el número de operaciones en la solución determinística son del orden de $\theta(n!)$, mientras que en la solución heurística el orden es de $\theta(n^2)$.

El error porcentual es también bajo, tiene un promedio cercano al 5 % y la varianza ronda los 1.5 % por lo que ofrecemos una solución bastante rápida y cercana a los valores óptimos.

Referencias

[1] J.A. Pérez Arcos, *"Encajes primitivos de gráficas planares exteriores"*, proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2014.

[2] C. D. Alfaro Quintero, *"Encajes primitivos de árboles planos"*, proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2013.

[3] J. J. Santana González, *"Diseño de reglas de Golomb óptimas"*, proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2014.

[4] Fedor V. Fomin, Daniel Lokshtanov and Saket Saurabh, *"An Exact Algorithm for Minimum Distortion Embedding"*, Theoretical Computer Science, vol. 412, no. 29, pp 3530-3536, 2011.

[5] Piotr Indyk, *"Algorithmic Applications of Low-distortion Geometric Embeddings"*, FOCS IEEE, p. 40, 2001.

[6] Ríos, R. González, J. *"Investigación de operaciones en acción: Heurísticas para la solución del Problema del Agente Viajero"*. Ingenierías, vol. 3, p. 9, 2000.

[7] Audet, Charles, *"Pooling problem: Alternate formulations and solution methods"*. Management science, vol. 50, no 6, p. 761-776, 2004

[8] Huchete, Joey; Dey, Santanu S.; Vielma, Juan Pablo. *"Beating the SDP bound for the floor layout problem"*, arXiv preprint arXiv:1602.07802, 2016.

[9] D. Salomon, *"David. A guide to data compression methods"*, Springer Science & Business Media, 2013.

[10] Fowler, E. James, R. Yagel, *"Lossless compression of volume data"*, ACM, p. 43-50.

10. Apéndices

Apéndice A: Pruebas de desempeño

Tabla 5: Pruebas de comportamiento para 3 puntos.

min	t_D	min_H	t_H	ϵ	$\epsilon_{\%}$
283	10	283	8	0	0
231	13	231	8	0	0
251	12	251	8	0	0
211	12	211	8	0	0
246	13	246	7	0	0
164	11	164	7	0	0
226	11	226	8	0	0
209	15	209	11	0	0
230	12	230	7	0	0
213	12	213	7	0	0

† .

Tabla 6: Pruebas de comportamiento para 4 puntos.

min	t_D	min_H	t_H	ϵ	$\epsilon_{\%}$
281	12	281	8	0	0
321	13	354	8	33	9.32203
343	13	350	9	7	2
324	14	340	10	16	4.70588
306	15	331	9	25	7.55287
342	15	342	8	0	0
306	12	306	7	0	0
385	13	399	8	14	3.50877
342	13	370	8	28	7.56757
338	16	361	8	23	6.37119

† .

Tabla 7: Pruebas de comportamiento para 5 puntos.

min	t_D	min_H	t_H	ϵ	$\epsilon_{\%}$
456	18	490	10	34	6.93878
431	18	481	10	50	10.395
434	22	434	8	0	0
483	18	512	9	29	5.66406
401	19	451	9	50	11.0865
436	21	462	9	26	5.62771
410	18	410	9	0	0
369	21	406	9	37	9.1133
401	20	469	9	68	14.4989
420	19	504	10	84	16.6667

† .

Tabla 8: Pruebas de comportamiento para 6 puntos.

min	t_D	min_H	t_H	ϵ	$\epsilon_{\%}$
513	54	513	10	0	0
505	52	506	9	1	0.197628
471	67	471	10	0	0
556	50	625	10	69	11.04
623	58	655	10	32	4.8855
477	51	526	10	49	9.31559
477	54	538	10	61	11.3383
522	59	534	10	12	2.24719
482	52	487	12	5	1.02669
591	56	594	11	3	0.50505

† .

Tabla 9: Pruebas de comportamiento para 7 puntos.

min	t_D	min_H	t_H	ϵ	$\epsilon_{\%}$
637	295	637	11	0	0
608	293	636	10	28	4.40252
557	292	594	12	37	6.22896
657	290	671	10	14	2.08644
620	293	651	10	31	4.7619
635	296	658	11	23	3.49544
584	331	639	11	55	8.6072
618	289	618	11	0	0
618	291	618	11	0	0
597	293	641	10	44	6.86427

† .

