

**Facultat d'Informàtica de Barcelona (FIB)
Universitat Politècnica de Catalunya (UPC)
BarcelonaTech**

Programación genética en mercados financieros

Construcción automática de reglas de inversión utilizando
programación genética

**INGENIERÍA EN INFORMÁTICA
PROYECTO FINAL DE CARRERA**

16/01/2012

AUTOR: Mario A. Llorente Lopez

**DIRECTOR: Argimiro A. Arratia Quesada
DEPARTAMENTO: Lenguajes y Sistemas Informáticos**

**PONENTE: Marta Arias Vicente
DEPARTAMENTO: Lenguajes y Sistemas Informáticos**

Contenido

CONTENIDO	1
1. INTRODUCCIÓN	4
1.1 DESCRIPCIÓN DEL PROYECTO	4
1.2 ANTECEDENTES	4
1.3 OBJETIVOS.....	5
1.4 PLANIFICACIÓN.....	5
1.5 ESTUDIO ECONÓMICO Y HERRAMIENTAS UTILIZADAS	8
1.5.1 <i>Recursos humanos</i>	8
1.5.2 <i>Material</i>	9
1.5.3 <i>Gastos generales</i>	10
1.5.4 <i>Coste total</i>	10
1.6 ORGANIZACIÓN DE LA MEMORIA	11
PARTE 1: PROGRAMACIÓN GENÉTICA Y MERCADOS FINANCIEROS	12
2. ALGORITMOS GENÉTICOS.....	12
2.1 CARACTERÍSTICAS GENERALES	12
2.2 PROGRAMACIÓN GENÉTICA.....	16
2.2.1 <i>Codificación</i>	16
2.2.2 <i>Operadores</i>	17
2.2.3 <i>Función de fitness</i>	20
2.2.4 <i>Parámetros por defecto</i>	21
3. MERCADOS FINANCIEROS	22
3.1 CARACTERÍSTICAS GENERALES	22
3.1.1 <i>Mercado de acciones</i>	23
3.2 ESTRATEGIAS DE INVERSIÓN	24
3.3 CRITERIOS DE INVERSIÓN	26
3.3.1 <i>Análisis fundamental</i>	26
3.3.2 <i>Análisis técnico</i>	28
3.3.3 <i>Hipótesis de la eficiencia del mercado</i>	32

4. DISEÑO DEL ALGORITMO.....	34
4.1 ADAPTACIÓN DE LA PROGRAMACIÓN GENÉTICA PARA ENCONTRAR REGLAS DE INVERSIÓN	34
4.2 DETALLES DE LA IMPLEMENTACIÓN.....	37
4.2.1 <i>Tipos de nodos usados</i>	37
4.2.2 <i>Representación e inicialización de los individuos</i>	39
4.2.3 <i>Evaluación</i>	43
4.2.4 <i>Operadores</i>	44
4.3 SOBREENTRENAMIENTO	48
PARTE 2: CONSTRUCCIÓN DE UNA APLICACIÓN PARA EXPERIMENTACIÓN	50
5. ANÁLISIS Y ESPECIFICACIÓN	50
5.1 METODOLOGÍA DE TRABAJO.....	50
5.2 REQUISITOS DEL SISTEMA.....	51
5.2.1 <i>Funcionales</i>	51
5.2.2 <i>No funcionales</i>	53
5.3 MODELO DE CASOS DE USO.....	53
5.3.1 <i>Actores</i>	54
5.3.2 <i>Diagramas</i>	54
5.3.3 <i>Descripción</i>	56
5.4 MODELO CONCEPTUAL.....	84
5.4.1 <i>Diagrama de clases</i>	84
6. DISEÑO	91
6.1 ARQUITECTURA DEL SISTEMA.....	91
6.1.1 <i>Capa de presentación</i>	92
6.1.2 <i>Capa de dominio</i>	97
6.1.3 <i>Capa de gestión de datos</i>	100
7. IMPLEMENTACIÓN	106
7.1 LENGUAJES UTILIZADOS	106
7.1.1 <i>C++</i>	106
7.1.2 <i>El framework Qt</i>	107

7.1.3	<i>Structured Query Languaje (SQL)</i>	108
7.2	SISTEMA GESTOR DE LA BASE DE DATOS	109
7.3	ENTORNO DE DESARROLLO.....	110
7.3.1	<i>Librerías externas</i>	110
7.3.2	<i>NetBeans</i>	112
7.3.3	<i>MySQL Workbench</i>	115
	PARTE 3: EXPERIMENTOS.....	120
	8. EVALUACIÓN DE LAS REGLAS DE INVERSIÓN	120
8.1	ENTORNO DE LA EXPERIMENTACIÓN.....	121
8.2	ANÁLISIS DE BENEFICIOS	123
8.3	ANÁLISIS DE LA EFECTIVIDAD SEGÚN LA TENDENCIA	132
8.4	RESUMEN DE CONCLUSIONES.....	134
	CONCLUSIÓN	136
	9. CONCLUSIONES	136
9.1	OBJETIVOS CONSEGUIDOS.....	136
9.2	AMPLIACIONES FUTURAS	139
9.3	VALORACIÓN PERSONAL	139
	10. REFERENCIAS.....	141
10.1	BIBLIOGRAFÍA.....	141
10.2	SITIOS WEB.....	142
	APÉNDICE A: INSTALACIÓN DE LIBRERÍAS EXTERNAS.....	143
	APÉNDICE B: EJEMPLO DE USO DEL PROGRAMA.....	144
	APÉNDICE C: DATOS DE LOS EXPERIMENTOS	160

1. INTRODUCCIÓN

1.1 Descripción del proyecto

El proyecto consiste en implementar un algoritmo de programación genética que permita descubrir reglas de inversión que indiquen cuándo entrar y salir de un mercado para obtener los máximos beneficios. Estas reglas estarán construidas con algunos de los indicadores más utilizados por el análisis técnico.

Adicionalmente se pretende construir un software de tamaño mediano que integre el algoritmo descrito y que permita hacer experimentos con el fin de optimizar los datos de entrada del algoritmo para obtener los máximos beneficios en el mercado. En concreto, estos beneficios se mostrarán respecto a los conseguidos mediante una estrategia de inversión pasiva, muy común entre los inversores, conocida como *buy and hold*, que consiste en comprar un valor en una fecha dada y venderlo en otra posterior, sin entrar ni salir de él entre los dos momentos.

El proyecto se enfoca dentro del campo de la inteligencia artificial y de la ingeniería del software. La orientación principal es hacia el ámbito de la investigación, aunque con un componente procedural en el diseño de software.

1.2 Antecedentes

La construcción automática de reglas de inversión utilizando programación genética ha sido estudiada con bastante profundidad desde que Franklin Allen y Risto Karjalainen publicaron su artículo titulado *Using Genetic Algorithms to Find Technical Trading Rules* en diciembre de 1993 [1]. En el artículo, además de describir el algoritmo, se hace un estudio estadístico detallado de la efectividad de las reglas generadas, tanto con datos reales (del índice SP 500) como con secuencias de precios generadas aleatoriamente siguiendo unas pautas determinadas. Es importante tener en cuenta que los autores tuvieron en cuenta los costes aplicados a cada transacción efectuada, lo cual es fundamental para determinar los beneficios dados por las reglas en la realidad. Desde ese momento, y especialmente a partir de principios de la década del 2000, numerosos autores han estudiado el tema basándose en el artículo mencionado. Este proyecto también parte del artículo anteriormente mencionado, aunque cuenta con influencias de [2], [3] y [4] así como aportaciones propias.

La mayoría de estudios se centran en encontrar la mejor configuración de parámetros o de realizar pequeñas modificaciones al algoritmo para obtener mayores beneficios que la estrategia *buy and hold*. Sin embargo, las condiciones del mercado tanto en los períodos utilizados para generar reglas como en los utilizados para testearlas no han sido debidamente estudiados. Tampoco lo ha sido la relación entre los costes por transacción utilizados en la generación y en el test. Este proyecto cuenta entre sus objetivos dar algo de luz a estas dos cuestiones.

1.3 Objetivos

Las metas a las que se pretende llegar al final de proyecto son las siguientes:

- Profundizar los conocimientos sobre algoritmos genéticos y en particular de programación genética.
- Aprender las características de los mercados de valores y los diferentes criterios de inversión, en particular el análisis técnico.
- Diseñar e implementar, a partir de otros trabajos de investigación sobre el tema, un algoritmo de programación genética que sirva para encontrar reglas de inversión basadas en el análisis técnico.
- Desarrollar una aplicación de tamaño medio siguiendo todas las etapas de la ingeniería del software: análisis, especificación, diseño e implementación. Esta aplicación ha de permitir utilizar el algoritmo de programación genética desde el punto de vista de un inversor o de un investigador.
- Mejorar la habilidad de programación en el lenguaje C++ y aprender a usar funciones contenidas en librerías dinámicas de terceras partes.
- Utilizar el *framework* Qt para construir una interfaz de usuario con apariencia profesional.
- Realizar un conjunto de experimentos que permitan determinar la relación existente entre los costes por transacción aplicados en la generación de las reglas de inversión con el algoritmo de programación genética y el beneficio obtenido por éstas junto al número de transacciones realizadas en un periodo de test.
- Realizar un conjunto de experimentos que permitan relacionar el beneficio dado por las reglas de inversión obtenidas con el algoritmo de programación genética con la tendencia del mercado en los periodos utilizados para generarlas y testearlas, teniendo en cuenta los costes por transacción.
- Realizar un conjunto de experimentos que permitan confirmar si las reglas de inversión obtenidas con el algoritmo de programación genética generan un beneficio superior a la estrategia *buy and hold*, teniendo en cuenta los costes por transacción.
- Elaborar una documentación que recoja todo lo realizado y que permita en un futuro entender y ampliar el proyecto por diferentes personas.

