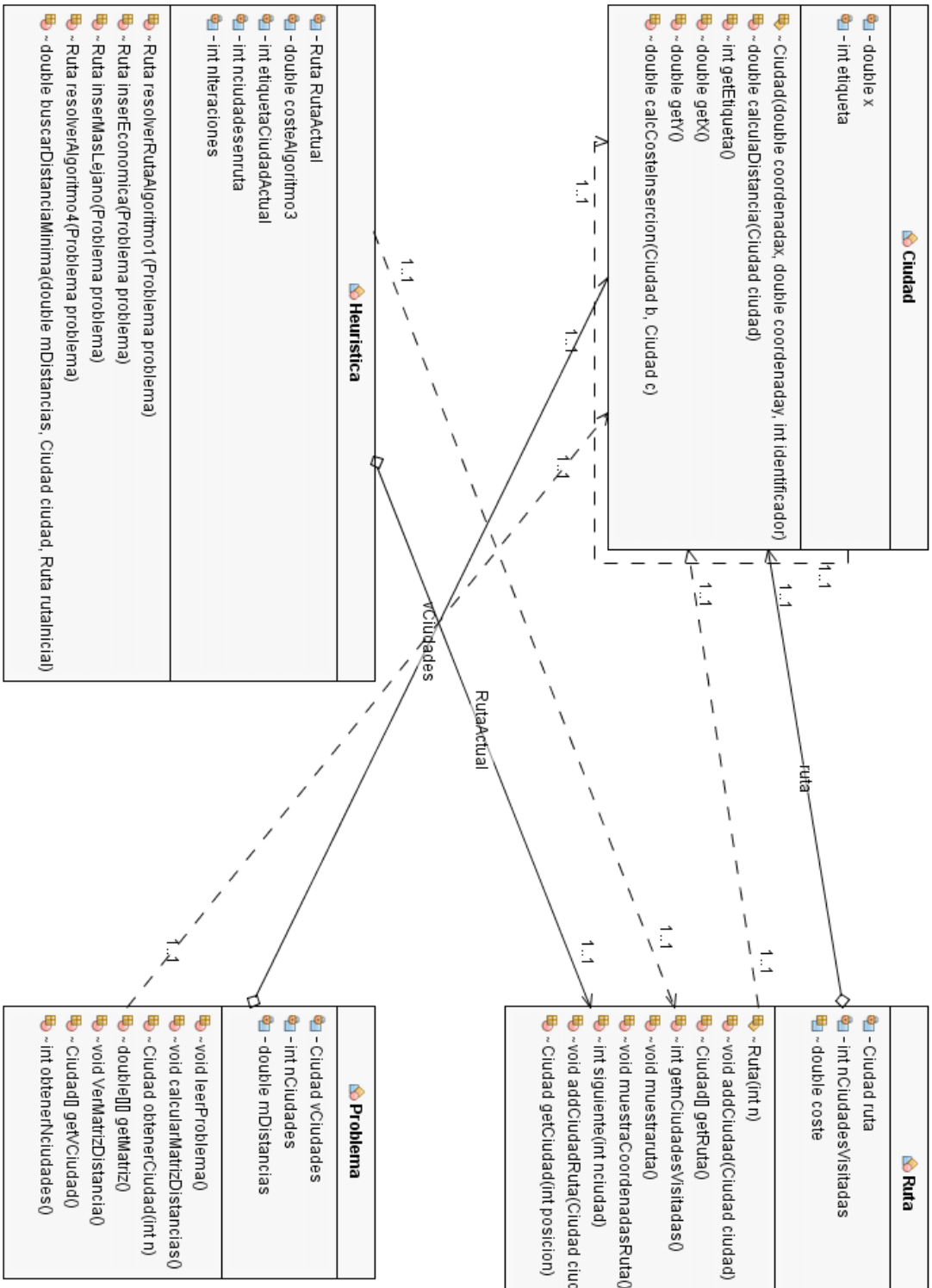
Práctica 4
El problema del viajante de comercio

José Manuel García Giménez
Manuel Moya Ferrer

INDICE

Diagrama UML



Justificación de la solución

Para la resolución del problema que se nos plantea utilizamos el algoritmo del vecino más cercano. Este algoritmo consiste en tomar una ciudad inicial e ir formando la ruta añadiendo la siguiente ciudad que se encuentre más cerca de entre las que no se han visitado. De esta manera se van añadiendo todas las ciudades y por último se añade de nuevo la primera de la ruta para que empiece y acabe por el mismo sitio.

Esta metodología se repite utilizando todas las ciudades como ciudad inicial, dando como resultado tantas rutas como ciudades. Nos quedamos con la ruta que tenga menor coste de entre las calculadas.

Las clases que utilizamos para la implementación de este algoritmo en Java son las que podemos ver en el diagrama UML.