Tabla 10: Pruebas de comportamiento para 8 puntos.

min	t_D	min_H	t_H	ϵ	$\epsilon_{\%}$
748	2246	777	11	29	3.7323
742	2265	822	13	80	9.73236
764	2264	766	12	2	0.261097
764	2247	766	12	2	0.261097
668	2256	678	13	10	1.47493
752	2264	786	12	34	4.3257
669	2256	713	12	44	6.17111
639	2253	678	12	39	5.75221
701	2244	728	13	27	3.70879
720	2247	720	12	0	0

† .

Tabla 11: Pruebas de comportamiento para 9 puntos.

min	t_D	min_H	t_H	ϵ	$\epsilon_{\%}$
842	20036	880	16	38	4.31818
779	20001	836	24	57	6.81818
835	20078	894	15	59	6.59955
893	20156	941	14	48	5.10096
852	20098	901	15	49	5.4384
751	20142	814	17	63	7.73956
804	20104	844	15	40	4.73934
779	20075	803	16	24	2.98879
840	20099	909	16	69	7.59076
832	20069	873	15	41	4.69645

† .

Tabla 12: Pruebas de comportamiento para 10 puntos.

min	t_D	min_H	t_H	ϵ	$\epsilon_{\%}$
836	200587	917	16	81	8.83315
980	202213	1013	15	33	3.25765
825	200480	825	16	0	0
950	200255	1032	16	82	7.94574
830	199146	887	16	57	6.42616
898	199156	982	15	84	8.55397
943	199626	961	15	18	1.87305
874	200226	893	16	19	2.12766
1052	200852	1071	16	19	1.77404
844	200548	863	16	19	2.20162

† .

Tabla 13: Pruebas de comportamiento para 11 puntos.

min	t_D	min_H	t_H	ϵ	$\epsilon_{\%}$
961	2257244	974	17	13	1.3347
1004	2206139	1083	17	79	7.29455
861	2192925	909	16	48	5.28053
985	2190543	1020	17	35	3.43137
927	2205134	957	17	30	3.1348
962	2201447	1048	19	86	8.20611
920	2205176	1016	16	96	9.44882
1004	2204130	1067	18	63	5.9044
1048	2191204	1088	16	40	3.67647
992	2203538	1045	21	53	5.07177

† .

Tabla 14: Pruebas de comportamiento para 12 puntos.

min	t_D	min_H	t_H	ϵ	$\epsilon_{\%}$
1112	26382161	1143	27	31	2.71216
1043	27490306	1099	38	56	5.09554
1020	26463627	1053	27	33	3.1339
1096	26464607	1138	30	42	3.69069
1043	26451406	1109	31	66	5.95131
1035	26393669	1107	37	72	6.50407
1011	26582212	1094	32	83	7.58684
1084	26499446	1168	32	84	7.19178
990	26713292	1082	25	92	8.50277
1051	26539186	1097	26	46	4.19325

† .

Apéndice B: Pruebas heurísticas

Tabla 15: Pruebas de comportamiento para $n > 12$.

n	min	t_D
15_1	1318	58
15_2	1295	35
15_3	1408	35
20_1	1841	44
20_2	1979	44
20_3	1863	45
25_1	2323	62
25_2	2248	58
25_3	2300	59
30_1	2697	64
30_2	2785	63
30_3	2572	65
35_1	3009	83
35_2	2984	102
35_3	3046	80
40_1	3346	99
40_2	3703	99
40_3	3535	103
45_1	3859	117
45_2	3838	116
45_3	3679	119
50_1	3951	139
50_2	4145	135
50_3	4104	137

† .

Tabla 16: Pruebas de comportamiento para $n < 1000$.

n	min	t_D
100_1	7745	436
100_2	7857	428
100_3	7677	446
200_1	14126	1678
200_2	13927	1722
200_3	14556	1664
300_1	20359	3638
300_2	20446	3740
300_3	20503	3776
400_1	26341	6276
400_2	26648	6265
400_3	26083	6237
500_1	32132	9885
500_2	32450	9757
500_3	31607	9637
600_1	37762	14557
600_2	37891	14813
600_3	37545	14573
700_1	42720	19778
700_2	42509	20734
700_3	43354	19587
800_1	48458	26526
800_2	48356	26544
800_3	48041	26763
900_1	52480	34132
900_2	53333	34528
900_3	52909	34811
1000_1	58408	44055
1000_2	58600	44531
1000_3	58937	44152

† .