1.4 Planificación

El proyecto se inicia el 7 de febrero de 2011, con previsión de acabarlo a finales de julio del mismo año. En total se pretenden dedicar unas 24 semanas a jornada completa, lo que equivale a 960 horas. El reparto de tiempo entre las diferentes tareas calculada inicialmente es la siguiente:

Tarea	Tiempo (horas)
Estudio de la documentación de programación genética	100
Estudio del lenguaje C++, preparación del entorno de programación	40
Estudio de mercados financieros, obtención de datos de históricos de precios	40

Reuniones con el director del proyecto	20
Diseño del algoritmo	100
Implementación y testeo del algoritmo	140
Análisis y especificación	100
Diseño	40
Implementación	200
Experimentación	60
Redacción de la documentación	120

A continuación se muestra un diagrama de Gantt con la distribución en el tiempo de cada etapa a excepción de las reuniones con el director del proyecto, las cuales se producen de forma irregular (la frecuencia se comprende entre una vez a la semana y una vez cada tres semanas) y suelen durar unas dos horas. Hay que tener en cuenta que hacia finales de abril habrá una semana en la que no se trabajará por vacaciones:

Id.	Nombre de tarea	Comienzo	Fin	Duración	feb 2011		mar 2011				abr 2011				may 2011				jun 2011				jul 2011					
					6/2	13/2	20/2	27/2	6/3	13/3	20/3	27/3	3/4	10/4	17/4	24/4	1/5	8/5	15/5	22/5	29/5	5/6	12/6	19/6	26/6	3/7	10/7	17/7
1	Estudio de mercados financieros, obtención de históricos de precios	07/02/2011	18/02/2011	2s																								
2	Estudio de la documentación de programación genética	07/02/2011	02/03/2011	3,6s																								
3	Diseño del algoritmo	03/03/2011	18/03/2011	2,4s																								
4	Estudio del lenguaje C++, preparación del entorno de programación	21/03/2011	01/04/2011	2s																								
5	Implementación y testeo del algoritmo	21/03/2011	20/04/2011	4,6s																								
6	Análisis y especificación	02/05/2011	18/05/2011	2,6s																								
7	Diseño	19/05/2011	25/05/2011	1s																								
8	Implementación	26/05/2011	29/06/2011	5s																								
9	Experimentación	30/06/2011	08/07/2011	1,4s																								
10	Redacción de la documentación	11/07/2011	29/07/2011	3s																								

Hacer una planificación lo más ajustada a la realidad es importante, ya que de ello depende el presupuesto mostrado al cliente. Cuanto más aproximado al precio final sea el presupuesto, mayor será la satisfacción del cliente. Esta es una tarea importante en las etapas destinadas a la construcción del sistema software, ya que es llevada a cabo por una empresa privada como un servicio dado a un cliente. Este cálculo corresponde hacerlo al director del proyecto.

1.5 Estudio económico y herramientas utilizadas

En este apartado se realiza un cálculo aproximado del coste que tendría desarrollar el proyecto en la realidad. Esto es el presupuesto mostrado a un cliente si pidiese realizar el proyecto a una empresa.

1.5.1 Recursos humanos

Este proyecto tiene dos componentes claramente separados. Por una parte se encuentra la que comprende la realización del algoritmo de programación genética, y por otra la que respecta el desarrollo de la aplicación mediante ingeniería del software. Normalmente, no sucede que ambas partes se realicen en una misma empresa, a causa de que son necesarios profesionales competentes en diferentes áreas.

Este proyecto es desarrollado por una persona, no obstante, en la realidad serían necesarios diferentes profesionales para llevarlo a cabo. Estos profesionales deberían ser los siguientes:

- **Jefe de proyecto (JP):** Coordina todas las actividades realizadas por los demás miembros del equipo, se encarga de planificarlas y de que el proyecto se lleve a cabo de acuerdo a la planificación realizada. Esta presente en el desarrollo de la aplicación.
- **Analista (A):** Recopila los requisitos con los que ha de contar el sistema y realiza la especificación del sistema en la parte del desarrollo de la aplicación. En la parte de la construcción del algoritmo, se encarga de su diseño.
- **Diseñador (D):** Realiza el diseño del sistema. Esta presente en el desarrollo de la aplicación.
- **Programador (P):** Implementa el sistema de acuerdo a la documentación generada por el diseñador, en la parte del desarrollo de la aplicación, o el analista, en la parte de la construcción del algoritmo.

Teniendo en cuenta las funciones que desempeña cada profesional, se pueden asignar las tareas del proyecto a cada uno de ellos. Para ello se parte de la tabla mostrada en el apartado 1.4, repartiendo cada una de las tareas entre uno o más profesionales y unas horas determinadas a cada uno, excepto a las tareas "Estudio de la documentación de programación genética" y "Estudio del lenguaje C++, preparación del entorno de programación", ya que se supone que en las empresas en que se realiza el proyecto, los trabajadores ya tienen los conocimien-

tos sobre dichas tareas y el entorno de programación configurado. A continuación se muestra el reparto realizado:

Tarea	Tiempo (horas)	Roles
Estudio de mercados financieros, obtención de datos de históricos de precios	40	A
Reuniones con el director del proyecto	20	A (90%), JP (10%)
Diseño del algoritmo	100	A
Implementación y testeo del algoritmo	140	P
Análisis y especificación	100	A (90%), JP (10%)
Diseño	40	D (90%), JP (10%)
Implementación	200	P
Experimentación	60	A
Redacción de la documentación	120	JP (20%), A (50%), D (30%)

Teniendo en cuenta el reparto realizado y los sueldos brutos en el mercado, es decir, los sostenidos por la empresa, los costes del personal resultantes son:

Rol	Coste (€/h)	Horas	Coste (€)
JP	90	40	3600
A	60	368	22080
D	50	72	3600
P	30	340	10200

Los costes calculados son para personal con cierta experiencia, que realiza el trabajo de forma más rápida que cuando el personal carece de ella. En este proyecto no se cuenta con experiencia, con lo cual habría que descontar algún porcentaje a los precios calculados para que éstos sean realistas, que podría ser entorno a un 15%. Con esto, el coste total del personal es:

$$\text{Coste personal} = 0,85 \cdot (3600 + 22080 + 3600 + 10200) = 33558 \text{ €}$$

1.5.2 Material

A continuación se muestra una tabla con el coste de todo el hardware y software utilizado para realizar el proyecto:

Producto	Coste (€)
Ordenador de gama media	700 (aprox.)
Visio Professional 2010	725
Word 2010	189
Netbeans 6.9.1	0
Qt Designer	0
Umbrello	0
MySQL Workbench 5.2.30	0

Ubuntu 10.04 y librerías externas	0
Total	1614

En cualquier empresa, el material es compartido por otros proyectos que realizados en la empresa y por ello debe tenerse en cuenta únicamente el coste de amortización. Esto es la proporción de vida útil de los productos que se utiliza en el proyecto. Si se asume una vida útil de 5 años y el proyecto dura 6 meses (0,5 años), el coste de amortización es el siguiente:

$$\text{Coste amortización} = \frac{0,5}{5} \cdot 1614 = 162 \text{ €}$$

1.5.3 Gastos generales

Aparte de los gastos mostrados anteriormente, también se deben tener en cuenta otros, principalmente la electricidad, internet y el coste del alquiler de una oficina. Los costes de la electricidad e internet se calculan sobre los que necesita una persona, ya que el proyecto no es desarrollado en ningún momento por más de una persona simultáneamente. Estos cálculos son medios y aproximados, ya que varían considerablemente entre los operadores. El coste de la oficina también se calcula como el coste del espacio ocupado por una persona; se toma como precio por metro cuadrado el existente en la zona del ensanche de Barcelona. A continuación se muestra el coste mensual de los gastos comentados:

Concepto	Coste (€/mes)
Electricidad	20
Internet	20
Alquiler (6 m ² por persona, 15 €/m ²)	90
Otros gastos	20
Total	150

Teniendo en cuenta que el proyecto tiene una duración de 6 meses, los costes generales para todo él son:

$$\text{Costes generales} = 150 \cdot 6 = 900 \text{ €}$$

1.5.4 Coste total

Una vez calculados todos los costes separados por concepto, se puede calcular el coste total que tendría el proyecto sumando los costes individuales. Este coste es una aproximación, ya que, como se ha explicado al calcular los costes anteriores, algunos conceptos se indican de forma aproximada:

$$\text{Coste total} = 33558 + 162 + 900 = 34620 \text{ €}$$

1.6 Organización de la memoria

La documentación sigue el orden temporal en el que se ha desarrollado el proyecto. Ésta consta de tres partes claramente indicadas: programación genética y mercados financieros, construcción del sistema software, y experimentación.

La primera de ellas ofrece una visión general, aunque imprescindible para entender el proyecto, sobre los algoritmos genéticos, programación genética y los mercados financieros. También detalla cómo está construido el algoritmo de programación genética implementado en el proyecto.

La segunda parte contiene la documentación necesaria para comprender el sistema software construido en este proyecto siguiendo las etapas de la ingeniería del software.

La última parte presenta los resultados de unos experimentos realizados con las reglas generadas mediante el software con el objetivo de indicar con qué parámetros de entrada el sistema genera las reglas de inversión más efectivas.

PARTE 1: PROGRAMACIÓN GENÉTICA Y MERCADOS FINANCIEROS

2. ALGORITMOS GENÉTICOS

La inteligencia artificial nació con el objetivo de crear programas que encontrasen de forma automatizada soluciones próximas a la mejor solución existente para un problema dado, cuyo espacio de búsqueda resulta inabordable para realizar una búsqueda exhaustiva, esto es comprobar una a una todas las soluciones posibles. Hay diversas familias de algoritmos en inteligencia artificial, siendo las más importantes las siguientes: redes neuronales, algoritmos genéticos, sistemas de lógica difusa y autómatas programables. La elección de un algoritmo perteneciente a una determinada familia depende del tipo de problema a resolver.