La clase Ciudad. Es la que nos permite almacenar información referente a las ciudades por las que queremos pasar, en nuestro caso tenía como atributos las coordenadas de la ciudad y una etiqueta. La etiqueta sirve para identificarla unívocamente.

La clase Ruta. Es la que nos permite representar la solución del problema mostrando las ciudades por las que pasa y el coste de la ruta. Para ello tiene como atributos: un vector de ciudades, que es donde se almacenan las ciudades por donde se pasa; el número de ciudades que lleva visitadas, como que se permita dar una solución parcial del problema, es necesario llevar una cuenta de las ciudades que se van visitando; y por último el coste, que es un atributo de tipo double que simplemente guarda la distancia total de la ruta.

La clase Heurística. Es la clase que contiene los algoritmos que resuelven el problema. Los algoritmos utilizados se incluyen como métodos de la clase. Los atributos ayudan a la implementación de los distintos algoritmos.

La clase Problema. Esta clase sirve para leer y almacenar los distintos datos del problema, principalmente la lista de ciudades y realizar algunas primeras operaciones con los datos. Como calcular la distancia de todas las ciudades y almacenarlas en una matriz.

Como podemos observar en el diagrama UML las dependencias son bastante sencillas ya que todas las clases están al mismo nivel. Las relaciones son de agregación (flecha línea continua y rombo en el origen en el diagrama), el objeto base utiliza al incluido para su funcionamiento. O de dependencia (línea discontinua en el diagrama), el elemento origen necesita del elemento destino para su implementación.

Mejoras implementadas

Hemos implementado, además del algoritmo del vecino más cercano, otros de los algoritmos que se nos proponían en el guión. Concretamente, hemos implementado 3 algoritmos adicionales:

- **Inserción más económica**. Este algoritmo consiste en, partiendo de una base de una ruta formada por 3 ciudades, ir añadiendo ciudades en una posición arbitraria, de forma que al insertar una ciudad el aumento del coste de la ruta sea mínimo. Para ello se prueba con todas las ciudades sin visitar en todas las posiciones posibles para la inserción quedándose finalmente con la que suponga menos incremento en coste. Este algoritmo se implementa como otro método de la clase heurística. Para ello añadimos algunos métodos en las clases iniciales, como calcCosteInsercion(), que lo que hace es calcular el incremento en cose que produce insertar una ciudad en una posición determinada.

- **Inserción del más lejano.** Este algoritmo se comienza igual que el anterior, con una ruta inicial de 3 ciudades. Pero en este caso en vez de insertar la ciudad que suponga un menor incremento en coste, se inserta la ciudad más lejana a todas las que ya se encuentren en la ruta en la posición que menos aumente el coste. De esta manera lo que se consigue es que el coste de la ruta quede aproximadamente definido con las primeras iteraciones del algoritmo. Para este algoritmo utilizamos los métodos añadidos para el anterior y añadimos alguno adicional. El método buscaDistnciaMinima(), que lo que hace es buscar la distancia mínima entre una ciudad dada y cualquiera de las que están en la ruta. De esta manera, se busca la ciudad cuya distancia mínima sea mayor y se inserta en la posición que aumente menos el coste.

- **Algoritmo 4.** En este algoritmo, se calcularán K rutas aleatorias del conjunto de ciudades dado, poniendo K=10000. De estas rutas, nos quedaremos con aquella que nos genere un menor coste.

Tabla

		Algoritmo1			Algoritmo2			Algoritmo3			Algoritmo4			
	Òptimo	**min**	**%opt**	**mili**	**min**	**%opt**	**mili**	**min**	**%opt**	**mili**	**min**	**%opt**	**mili**	
a280	2579	3425,42	32,81969756		81	3139,15	21,71965878	9	5127,66	98,82357503	24	33153,04	1185,499806	279
berlin52	7542	10012,23	32,75298329		4	8324,43	10,3743039	2	18097,08	139,9506762	7	31924	323,2829488	54
eil101	629	785,55	24,88871224		14	693,62	10,27344992	2	1126,5	79,09379968	9	3440,62	446,9984102	99
KROA200	29368	37941,67	29,19391855		36	31805,4	8,29950967	6	80331,19	173,5330632	14	336159,74	1044,64635	198
lin318	42029	51340,27	22,1543934		110	48011,35	14,23386233	11	69288,85	64,85962074	30	596752	1319,857717	327
pr1002	259045	317836,8	22,69559343		2799	316185,22	22,05802853	28	532671,28	105,6288599	607			
small10	3256,13	3456,38	6,149938731		0	2826,5	-13,19449776	0	5276,26	62,04082761	0	6190,33	90,11310973	23
Promedio	49206,87571	10012,23	24,88871224		36	8324,43	10,3743039	6	18097,08	98,82357503	14	32538,52	745,82238	148,5