11. Entregables comprometidos en la propuesta

Clase Punto

```
#include "bibliotecas.h"
class Punto{
        int coord[10];
        unsigned short dimension;
public:
        Punto();
        Punto(unsigned short,list<unsigned int>);
        Punto(const Punto &old);
        ~Punto();
        unsigned int getCoord(unsigned int);
        unsigned short getDim();
        unsigned int distanciaEuclideana(Punto*);
        unsigned int distanciaManhattan(Punto*);
};
```

```
#include "Punto.h"
Punto::Punto(){
        dimension=0;
        for(int i=0;i<10;i++){
                coord[i]=0;
        }
}
Punto::Punto(unsigned short dimension,
  list<unsigned int> lista){
        this->dimension=dimension;
        list<unsigned int>::iterator it=lista.begin();
        for(int i=0;i<dimension;it++,i++){
                coord[i]=*it;
        }
}
Punto::Punto(const Punto &old){
        dimension= old.dimension;
        for(int i=0;i<dimension;i++){
                coord[i] = old.coord[i];
        }
}
Punto::~Punto(){}
unsigned int Punto::getCoord(unsigned int index){
        return coord[index];
}
```

```
unsigned short Punto::getDim(){
        return dimension;
}
unsigned int Punto::distanciaEuclideana(Punto *otro){
        float resultado=0;
        for(int i=0;i<dimension;i++){
                resultado+=pow((double)getCoord(i)-
                  (double)(otro)->getCoord(i),2);
        }
        resultado=sqrt(resultado);
        return ceil(resultado);
}
unsigned int Punto::distanciaManhattan(Punto *otro){
        unsigned int resultado=0;
        for(int i=0;i<dimension;i++){
                resultado+=std::abs((int)getCoord(i)-
                  (int)otro->getCoord(i));
        }
        return resultado;
}
```

Generadores

```
#include "bibliotecas.h"
using namespace std;
int** floydWarshall(int n,int** A);
std::list<unsigned int> generaPuntos(unsigned int i,
  unsigned short d,unsigned short l);
int** genEuclideano(unsigned short n,unsigned short d,
  unsigned short l);
int** genManhattan(unsigned short n,unsigned short d,
  unsigned short l);
int** genGrafico(unsigned short n,unsigned short l);
```

```
#include "genera.h"
#include "Punto.h"
int** floydWarshall(int n,int** A){
  for(int k=0;k<n;k++){
    for(int i=0;i<n;i++){
      for(int j=0;j<n;j++){
        A[i][j]= std::min(A[i][j],A[i][k] + A[k][j]);
      }
    }
```

```
  }
  return A;
}
std::list<unsigned int> generaPuntos(unsigned int i,
  unsigned short d,unsigned short l){
  int x;
        std::list<unsigned int> lista;
        //std::cout<<"punto "<<i<<": ";
        for(int j=0;j<d;j++){
                //x=distribution(generator);
                x=rand()%(l-1)+1;
                lista.push_back(x);
                //std::cout<<" "<<x;
        }
  return lista;
}
int** genEuclideano(unsigned short n,unsigned short d,
  unsigned short l){
        Punto punto[n];
        for (unsigned int i=0;i<n;i++){
                Punto punkt(d,generaPuntos(i,d,l));
                //std::cout<<endl;
                punto[i]=punkt;
        }
        int** A = new int*[n];
        for (int i=0;i<n;i++){
                A[i]=new int[n+2];
                for(int j=0;j<n;j++){
                        A[i][j]=punto[i].
                          distanciaEuclideana(&punto[j]);
                }
        }
        return A;
}
int** genManhattan(unsigned short n,unsigned short d,
  unsigned short l){
        Punto punto[n];
        for (unsigned int i=0;i<n;i++){
                Punto punkt(d,generaPuntos(i,d,l));
                cout<<endl;
                punto[i]=punkt;
  }
        int** A = new int*[n];
        for (int i=0;i<n;i++){
```

```
                A[i]=new int[n+2];
                for(int j=0;j<n;j++){
                        A[i][j]=punto[i].
                          distanciaManhattan(&punto[j]);
                }
        }
        return A;
}
int** genGrafico(unsigned short n,unsigned short l){
        int** A = new int*[n];
        for (int i=0;i<n;i++){
                A[i]=new int[n+2];
                for(int j=0;j<n;j++){
                        A[i][j] = rand()%(l-1)+1;
                }
                A[i][i]=0;
        }
        for (int i=0;i<n;i++){
                for(int j=0;j<n;j++){
                        A[i][j] = A[j][i];
                }
                A[i][i]=0;
        }
        A = floydWarshall(n,A);
        return A;
}
```