2.1 Características generales

Los algoritmos genéticos, también llamados algoritmos evolutivos, están inspirados en los principios que rigen la evolución de los seres vivos en la naturaleza que aparecen en el libro *Origen de las especies* de Charles Darwin (1859). Aunque en la década de 1960 surgieron trabajos sobre algoritmos que simulaban estrategias evolutivas, no fue hasta 1975 que John H. Holland definió y concretó en su libro *Adaptation in Natural and Artificial Systems* (1975) las bases utilizadas hoy en día para los algoritmos genéticos. [1]

Estos algoritmos se fundamentan en que los individuos mejor adaptados al entorno en el que viven son los que tendrán más probabilidades de tener descendencia y, consecuentemente, que sus características se combinen con las de otros individuos. Es probable que un individuo bien adaptado al entorno escoja a otro individuo que también esté muy adaptado, ya que ambos poseen características que les hacen sobresalir del resto. La combinación del material genético de los dos producirá una descendencia que tendrá características de ambos individuos, que, combinadas en distinto grado, posiblemente mejoren la adaptación de la descendencia al entorno aún más. Como ejemplo, si una gacela que acierta a detectar depredadores con mucha antelación, tiene descendencia con otra que destaca por su rapidez, su descendencia combinará en cierto grado estas dos características y sus posibilidades de sobrevivir ante el ataque de un depredador posiblemente aumentarán respecto a las de sus progenitores. Es también más probable que sobrevivan más tiempo que la media de la población, al poseer ambas características que lo facilitan, con lo cual es probable que tengan mayor número de descendencia que la media. De esta forma, siendo los mejor adaptados los que tienen más descendencia, se aumenta la calidad de la población con el paso de las generaciones.

Este tipo de algoritmos no son los únicos inspirados en el comportamiento observado en la naturaleza, en este caso en la evolución de los seres vivos. Las redes neuronales, surgidas con posterioridad, se basan en el funcionamiento del sistema nervioso de los seres vivos, de tal forma que posibilitan, entre otras cosas, aprender autónomamente y tolerar imprecisiones en los datos de entrada sin afectar a la respuesta dada. Es acertado basarse en la naturaleza para diseñar algoritmos de inteligencia artificial, fundamentalmente por dos razones. En primer

lugar, la inteligencia proviene de ella y se observa constantemente en los seres vivos. En segundo lugar, la naturaleza ha estado presente en el planeta prácticamente desde su creación, con lo cual los métodos que emplea están perfeccionados de forma extraordinaria.

En cierto modo, la evolución de los seres vivos puede ser vista como una búsqueda de la mejor combinación de características que hagan a un individuo adaptarse lo mejor posible a un determinado entorno. Esto es precisamente lo que hacen los algoritmos genéticos. Un algoritmo genético empieza con una población de individuos, en la que cada uno representa a una solución válida para el problema a resolver. Cada una de estas soluciones (o individuos) son normalmente generadas aleatoriamente de una forma rápida en términos computacionales. Una vez generada la población inicial, se eligen parejas de individuos para dar lugar a nuevos individuos resultado de la combinación de los genomas de sus progenitores. La población de individuos normalmente permanece con el mismo número de miembros a medida que las generaciones van pasando, con lo cual, para construir la siguiente generación se sustituye a alguno de los progenitores (al más débil de ellos con más probabilidad) por uno de sus descendientes (con mayor probabilidad el más fuerte).

Si se representa el espacio de búsqueda de un problema como una superficie de puntos en tres dimensiones muy extensa (con un número de puntos fácilmente mayor que el número de átomos en el universo, unos 10^{78}) con diferentes alturas o montículos, donde cada punto representa a una solución para el problema y su altura representa la calidad de la solución, obtendríamos una figura como la siguiente:

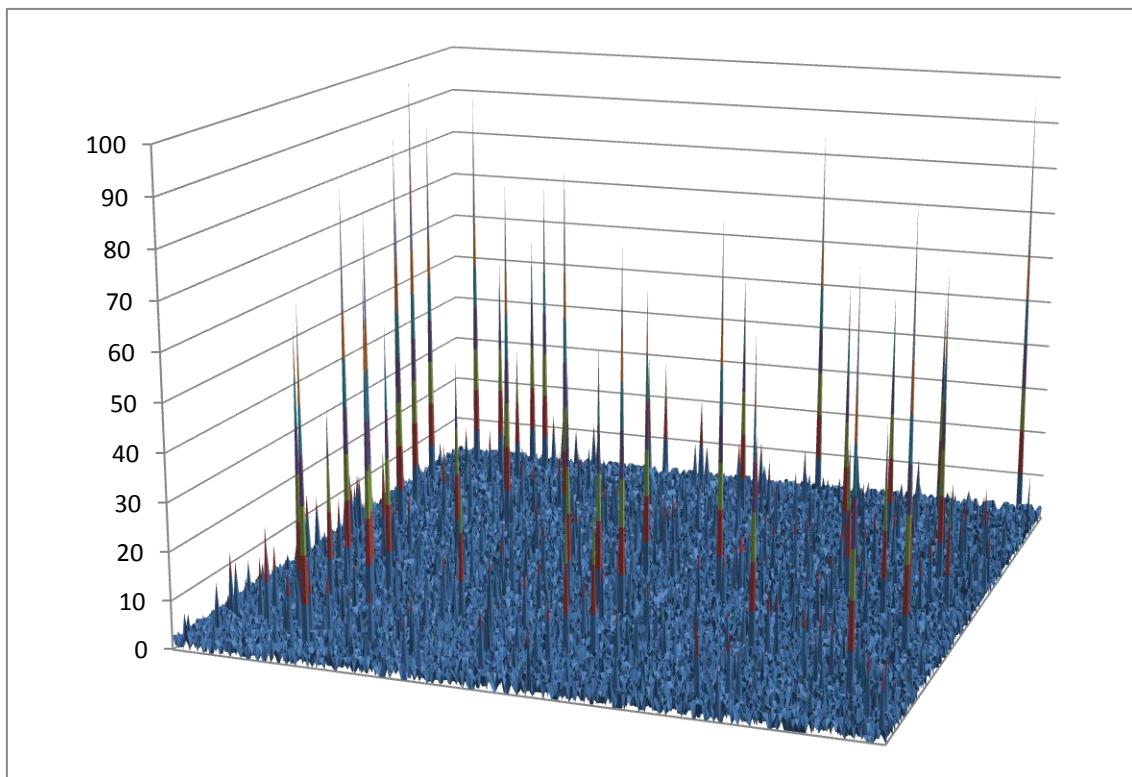


Figura 1: Representación del espacio de soluciones de un problema cualquiera. La altura de los montículos representa la calidad de los puntos (soluciones) que se encuentran en ellos. Cada montículo está formado por infinidad de soluciones.

La población inicial sería un conjunto de puntos elegidos de forma aleatoria de entre todos los de la superficie. En esta representación, los puntos cercanos unos a otros representarían a soluciones que tendrían características similares, y, del mismo modo, cuanto más lejanos estén unos de otros, más diferentes serían las soluciones que representan. En la mayoría de los problemas, el espacio que ocupan los puntos (o soluciones) con una altura (o calidad) baja es inmensamente mayor que los que tienen cierta altura (o calidad), con lo cual, en la generación inicial es muy probable que todos sus individuos representen a soluciones que tengan una calidad baja. Sin embargo, a medida que se combinan las características de las soluciones, priorizando a las mejores y sustituyendo a las de peor calidad, la altura de los puntos que representan a los individuos de la población va aumentando. En otras palabras, los puntos de la población tienden a situarse en montículos a una altura mayor con el paso de las generaciones. La cuestión de cómo representar a una solución válida de tal forma que pueda ser generada y combinada con otras soluciones y producir descendencia válida se abordará en el siguiente apartado.

Los algoritmos genéticos permiten explorar el espacio de búsqueda de forma paralela [1]. La posibilidad de combinar dos individuos distanciados considerablemente en la superficie de puntos, es decir, con características diferentes, permite a este tipo de algoritmos explorar al mismo tiempo diferentes soluciones prometedoras. Por la misma razón evitan los llamados "óptimos locales", puntos cercanos a la cima de los montículos que no permiten mejorar más la calidad de una solución a menos que ésta cambie su forma drásticamente, es decir, se explore otro montículo con una altura mayor al actual. La posibilidad de quedar estancado en la zona de un óptimo local es el mayor problema de los algoritmos basados en búsqueda por ascenso o *Hill Climbing*.

Al igual que sucede en la naturaleza, es importante asegurar que el conjunto de la población no sea homogénea, es decir, que los genomas de sus individuos sean suficientemente diferentes entre ellos. En caso de no ser así, el paso de generaciones no produciría mejoras apreciables en el conjunto de la población al no disponer los individuos de características diferentes que combinar. Volviendo al ejemplo de la superficie de puntos, los puntos escogidos en la solución inicial estarán repartidos por toda la superficie de forma aleatoria, con lo cual la variedad de genes, o características diferentes, en la población inicial está asegurada.

El número de soluciones posibles para los problemas que solucionan los algoritmos genéticos da una idea de la potencia de estos. Por ejemplo, en el problema de encontrar una función que especifique la salida correcta para cada entrada de un multiplexor de 11 entradas, cuyo espacio de búsqueda es de $2^{2048} \approx 10^{616}$ soluciones, es posible encontrar una solución exacta al problema si la población de individuos es de 4000 miembros y se ejecuta al menos durante 10 generaciones [6]. Es inmediato observar que estos problemas, con tantas soluciones posibles, no resultan en absoluto abordables por algoritmos de búsqueda exhaustiva, los cuales explotan una a una todas las soluciones posibles para poder determinar con certeza cuál es la mejor. A pesar de que los algoritmos genéticos normalmente no dan con la mejor solución al problema, la calidad de la solución dada suele ser muy cercana a la mejor posible.

El diseño de un algoritmo genético implica principalmente abordar los siguientes puntos:

- Representación de las soluciones: Forma de pasar los datos relevantes de cualquier solución a una representación que permita combinar soluciones mediante los operadores utilizados en el algoritmo. Este mapeo de las soluciones consiste en representar a cada una de ellas mediante una cadena de caracteres, normalmente en formato binario, ya que es el lenguaje natural de los programas informáticos.
- Generación de la población inicial: Método utilizado para obtener las soluciones que formarán parte de la población inicial. Normalmente las soluciones se generan de forma aleatoria, lo que permite obtener rápidamente los individuos, aunque es posible utilizar alguna técnica para intentar elevar la calidad de las soluciones generadas y que el algoritmo converja con mayor rapidez.
- Operadores: Métodos usados para obtener la siguiente generación de individuos. Algunos de los métodos más importantes son: reproducción (soluciones que pasan sin ningún cambio a la siguiente generación), cruce (dadas dos soluciones, combinar sus características) y mutación (dada una solución, variar una parte de ella de forma aleatoria). Junto a ellos también hay que establecer su frecuencia de aplicación, es decir, cuántos individuos resultan afectados por cada uno de ellos en cada generación. Estos operadores se tratarán en detalle en el siguiente apartado, ya que son muy parecidos a los utilizados en programación genética.
- Evaluación de los individuos: Función para evaluar numéricamente la calidad de un individuo en el escenario del problema, también llamada función de *fitness*.
- Criterio de finalización: Estos algoritmos no suelen encontrar a la mejor solución aunque el número de generaciones ejecutadas sea elevado, por ello es necesario decidir cuándo el proceso debe detenerse. Debido a que la calidad media de la población sigue una progresión logarítmica, es decir, mejora extraordinariamente en las primeras generaciones y permanece estable a partir de un cierto número de ellas (varía según el problema y el resto de parámetros del algoritmo), normalmente la ejecución se detiene a partir de un número máximo de generaciones escogido de antemano y/o en caso de no encontrar una solución mejor que la existente pasadas un número determinado de generaciones.

Adicionalmente a los puntos descritos, también hay que decidir si el mejor individuo en una generación pasa directamente a la siguiente (estrategia elitista). Normalmente se suele aplicar esta regla, ya que permite conservar la mejor solución encontrada hasta el momento.

Por último, se ha observado que este tipo de algoritmos funcionan especialmente bien con problemas en los que no hay una forma clara de medir la calidad de una determinada solución. Es decir, aunque haya una función que, dada una solución, devuelva un número que represente su calidad, esta función puede ser solo orientativa y no tener en cuenta todos los factores que afectan a la calidad de una solución. Esto implica que no siempre es cierto que una determinada solución con un *fitness* superior a otra, según la función que mide su calidad, en realidad sea de mejor calidad. En cierta forma, los algoritmos genéticos toleran ligeras imprecisiones en la función de *fitness*. [13]

2.2 Programación genética

La programación genética es un método derivado de los algoritmos genéticos que fue desarrollado principalmente por John R. Koza en su libro *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (1992) [6].

Principalmente, la programación genética elimina la limitación que tienen los algoritmos genéticos para representar soluciones con un tamaño desconocido a priori. Éstos últimos representan a cada solución con una cadena de caracteres de longitud fija, lo cual limita la información que pueda contener una solución. La programación genética solventa este obstáculo representando a cada solución como una estructura de nodos organizada jerárquicamente, habitualmente conocida como árbol. Nuevos nodos pueden ser añadidos a dicha estructura en cada generación, o bien nodos existentes quitados o remplazados por otros.

No obstante, en la práctica, la limitación de espacio de una solución no se puede eliminar. Cualquier máquina en donde se ejecute el algoritmo dispone de una cantidad limitada de memoria para poder guardar las soluciones de cada generación, con lo cual siempre hay un límite en la longitud de las soluciones. Además, a causa de la organización de la memoria en los computadores, cuanto más crece el tamaño de una solución, más tiempo lleva acceder a ella para combinarla con otras o calcular su calidad, con lo cual se suele limitar considerablemente el espacio máximo que puede ocupar una solución (a lo sumo unos pocos Kilobytes).

2.2.1 Codificación

Como se ha comentado, en la programación genética los individuos o soluciones se representan como una estructura jerárquica de nodos, es decir, un árbol. Cada nodo representa una función, la cual devuelve un resultado al nodo del cual desciende y tiene unos parámetros dados por sus hijos:

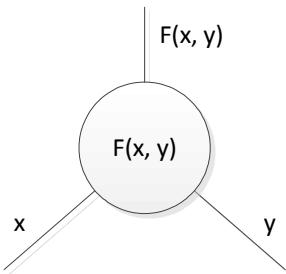


Figura 2: Comportamiento de un nodo con dos descendientes.

Los nodos terminales (hojas) representan a constantes o a datos concretos del escenario del problema. Las estructuras se evalúan recursivamente de la raíz a las hojas, y su resultado es el que se obtiene de la raíz.

Para construir una solución cualquiera se dispone de un conjunto predefinido de nodos, es decir, funciones, datos del problema y constantes. Estos nodos se eligen de acuerdo a criterios específicos para cada problema, ya que han de ser relevantes para que una combinación de ellos produzca una solución de buena calidad. También se debe tener en cuenta que cualquier combinación de estos nodos debe producir una solución válida, es decir, el resultado de la evaluación de cada tipo de nodo debe servir como parámetro para cualquier otro. Por ejemplo, si un nodo de tipo suma devuelve un entero y desciende de un nodo de tipo AND, éste debe saber interpretar como cierto o falso el entero que devuelve su descendiente. Esta propiedad asegura que al combinar dos individuos cualesquiera de la población, den como resultado un individuo válido. También es útil en la inicialización de los individuos, ya que en esta etapa se suelen escoger los nodos que los forman de manera aleatoria de entre todos los disponibles para el problema, lo cual elimina la necesidad de comprobar que los descendientes de cada nodo sean compatibles con él.

2.2.2 Operadores

Los operadores para pasar de generación utilizados en la programación genética son los mismos que en los algoritmos genéticos. Hay dos operadores fundamentales: **reproducción** y **cruce**. El operador de reproducción es el responsable de pasar sin cambios a un individuo a la siguiente generación, mientras que el de cruce se encarga de obtener uno o más individuos a partir de dos. El operador de reproducción normalmente depende de con cuánta frecuencia se apliquen el resto de operadores, ya que el número de individuos se suele mantener constante a lo largo de las generaciones, con lo cual, los individuos que no han sido alterados por otros operadores pasan sin cambios a la siguiente generación. Otro operador ampliamente utilizado, aunque no imprescindible, es el de mutación, que cambia de forma aleatoria una parte de un individuo. Al ser los operadores usados y configurables en el programa realizado para este proyecto, se detallarán en los siguientes apartados los operadores de cruce y mutación. Otros operadores incluyen: **permutación**, para intercambiar la posición de dos nodos de un mismo individuo; **edición**, para simplificar individuos sustituyendo subárboles que siempre dan el mismo resultado por nodos con la constante resultado de la evaluación; **encapsulación**, para crear un nodo que contenga la funcionalidad de un subárbol de un individuo; **destrucción**, para eliminar, normalmente en las primeras generaciones del algoritmo, individuos con una calidad muy baja. [6]

Cada operador se aplica a un cierto número de individuos en cada generación, o, visto de otra forma, con una cierta probabilidad. La elección de las probabilidades de forma adecuada permite al algoritmo encontrar soluciones de alta calidad de forma más o menos rápida. En casos extremos de una mala elección de probabilidades, puede suceder que la calidad media de la población de individuos no mejore o empeore con el paso de generaciones, o bien que varíe mucho de generación en generación. Una probabilidad de cruce baja provoca que la siguiente generación sea bastante parecida a la actual, con lo cual se necesitará un número elevado de generaciones para lograr subir la calidad media de la población. Sin embargo, una probabilidad alta de mutación impide que se mejore significativamente la calidad de los individuos de la población, ya que la generación siguiente tiene características muy diferentes a la anterior y no

se permite que las que en realidad mejoran la calidad de sus individuos emergan. Este último caso sería parecido a una búsqueda aleatoria de la mejor solución. De todas formas, la elección de las probabilidades para los operadores es similar para muchos problemas a los que se aplica programación genética, con lo que su elección no suele suponer un gran problema. [13]

2.2.2.1 Cruce

El operador de cruce es el más importante en toda la familia de los algoritmos genéticos porque combina las características de dos soluciones existentes con el fin de obtener otras dos con las características de los padres en una proporción diferente. Junto con una selección que favorezca la supervivencia de los mejores individuos en cada generación, permite que las características que dan soluciones de mejor calidad sean más comunes en el conjunto de la población con el paso de generaciones, elevando la calidad media de ésta.