Funciones de utilidad

```
#include "bibliotecas.h"
using namespace std;

void imprimeMatriz(unsigned short n,int** A);
void imprimeVector(int n,int* S);
void imprime_vector_matriz(int n,int j,int** A);
bool esMinima(int x,int &longitudMinima);
```

```
#include "util.h"
#include "Punto.h"

void imprimeMatriz(unsigned short n,int** A){
        for(int i=0;i<n;i++){
                for(int j=0;j<n;j++){
```

```
                        std::cout<<A[i][j]<<"\t";
                }
                        std::cout<<endl;
        }
}
void imprimeVector(int n,int* S){
        for(int i=0;i<n;i++){
                std::cout<<S[i]<<"\t";
        }
        std::cout<<endl;
}
void imprime_vector_matriz(int n,int j,int** A){
        for(int i=0;i<n;i++){
                std::cout<<A[i][j]<<"\t";
        }
        std::cout<<endl;
}
bool esMinima(int x,int &longitudMinima){
        longitudMinima=min(x,longitudMinima);
        if(longitudMinima!=x){
                return true;
        }
        return false;
}
```

Algoritmo determinístico

```
#include "bibliotecas.h"
#include "util.h"
long long int nf[16]={1,1,2,6,
24,120,720,5040,40320,362880,
3628800,39916800,479001600,
6227020800,87178291200,
1307674368000};
int longitudMinima;
int respuesta[1001];
int contador;
int* genPerm(unsigned short n,long long int T){
  int* S = new int[n];
        int posE;
        int contador=0;
        vector<int> buffer;
        vector<int>::iterator it=buffer.begin();
```

```
        for(int i=0;i<n;i++){
                buffer.push_back(i);
        }
        for(int i=n-1;i>1;i--){
                it=buffer.begin();
                posE=(int)((long long int)T/nf[i]);
                T=T%nf[i];
                S[contador]=buffer[posE];
                contador++;
                buffer.erase(it+posE);
        }
        if(T%2==0){
                S[n-1]=buffer.back();
                buffer.pop_back();
                S[n-2]=buffer.back();
                buffer.pop_back();
        }else{
                S[n-2]=buffer.back();
                buffer.pop_back();
                S[n-1]=buffer.back();
                buffer.pop_back();
        }
  return S;
}
int calcSol(unsigned short n,int* S,int** A){
        int longitud=0;
        for(int i=1;i<n;i++){
                longitud+=A[S[i]][S[i-1]];
        }
        return longitud;
}

int* evalua(unsigned short n,int** A){
        longitudMinima=5000;
        int* S= new int[n+2];
        int buffer;
        for(long long int i=0;i<nf[n];i++){
                S=genPerm(n,i);
                int x=calcSol(n,S, A);
                if(!esMinima(x,longitudMinima)){
                        buffer=i;
                }
        }
        S=genPerm(n,buffer);
```

```
        S[n]=longitudMinima;
        S[n+1]=buffer;
        return S;
}
int* evaluaMinimal(unsigned short n,int**A){
        longitudMinima=5000;
        int* S= new int[n+2];
        for(int i=0;i<n;i++){
                S[i]=i;
        }
        permute(n,n,S,A);
        return respuesta;
}
void evenPermute(int N,int n,int*S,int**A,int suma){
        if(N>n+1){
                suma+=A[S[n]][S[n+1]];
        }
        oddPermute(N, n-1, S, A,suma);
        for(int i=0;i<(n-1);i++){
                swap(S[i], S[n-1]);
                oddPermute(N, n-1, S, A,suma);
        }
}
void oddPermute(int N,int n,int*S,int**A,int suma){
        if(n==1){
                suma+=A[S[n-1]][S[n]]+A[S[n]][S[n+1]];
                if(!esMinima(suma,longitudMinima)){
                        for(int i=0;i<N;i++){
                                respuesta[i]=S[i];
                        }
                respuesta[N]=longitudMinima;
                respuesta[N+1]=contador;
                }
                contador++;
        }else{
                if(N>n+1){
                        suma+=A[S[n]][S[n+1]];
                }
                evenPermute(N,n-1,S, A,suma);
                for(int i=0;i<(n-1);i++){
                        swap(S[0], S[n-1]);
                        evenPermute(N,n-1,S, A,suma);
                }
        }
```

```
}
void permute(int N,int n,int * S,int ** A){
        int suma=0;
        contador=0;
        longitudMinima=5000;
        int x=calcSol(n,S, A);
        if(n%2!=0){
                oddPermute(N,n,S,A,suma);
        }else{
                evenPermute(N,n,S,A,suma);
        }
}
```