En programación genética, el funcionamiento del operador consiste en escoger dos subárboles al azar de sendos individuos, y sustituir en ambos los subárboles escogidos por el del individuo contrario:

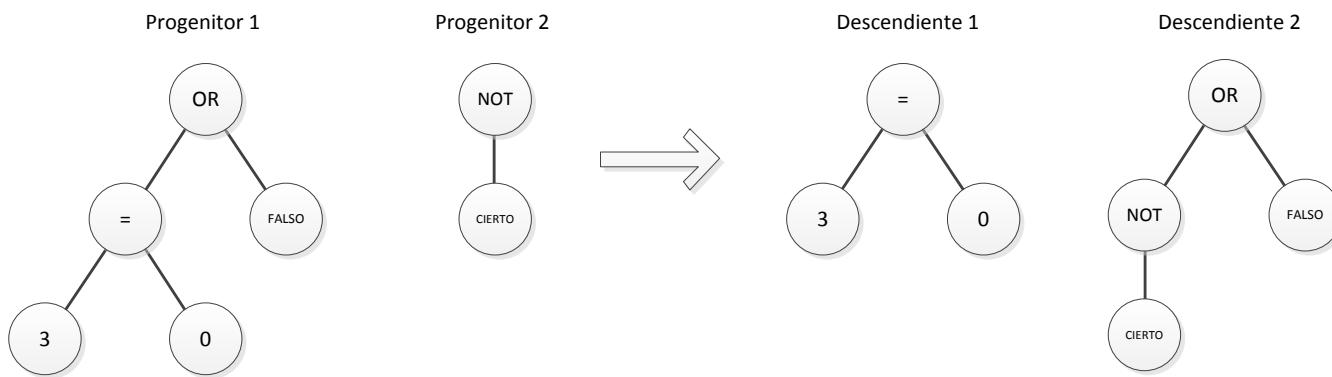


Figura 3: Ejemplo de cruce entre el progenitor 1 y 2. El subárbol seleccionado en el primero es el que tiene la raíz en el nodo "=", y en el segundo individuo es el que encabeza el nodo "NOT" (todo el individuo).

De esta forma siempre se obtienen dos nuevos individuos. La elección de los individuos progenitores depende de su calidad: tendrá más probabilidad de ser escogido aquél individuo con una calidad superior al resto; es poco probable que los peores individuos de la población resulten elegidos.

Como apunte, en los algoritmos genéticos, que representan a los individuos mediante cadenas de caracteres de longitud fija, existen otras formas de aplicar el operador. En éstos, se escoge un punto al azar de la cadena de caracteres que representa a cada individuo, y se intercambian todos los caracteres anteriores o bien todos los posteriores a ese punto. No existe una limita-

ción al número de puntos a escoger para hacer el intercambio, tal como se muestra en la siguiente figura:

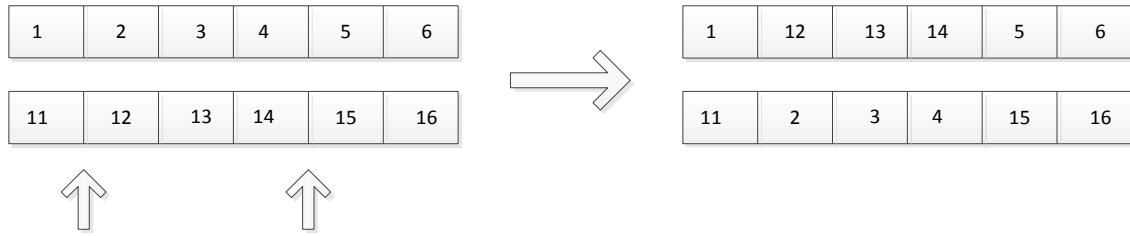


Figura 4: Ejemplo de cruce por dos puntos (señalados por las flechas) en un algoritmo genético.

2.2.2.2 Mutación

Como se ha comentado en el apartado 2.1, es importante asegurar que la población tenga características variadas para poder explorar distintas combinaciones de ellas. Aplicando el algoritmo solo con los operadores de cruce y reproducción, se tiende a llegar a un punto en que la población se vuelve homogénea, ya que la mayoría de sus individuos poseen el mismo conjunto de características ya optimizadas en la proporción justa para dar los mejores resultados. Sin embargo, es posible que haya otras características no presentes en la población inicial o que se hayan perdido en las primeras generaciones de la evolución por diferentes motivos (por ejemplo, su combinación con alguna característica resulta en individuos de mala calidad), que podrían permitir generar individuos mejores. [6]

Para introducir nuevas características en la población de individuos y seguir explorando nuevas combinaciones entre ellas, es necesario el operador de mutación. En programación genética, su funcionamiento consiste en escoger al azar un subárbol de un individuo y remplazarlo por otro generado de manera aleatoria:

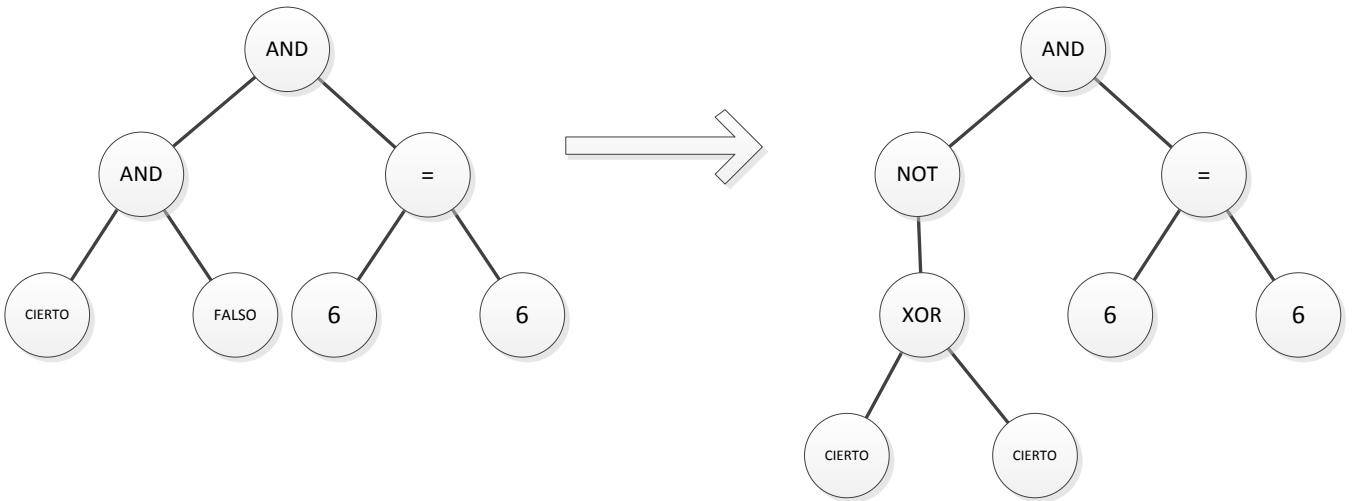


Figura 5: Ejemplo de mutación de un individuo. El subárbol a remplazar es el que tiene como raíz el nodo "AND" del segundo nivel del árbol.

Es importante señalar que este operador es más necesario cuanto menor sea el tamaño de la población, ya que con un número pequeño de individuos es probable no disponer en estos de todas las características necesarias para encontrar soluciones de buena calidad. En la práctica, las poblaciones suelen ser considerablemente grandes (entorno a los 500 individuos), y por ello, la probabilidad con la que se aplica el operador es baja (5% o inferior).

No obstante, mediante este operador es difícil conseguir introducir en un individuo características que aparecen en una porción reducida del espacio de búsqueda, ya que todos los cambios se hacen de manera aleatoria. Este hecho dificulta introducir en la población de individuos características poco comunes en el espacio de soluciones.

2.2.3 Función de *fitness*

La función que calcula de cuánta calidad es un individuo es fundamental para el correcto funcionamiento del algoritmo, ya que guía la evolución de la población. Los otros componentes involucrados en el algoritmo seleccionan los individuos para la siguiente generación dependiendo del valor que les devuelve dicha función de *fitness*.

Esta función emula en la realidad a las probabilidades que tiene un individuo de reproducirse, o dicho de otra forma, su tiempo esperado de supervivencia y número de descendientes. No obstante, esta no es una simulación fiel de la realidad, ya que en los algoritmos genéticos y programación genética, un individuo de mucha calidad puede sobrevivir un número indefinido de generaciones porque podría no ser remplazado por otro. Esto se puede evitar añadiendo a la función de *fitness* una penalización según la edad o número de generaciones desde que existe el individuo. [6]

La función se aplica a través de cada generación a cada individuo de la población. En primer lugar se evalúa el individuo, es decir, se obtiene el resultado de la raíz recorriendo recursivamente todos los nodos que componen el individuo. En segundo lugar, el resultado devuelto es aplicado al entorno del problema por la función de *fitness* para determinar el comportamiento del individuo y poder obtener su calidad. Además del resultado de la evaluación del individuo, la función de *fitness* también puede tener en cuenta otros factores para determinar su calidad, como por ejemplo el número de generaciones desde que existe en la población o la homogeneidad en la población (cuando las diferencias entre la calidad de los individuos se vuelven pequeñas, es decir, la mayoría tienen una calidad similar, entonces se enfatizan las diferencias pequeñas) [6].

2.2.4 Parámetros por defecto

Este capítulo finaliza mostrando los parámetros usados para aplicar el algoritmo de programación genética en un problema genérico (según Koza [6]):

Parámetros principales	
Tamaño de la población	500
Número máximo de generaciones	50

Parámetros secundarios	
Probabilidad de cruce	90%
Probabilidad de reproducción	10%
Tamaño máximo de los árboles creados durante la evolución	17
Tamaño máximo de los árboles creados en la inicialización	6
Probabilidad de mutación	0%
Probabilidad de permutación	0%
Probabilidad de edición	0%
Probabilidad de encapsulación	0%
Condición para la destrucción	Ninguna
Objetivo de destrucción	0%

Características cualitativas
El método de selección para el operador de cruce es proporcional al <i>fitness</i> del individuo.
El método de emparejamiento es proporcional al <i>fitness</i> (es más probable cruzar dos individuos cuanto más parecido sea su <i>fitness</i>).
La estrategia elitista no se utiliza.

Referencias utilizadas en el capítulo: [1], [6], [13], [14].

3. MERCADOS FINANCIEROS

3.1 Características generales

Un mercado financiero es cualquier lugar, sistema o mecanismo en/mediante el cual se intercambian activos financieros entre inversores. Un activo financiero es un instrumento emitido por empresas, gobiernos u otras entidades económicas con el fin de elevar su capital para financiar sus actividades. Los activos financieros se agrupan en las siguientes categorías: deuda pública, pagarés de empresas, depósitos bancarios, acciones, renta fija de empresas privadas y dinero de curso legal.