Algoritmo heurístico

```
#include "bibliotecas.h"
bool apartado[1001];
bool ocupado[1001];
int** copia(int n,int** A){
        int**nueva = new int*[n];
        for(int i=0;i<n;i++){
                nueva[i]=new int[n+2];
                for(int j=0;j<n;j++){
                        nueva[i][j]=300-A[i][j];
                }
                nueva[i][i]=0;
        }
        return nueva;
}
void suma(int n,int** A){
        for(int i=0;i<n;i++){
                A[i][n]=0;
                for(int j=0;j<n;j++){
                        A[i][n] += A[i][j];
                }
        }
}

pair<int,int> compara_primera(int n,int**A){
        int minima_i,minima_j;
        minima_i=0;
        minima_j=0;
        for(int i=1;i<n;i++){
```

```
                if(A[i][n] > A[minima_i][n]){
                        minima_i=i;
                }
        }
        for(int j=1;j<n;j++){
                if(A[minima_i][j] >
                  A[minima_i][minima_j]){
                        minima_j=j;
                }
        }

        pair<int,int> par;
        par.first=minima_i;
        par.second=minima_j;
        apartado[minima_i]=true;
        apartado[minima_j]=true;
        return par;
}
void aparta_primera(int n,int** A,
  pair<int,int> indice_min){
        A[indice_min.first][n]+=300;
        A[indice_min.second][n]+=300;
        A[indice_min.first][indice_min.second]=0;
        A[indice_min.second][indice_min.first]=0;
}
pair<int,int> compara(int n,pair<int,int>
  indice_min,int**A){
        int minima_i,minima_j;
        minima_i=0;
        if(A[indice_min.first][n] >
          A[indice_min.second][n]){
                minima_i=indice_min.first;
        }else{
                minima_i=indice_min.second;
        }
        minima_j=0;
        for(int j=1;j<n;j++){
                if(A[minima_i][j] >= A[minima_i][minima_j]
                  && !ocupado[j]){
                        minima_j=j;
                }
        }

        if(!apartado[minima_j]){
```

```
                apartado[minima_j]=true;
        }
        ocupado[minima_i]=true;
        pair<int,int> par;
        par.first=minima_i;
        par.second=minima_j;
        return par;
}

void aparta(int n,int** A,pair<int,int> indice_min,
  pair<int,int> anterior){
        for(int i=0;i<n;i++){
                A[i][n]+=300;
        }

        A[anterior.second][n]-=A[anterior.first]
          [anterior.second];

        for(int i=0;i<n;i++){
                A[indice_min.first][i]=0;
                A[i][indice_min.first]=0;
        }
        A[indice_min.first][n]=0;
        anterior.first=indice_min.second;
}

int* gloton(int n,int** A){
        int* S=new int[n+2];
        int** nueva;


        pair<int,int> indice_min,actual_min;
        deque<int> orden;
        nueva=copia(n,A);
        suma(n,nueva);

        indice_min = compara_primera(n,nueva);
        aparta_primera(n,nueva,indice_min);
        orden.push_back(indice_min.first);
        orden.push_back(indice_min.second);

        for(int i=2;i<n;i++){

                nueva[orden.front()][orden.back()]=0;
```

```
                nueva[orden.back()][orden.front()]=0;
                actual_min = compara(n,indice_min,nueva);
                //actualiza minimo


                if(orden.front() == actual_min.first){
                    orden.push_front(actual_min.second);
                }else{
                    orden.push_back(actual_min.second);
                }
                indice_min.first=orden.front();
                indice_min.second=orden.back();

                aparta(n,nueva,actual_min,indice_min);
        }
        deque<int>::iterator it=orden.begin();
        for(int i=0;i<n, it!=orden.end();i++,it++){
                S[i]=(*it);
        }
        S[n]=calcSol(n,S,A);

        return S;
}
```