Estos mercados proporcionan seguridad a los inversores que hacen uso de ellos, ya que están fuertemente regulados para asegurar la legalidad de los intercambios. El precio de intercambio de un activo viene dado por la oferta y la demanda, de tal forma que su precio de compra/venta bajará si hay más cantidad de activos en venta (oferta) que los que van a ser comprados (demanda), y el activo se encarecerá en caso contrario.

Existen agentes responsables de ejecutar los intercambios de activos ordenados por los inversores en los mercados financieros. Estos agentes, también denominados *brokers*, están designados por los reguladores de cada mercado, y deben de asegurar que los inversores que operan a través de ellos dispongan efectivamente de los activos que van a ser intercambiados, entre otros requerimientos. Tanto los *brokers* como los mercados cobran comisiones a los inversores por las transacciones realizadas, las cuales varían dependiendo del mercado y del *broker*, aunque por lo general las comisiones son menores cuando el volumen de transacciones en el mercado en cuestión es elevado. Por ejemplo, los mercados de divisas suelen cobrar comisiones muy bajas a causa de que su ámbito es global y consecuentemente se realizan un gran número de intercambios. Sin embargo, en el mercado de acciones las comisiones son superiores, ya que la mayoría de intercambios son realizados dentro del país al que pertenece el mercado de valores y el volumen de transacciones es muy inferior al ejemplo anterior.

A partir de la aparición de los *brokers* en Internet y la posibilidad de realizar intercambios de activos *online* desde cualquier lugar, el número de transacciones en todos los mercados financieros ha aumentado extraordinariamente, lo que ha permitido bajar las comisiones cobradas a los inversores. Estas comisiones son menores, en proporción al valor de los activos intercambiados, a medida que el número de activos en una transacción aumenta. Por ejemplo, los inversores institucionales (fondos de inversión, bancos,...), que realizan operaciones con un volumen muy elevado de activos, pueden tener unos costes totales inferiores a 0,1% del valor de los activos intercambiados en la operación, mientras que un inversor particular puede llegar a soportar costes superiores al 1% del valor de los activos. No obstante, actualmente existen *brokers* que ofrecen cobrar una tasa fija, en lugar de un porcentaje, para un rango de activos intercambiados. Por ejemplo, el inversor puede pagar 20€ para intercambiar un número de activos comprendido entre 1.000 y 10.000 en un periodo de tiempo de tres meses. Esto tiene la finalidad de hacer más atractiva la inversión en los mercados a los pequeños inversores.

En los siguientes apartados se explicarán los detalles de los mercados de acciones, al ser el principal objetivo de este proyecto.

3.1.1 Mercado de acciones

En los mercados de acciones se intercambian participaciones en las empresas presentes en él. Cualquier inversor puede adquirir estas participaciones, cuyo objetivo es ser partícipe de la empresa y sus beneficios. Al comprar acciones de una empresa, el inversor se convierte en socio de ella y, si el número de participaciones que posee es suficientemente elevado, tiene derecho a voto en las decisiones del consejo de administración.

El objetivo de las empresas que colocan participaciones en estos mercados es el de obtener financiación. Las participaciones son adquiridas por inversores, y, el capital aportado por éstos al comprarlas pasa a formar parte del pasivo de la empresa.

Es de esperar que cuanto más elevado sea la relación beneficios / número de participaciones de la empresa en el mercado, más atractivo sea para los inversores adquirir sus participaciones a causa del éxito de la empresa; lo que provocará que haya más inversores queriendo formar parte de ella. Esto a su vez hará subir el precio de las participaciones, al aumentar su demanda, lo que pondrá a disposición de la empresa más capital para finanziarse y seguir creciendo. En cierto modo, los inversores suelen "premiar" a las empresas que marchan bien aportándoles más capital en forma de acciones que, a su vez, les permite seguir creciendo.

Otro motivo importante para adquirir acciones de una empresa es el dividendo que reparte a los inversores. Estos dividendos provienen de los beneficios que genera, de tal forma que la cantidad a repartir es la misma a cada una de las participaciones de la empresa. Cada empresa tiene su política en cuanto a la retribución dada como dividendo a los inversores; mientras unas destinan prácticamente la totalidad de su beneficio a este fin, otras no reparten ningún dividendo en absoluto. Los dividendos se suelen repartir con una frecuencia pequeña, normalmente unas dos o tres veces al año, y, en el momento de repartirse, el precio de las acciones de la empresa disminuye lo mismo que el importe del dividendo que es abonado en la cuenta del inversor. Por ello, para que el precio de la acción no disminuya con el tiempo si no entran nuevos inversores, la empresa repone el capital repartido como dividendo poco a poco, con lo cual, el precio de la acción debería de mantenerse con una media estable si no entrasen ni saliesen inversores de ella. Esto implica que la inversión debe mantenerse durante un periodo de tiempo prolongado para poder aprovechar su rentabilidad. Hay inversores que no están dispuestos a mantener su inversión durante tanto tiempo y/o buscan otra forma de obtener beneficios.

Como se ha comentado anteriormente, las expectativas que tengan los inversores de la marcha de la empresa en el futuro, hará que el precio de sus participaciones varíe: un incremento de sus beneficios provocará un aumento en la relación beneficios / número de participaciones y consecuentemente será más atractiva para los inversores y aumentará el precio de sus participaciones; y a la inversa si sus beneficios disminuyen. Esta variación en el precio de las participaciones es la principal guía de los inversores que buscan especular con su precio. Estos inversores compran participaciones esperando que en un futuro próximo su precio de com-

pra/venta aumente y puedan venderlas a un precio superior al de compra. Este tipo de inversión se comentará con más detalle en próximos apartados.

Los mercados de acciones se denominan también bolsas de valores. Prácticamente todos los países disponen de al menos un mercado de acciones donde cotizan muchas de las empresas medianas y grandes del país en cuestión. Cada una de estas bolsas tiene un valor numérico, llamado **índice bursátil**, que indica cuánto valen todas las empresas que cotizan en él, o, en otras palabras, es un indicador de cuánto dinero tienen los inversores en acciones. Este valor no tiene nada que ver entre índices. Por ejemplo, el SP 500, uno de los índices con mayor capitalización del mundo, oscila en un rango comprendido entre los 1150 - 1200 puntos, mientras que el IBEX 35, con una capitalización muy inferior al anterior, está entorno a los 8000 puntos (datos a Noviembre de 2011). Este valor depende mayoritariamente del precio de compra/venta de las acciones de las empresas participantes en la bolsa de valores, y, en menor medida, de pequeños ajustes realizados para tener en cuenta los dividendos, ampliaciones o reducciones de capital, fusiones, absorciones,... de las empresas cotizantes. Hay dos formas diferentes de calcular el componente mayoritario: proporcionalmente a la capitalización de cada empresa dentro del índice o contando a todas las empresas de igual forma (media aritmética). De todos modos, la forma exacta para calcular el valor de un índice es diferente entre uno y otro, y es complicado encontrar dos índices tengan la misma forma de calcularlo.

Existe la posibilidad de invertir directamente en un índice sin tener que hacerlo individualmente en cada una de las empresas que lo forman, de tal forma que el resultado sería similar a repartir la inversión entre todas las empresas que cotizan en dicho índice. La forma de invertir en un índice difiere ligeramente de la forma habitual de inversión en acciones, ya que hay que hacerlo mediante instrumentos financieros conocidos como *Exchange Traded Funds (ETFs)* o bien mediante otros instrumentos derivados.

3.2 Estrategias de inversión

Los inversores suelen disponer en su cartera de distintos tipos de activos, cada uno de ellos encasillable según diferentes criterios: riesgo, duración, mercado, país, etc. Es necesario tener alguna estrategia para poder dirigir esta cartera, la cual dependerá de las características particulares del inversor: tiempo para dedicar al seguimiento, criterio seguido para tomar las decisiones, riesgo asumido, herramientas y datos a su alcance, etc. En muchas ocasiones, los inversores delegan la gestión de su cartera a gestores profesionales, en cuyo caso, el inversor normalmente decide el riesgo que está dispuesto a asumir como factor más importante. Cuánto más riesgo se asuma en los activos escogidos, mayores pueden ser las ganancias y las pérdidas que se produzcan en el futuro sobre dichos activos.

Tanto si la gestión la realiza el propio inversor como si la hace un profesional, existen dos grupos en los que clasificar la forma de dirigir una cartera de activos: activa y pasiva.

Las estrategias pasivas están orientadas a la inversión a largo plazo, ya que el tiempo destinado a la gestión de la cartera es pequeño. Consecuentemente, los activos en cartera se modifican (compran o venden) cada cierto periodo prolongado de tiempo y los costes derivados de

las transacciones efectuadas son muy pequeños. Estas estrategias suelen buscar un riesgo bajo y por ello invierten en índices o en más de un activo al mismo tiempo, ya que de esta manera la cartera no queda tan expuesta a la marcha de una determinada empresa o sector, sino que está diversificada en muchas empresas de un país. Esta estrategia considera que el mercado es eficiente y que no es posible batir el beneficio dado por un índice siguiendo la estrategia *buy and hold* (ver apartado 3.3.3). Es por este motivo que lo que se busca es igualar el beneficio de un índice o sector determinado, y no batir al mercado.

Lo contrario de las estrategias pasivas son las llamadas activas. Éstas buscan batir los beneficios dados por un índice o sector mediante la compra de algunos de sus componentes o empresas que los componen. Esta selectividad en los activos comprados implica un estudio detallado de la situación de todos los componentes del índice en el que se desea invertir (ver apartado 3.3), para poder determinar en cuáles de ellos hay más potencial de revalorización. Por ello, los inversores que se decantan por este tipo de gestión, creen que el mercado en el que van a invertir no es del todo eficiente (ver apartado 3.3.3) y que pueden aprovechar las ineficiencias para obtener beneficios. Por lo general, estas estrategias tienen unos costes derivados de las transacciones efectuadas superiores a las pasivas, ya que se producen más transacciones para poder aprovechar todas las oportunidades de obtener beneficios. Consecuentemente, los activos que forman parte de la cartera se modifican frecuentemente y las inversiones realizadas no suelen ser a largo plazo. El riesgo de los activos adquiridos siguiendo esta estrategia puede ser ajustado mejor que con las estrategias pasivas, ya que los activos pueden ser escogidos de acuerdo al riesgo o volatilidad históricos de cada uno de ellos (es de esperar que el riesgo sea aproximadamente constante con el tiempo). No obstante, al adquirir acciones de empresas individuales, normalmente el riesgo es más elevado que con estrategias pasivas, ya que existe la pequeña posibilidad de que, una empresa que históricamente ha tenido el precio de sus acciones estable por ser competitiva en su sector, por algún motivo excepcional, la empresa pierda cuota de mercado y sus acciones bajen de forma acusada por este motivo. La posibilidad de darse esta situación se reduce prácticamente a cero en el caso de un índice, ya que tiene a muchas empresas representadas en él (decenas o cientos) y, la marcha negativa de una de ellas, no tiene un efecto importante sobre el valor del índice.

Atendiendo al comportamiento de cada estrategia en la realidad, en contra de lo que puede parecer debido al coste que implica su gestión, parece darse el caso de que las estrategias pasivas obtienen mejores resultados que las activas en muchos casos. Por ejemplo, analizando los fondos de inversión en EEUU, Europa y Asia que ofrecen las 15 mayores empresas en el sector entre 2004 y 2009, el 80% de los fondos que utilizan una gestión activa de la cartera obtuvieron unos beneficios inferiores a sus correspondientes estrategias pasivas [10]. Otro ejemplo concreto se produjo entre 1984 y 2004 en el índice SP 500, en el que la media del beneficio de los fondos de inversión con gestión activa que operaron sobre el índice acumularon un beneficio medio anual del 3,7%, mientras que el correspondiente beneficio medio del índice fue del 13,2% [10]. No obstante, es obvio que los resultados dependen de la habilidad que tenga el gestor de la cartera en detectar oportunidades. Sin embargo, hay indicios que en algunos mercados emergentes los fondos con gestión activa de la cartera se pueden comportar mejor que utilizando estrategias pasivas, ya que en estos mercados la hipótesis de la eficiencia del mercado no es tan clara.

ciencia del mercado podría no ser cierta (ver apartado 3.3.3), con lo cual sería posible aprovechar las ineficiencias que se producen para obtener beneficios.

El programa implementado para este proyecto está orientado a una gestión activa de la cartera, ya que utiliza el análisis técnico como criterio de inversión y consecuentemente no considera cierta la hipótesis de la eficiencia del mercado (ver siguiente apartado).

3.3 Criterios de inversión

Los criterios que siguen los inversores para adoptar sus decisiones son muy diversos, aunque se pueden clasificar en dos: análisis fundamental y análisis técnico. Es cierto que muchas veces se utilizan los dos al mismo tiempo, o al menos, uno de ellos tiene algún peso en las decisiones tomadas.

Los inversores buscan invertir en aquellas empresas que les den mayores beneficios en un futuro, tanto en forma de dividendos como de revalorización del precio de sus acciones. Si una empresa reparte grandes dividendos pero el precio de sus acciones a la larga baja y anula toda ganancia proveniente vía los dividendos, no resultará atractiva para invertir en ella. En cambio, si una empresa no reparte dividendos pero el precio de sus acciones se revaloriza constantemente, sí que resultaría una buena inversión.

3.3.1 Análisis fundamental

El análisis fundamental consiste en determinar cuál es el precio objetivo al que deberían cotizar las acciones de una empresa, teniendo en cuenta su entorno económico, y aprovechar aquellas que coticen con mayores descuentos para comprarlas. Esta forma de inversión se basa en que, a largo plazo, el precio de las acciones reflejará los fundamentales o la marcha de una empresa.

Son muchos los indicadores que se pueden utilizar para comprobar la "salud" de una empresa, aunque principalmente se tienen en cuenta los siguientes tres informes: balance, resultados y flujo de caja. El informe del balance de la empresa refleja el importe de los activos, el pasivo, el capital aportado por los accionistas y el valor de liquidación de la compañía. Cuanto menor sea la relación entre los activos y el pasivo, menos compromisos financieros tiene la empresa y mayor capacidad para financiarse tendrá, ya que no posee elevadas deudas. La cuenta de resultados detalla los ingresos, gastos y beneficios obtenidos en el periodo de cálculo. A mayores beneficios, en comparación con ejercicios anteriores, más valor tendrá previsiblemente la empresa en el futuro. En cuanto al flujo de caja, informa de los movimientos de capital y detalla de dónde proviene y en dónde se gasta. Cuanto mayor sea el flujo de caja excedente y mayor sea el flujo proveniente de las actividades empresariales (no de la financiación externa u otros conceptos), mayor autonomía tiene la empresa para mantenerse por sí sola sin recurrir a financiarse externamente y mayor podrá ser el dividendo repartido a los inversores.

Todos los datos nombrados se pueden cuantificar y medir de forma objetiva. Esto da lugar a indicadores que pueden ser utilizados para medir de forma automática el estado de la empresa. Algunos de los más utilizados son los siguientes [7]:

- Beneficio por acción [*Earnings Per Share (EPS)*]: Indica los beneficios generados por la compañía en relación al número de acciones que posee. Es uno de los indicadores más comunes para estimar el precio al que deberían cotizar las acciones de una empresa:

$$\text{EPS} = \frac{\text{beneficio neto}}{\text{número de acciones}}$$

- Precio a beneficio [*Price to Earnings (P/E o PER)*]: Indica la relación que existe entre el precio por acción y el beneficio por cada una. Un valor alto indica que los inversores tienen expectativas de que el beneficio de la empresa aumente considerablemente en el futuro o bien que las acciones están sobrevaloradas; un valor bajo indica lo contrario. Para poder interpretar su valor, se ha de comparar con el de otras empresas del sector o el histórico de la propia empresa:

$$\text{PER} = \frac{\text{precio por acción}}{\text{EPS}}$$

- Precio a valor [*Price to Book (P/B)*]: Indica la relación existente entre el precio por acción y el valor de liquidación de la empresa. Un resultado bajo puede indicar o bien que las acciones están infravaloradas o bien que existe algún problema en la empresa; un valor alto indica que las acciones están sobrevaloradas.
- Beneficio por acciones [*Return On Equity (ROE)*]: Indica cuánto beneficio genera la empresa en relación al capital aportado por los accionistas. Es usado para comparar a empresas pertenecientes al mismo sector.
- Beneficio por activos [*Return On Assets (ROA)*]: Indica el beneficio generado por la empresa en relación al número de activos que posee. Como ROE, es útil cuando se comparan compañías del mismo sector.

A parte de estos indicadores, existen otros factores involucrados en analizar fundamentalmente una empresa que no se pueden medir de una forma tan objetiva como los anteriores. Estos indicadores son los llamados factores cualitativos; los más importantes son la organización interna de la empresa y el estado que ocupa dentro del sector al que se dedica. Sin embargo, estos indicadores suelen tener menos peso en la decisión de invertir en la empresa, ya que tienen más dificultad para ser medidos y cuantificados.

Por último, cabe destacar que, debido a que no se predice cuándo los fundamentales se reflejarán en el precio de las acciones, estas inversiones están orientadas a ser de largo plazo. Por ello, el análisis fundamental tiene un peso importante en el diseño de estrategias pasivas.

3.3.2 Análisis técnico

El análisis técnico consiste en tomar decisiones de inversión teniendo en cuenta únicamente el histórico de precios y de volumen de negociación de las acciones de una empresa. Este tipo de análisis se basa en tres principios [7]:

- Todo está incluido en el precio: Todos los factores que pudieran afectar al precio futuro de las acciones, sean psicológicos, económicos, políticos o de otra índole, están reflejados en el precio actual. Consecuentemente, los cambios en el precio en las acciones se producen por cambios en un factor externo que afecta la marcha de la empresa. Esto contrasta con el análisis fundamental, el cual analiza los factores externos que pueden afectar a la cotización de la empresa y determina si invertir en ella en base a estos factores.
- Los precios se mueven siguiendo tendencias: La variación del precio en las acciones de las empresas no se produce de manera aleatoria, sino que sigue una determinada tendencia. Detectar un cambio de tendencia en sus primeras fases, es decir, el momento en el que los precios van a moverse en dirección contraria a la actual, es uno de los principales objetivos del análisis técnico.
- La historia se repite: Los patrones que siguen los cambios en los precios en el pasado suelen repetirse en el futuro. Los inversores suelen seguir los mismos sistemas de inversión y además los factores psicológicos por los que se dejan influir algunos inversores no cambian con el tiempo, ya que están implícitos en el comportamiento humano.

El análisis técnico puro está formado por numerosos indicadores numéricos que dan datos exactos sobre el estado de una tendencia. Hay más de 200 indicadores disponibles, aunque en la mayoría de casos se usan solo un subconjunto reducido de ellos, ya que resultaría inabordable usarlos todos. Muchas veces se utiliza más de un indicador para averiguar un estado concreto del precio (por ejemplo, si hay sobrecompra o sobreventa en el precio), con el fin de tener mayor certidumbre sobre la evolución futura de los precios. A continuación se detallan algunos de los indicadores más utilizados:

- Medias móviles: Indican cuál ha sido el precio medio de las acciones en un determinado número de días previos. Normalmente se coge el precio de cierre de cada sesión de negociación y se divide por el número de sesiones. El propósito de este indicador es ignorar las variaciones de precio que se producen en el corto plazo y dejar entrever cuál es la tendencia del precio. Cuando la tendencia es alcista, la media en un día previo es inferior a un día posterior, y viceversa. Es común comprar cuando el precio de cierre de la acción supera al valor indicado por la media.