Bibliotecas en general

```
#include <cmath>
#include <list>
#include <iostream>
#include <cstdlib>
#include <vector>
#include <ctime>
#include <deque>
using namespace std;

int calcSol(unsigned short n,int* S,int** A);
int* evalua(unsigned short n,int** A);
int* evaluaMinimal(unsigned short n,int**A);
void evenPermute(int N,int n,int*S,int**A,int suma);
void oddPermute(int N,int n,int*S,int**A,int suma);
void permute(int N,int n,int * S,int ** A);
int reEvalua(int N,int i,int j,int** A,int* S,int suma);
```

```
int** copia(int n,int** A);
void suma(int n,int** A);
pair<int,int> compara_primera(int n,int**A);
void aparta_primera(int n,int** A,
  pair<int,int> indice_min);
pair<int,int> compara(int n,
  pair<int,int> indice_min,int**A);
void aparta(int n,int** A,pair<int,int>
  indice_min,pair<int,int> anterior);
int* gloton(int n,int** A);
```

Función main

```
#include "Punto.h"
#include "bibliotecas.h"
#include "genera.h"
#include "util.h"

void prueba_det(){
        unsigned short n,d,l,switcher;
        //cin>>n>>d>>l>>switcher;
        d=10;l=100;switcher=2;
        cin>>n;
        int** A;
        switch(switcher){
        case 1:
                A = genEuclideano(n,d,l);
                imprimeMatriz(n,A);
                break;
        case 2:
                A = genManhattan(n,d,l);
                imprimeMatriz(n,A);
                break;
        case 3:
                A = genGrafico(n,l);
                imprimeMatriz(n,A);
                break;
        default:
                break;
        }
        int* S=new int[n+2];
        S=evaluaMinimal(n,A);
```

```
        cout<<"Deterministicamente:"<<endl;
        cout<<"La menor incrustacion en la recta es de
          longitud "<<S[n]<<endl;
        cout<<"Y sigue la secuencia: "<<endl;
        imprimeVector(n,S);
        cout<<"permutacion numero "<<S[n+1]<<endl;
}

void prueba_heu(){
        unsigned short n,d,l,switcher;
        d=10;l=100;
        cin>>n;
        int** A;

        int* S=new int[n+2];
        A = genEuclideano(n,d,l);

        S = gloton(n,A);

        cout<<"Heuristicamente: "<<endl;
        cout<<"La menor incrustacion en la recta es de
          longitud "<<S[n]<<endl;
        cout<<"Y sigue la secuencia: "<<endl;
        imprimeVector(n,S);
}

void det_contra_heu(int n){
        time_t inicia_d,termina_d,tiempo_d,inicia_h,
          termina_h,tiempo_h;
        unsigned short d,l;
        //cin>>n>>d>>l;
        d=10;l=100;
        int** A;
        int** nueva;
        int* S=new int[n+2];
        A = genEuclideano(n,d,l);

        int long_det,long_heu;
        float dif_porc;
        if(n<13){
                inicia_d=clock();
                S=evaluaMinimal(n,A);
                termina_d=clock();
                tiempo_d=termina_d-inicia_d;
```

```
                long_det=S[n];

                cout<<long_det<<"\t"<<tiempo_d<<"\t";

                inicia_h=clock();
                S = gloton(n,A);
                termina_h=clock();
                tiempo_h=termina_h-inicia_h;

                long_heu=S[n];
                dif_porc = 100*((float)(long_heu-long_det))
                  /(float)long_heu;

                cout<<long_heu<<"\t"<<tiempo_h<<"\t";
                cout<<(long_heu-long_det)<<"\t"<< dif_porc;
                cout<<"\t"<<(tiempo_d-tiempo_h);
                cout<<endl;
        }else{

                inicia_h=clock();
                S = gloton(n,A);
                termina_h=clock();
                tiempo_h=termina_h-inicia_h;

                long_heu=S[n];

                cout<<long_heu<<"\t"<<tiempo_h<<endl;
        }


}

int main(){
        srand(time(NULL));
        //prueba_det();//15! tarda 19 hrs.
        //prueba_heu();
        det_contra_heu(12);
        return 0;
}
```