No obstante, en la realidad se suelen utilizar dos o más medias móviles con diferente base para tomar decisiones. Por ejemplo, utilizando dos medias se decidiría comprar cuando la media de más corto plazo supera a la media de más largo plazo, y vender en caso contrario:

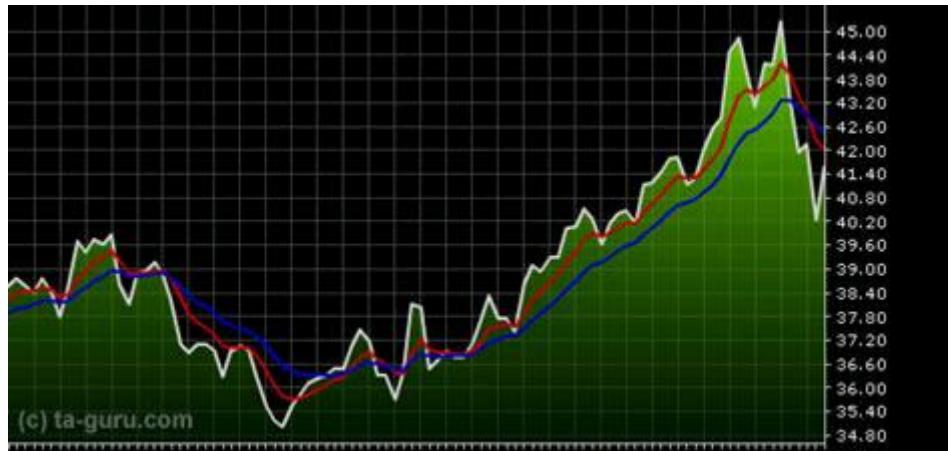


Figura 6: Representación de dos medias móviles sobre el gráfico de precios de algún activo.

Una técnica ampliamente utilizada es utilizar tres medias móviles para decidir cuándo comprar y vender una acción. La tendencia actual sería alcista cuando cada media supera a las otras de más largo plazo, y bajista en caso contrario. La decisión de compra se tomaría cuando, en una tendencia bajista, la media de más corto plazo supera a las otras dos (para mayor seguridad, hay que esperar a que la media de plazo intermedio supere a la de más largo plazo). Del mismo modo, la decisión de venta se produciría cuando, en una tendencia alcista, la media de más corto plazo sea inferior a las otras dos (confirmada cuando la media de plazo intermedio sea inferior a la de más largo plazo). Por lo general, los días con los que calcular cada una de las medias son 12, 20 y 50.

- **Momentum:** Anticipa cambios en la tendencia actual de los precios mediante la medición del ritmo de las subidas y bajadas. Cuando la evolución de los precios es alcista, el indicador es positivo, y cuanto más elevado sea su valor, más fuerte es la tendencia. En el caso de que la situación sea bajista, el indicador da un resultado negativo; y más fuerte es la tendencia cuanto más negativo sea. Se calcula de la siguiente forma:

$$\text{Momentum} = \text{precio de cierre}_i - \text{precio de cierre}_{i-n}$$

Normalmente n tiene por valor 20. Este indicador también se usa para detectar el estado de la tendencia actual. Si la tendencia actual es alcista y el precio marca un nuevo máximo pero el indicador no lo hace, entonces la tendencia se vuelve débil. Lo contrario ocurre en el caso de una tendencia bajista: si el precio marca un nuevo mínimo en la tendencia pero el indicador no lo hace, la tendencia pierde fuerza:



Figura 7: Representación del comportamiento del indicador "momentum"

- Índice de fuerza relativa [Relative Strength Index (RSI)]: Indica principalmente si una acción está sobrevalorada o infravalorada. Se calcula de la siguiente forma:

$$RSI = 100 - \frac{100}{1+RS}$$

$$RS = \frac{\text{ganancia media}}{\text{pérdida media}}$$

$$\text{ganancia media}_i = \frac{\text{ganancia día actual} + (n-1) \text{ ganancia media}_{i-1}}{n}$$

$$\text{pérdida media}_i = \frac{\text{pérdida día actual} + (n-1) \text{ pérdida media}_{i-1}}{n}$$

Donde la ganancia media de un día equivale a la diferencia entre el precio de cierre del día actual y el del día anterior; n equivale al número de días del periodo de cálculo, normalmente 14.

Un valor superior a 70 indica sobrecompra, mientras que 30 o menos indica sobreventa de la acción. También es utilizado para detectar cambios de tendencia antes de que estos se produzcan: si el precio de la acción se encuentra en tendencia bajista y el precio marca un nuevo mínimo pero el RSI no lo hace, entonces la tendencia bajista se encuentra débil y es probable que se acabe; al contrario ocurre con una tendencia alcista:



Figura 8: Representación del comportamiento del indicador RSI

- Bandas de *Bollinger*: Indican, al igual que el *RSI*, si una acción está sobrecomprada o sobrevendida. Consiste en el cálculo de la media del precio de cierre de un número determinado de sesiones y la desviación estándar de la muestra. Se calcula de la siguiente forma:

$$\text{Media}(n) = \frac{\sum_{i=1}^n \text{precio cierre}_i}{n}$$

$$\text{Desv. estándar}(n) = \sqrt{\frac{\sum_{i=1}^n (\text{precio de cierre}_i - \text{Media}(n))^2}{n}}$$

$$\text{Límite superior} = \text{Media}(n) + \text{Desv. estándar}(n)$$

$$\text{Límite inferior} = \text{Media}(n) - \text{Desv. estándar}(n)$$

Normalmente se da a n un valor de 20. De esta forma se obtiene un límite superior e inferior dentro del cual estarán el 95% de los precios de cierre calculados:



Figura 9: Representación de las bandas de Bollinger. Las líneas azules son los límites superior e inferior, y la línea gris es la media de los precios.

Si el precio actual de la acción está próximo a uno de los límites, entonces ésta está sobrecomprada (límite superior) o sobrevendida (límite inferior). No obstante, no es infrecuente que, al aproximarse el precio a uno de los límites, continúe durante bastantes sesiones cerca de ese límite. Es por ello que es importante confirmar la sobre compra o sobreventa con otros indicadores.

Otra interpretación es que, si el precio cruza uno de los límites y a continuación rebota y cruza la línea que marca la media, probablemente llegará al límite contrario.

A parte de este tipo de indicadores, hay otra variante del análisis técnico que consiste en analizar gráficos de precios y detectar formaciones que presumiblemente den lugar a una variación futura en estos precios. Este tipo de análisis es más subjetivo, ya que es posible que una persona observe una formación en los precios y otra no detecte la formación o bien la interprete de otro modo; además es posible interpretar una formación cuando aún no está completamente formada, lo cual puede producir una falsa entrada en el mercado. Al no utilizarse en este proyecto el análisis de gráficos, no se comentará en detalle.

3.3.3 Hipótesis de la eficiencia del mercado

La hipótesis de la eficiencia del mercado es uno de los principios más importantes a tener en cuenta para operar en cualquier mercado financiero. Esta hipótesis sostiene que no es posible batir al mercado, es decir, superar los beneficios que se obtienen comprando en un momento dado y vendiendo en otro posterior. La razón es que los precios de los activos ya reflejan toda la información que les es relevante en cualquier momento. Por ello, según esta hipótesis, las acciones ya cotizan a su valor adecuado en todo momento, lo que impide obtener beneficios, por ejemplo, comprando acciones sobrevendidas o vendiendo sobrecompradas ya que nunca ocurrirá ninguno de los dos casos.

No obstante, existen tres variantes diferentes de esta hipótesis según la cantidad de información que se refleja en los precios en cada momento. La primera de ellas, conocida como el tipo fuerte, dice que no es posible obtener ningún beneficio en absoluto en el mercado, ya que toda la información que pueda afectar al activo en cuestión en el momento actual, incluso la que no es conocida públicamente, ya está reflejada en su precio, tal y como se ha comentado en el anterior párrafo. En segundo lugar se encuentra la variante semi-fuerte, que sostiene que el precio del activo refleja toda la información pública que pueda afectar a su cotización. En último lugar, la variante débil defiende que solo está reflejada en el precio actual toda la información que se incluye en el histórico de precios del activo.

De esta forma, operando en el mercado utilizando el análisis técnico no sería posible obtener beneficios si alguna de las tres variantes de la hipótesis fuera cierta. Sin embargo, sí sería posible obtener beneficios mediante el análisis fundamental en el caso de que el mercado fuera eficiente en su variante débil únicamente.

No hay consenso con respecto a si alguna de las variantes de la hipótesis es cierta, aunque algunos estudios realizados sobre ello concluyen que, en los mercados de los países desarrollados, podría ser cierta la variante débil. Sin embargo, no existen argumentos que clarifiquen

por completo estas conclusiones. Este proyecto tiene entre sus objetivos demostrar si es cierta la versión débil, aplicando las reglas de inversión producidas mediante programación genética, basadas en indicadores de análisis técnico, y viendo si los resultados dados consiguen batir a una estrategia de inversión pasiva de forma consistente en el tiempo.

Referencias utilizadas en el capítulo: [7], [10], [15], [16], [17], [18], [28].

4. DISEÑO DEL ALGORITMO

En este capítulo se detallará la estructura del algoritmo de programación genética implementado en este proyecto, así como la adaptación de la técnica de programación genética para encontrar reglas de inversión utilizando indicadores de análisis técnico. Todas las ideas explícadas en este capítulo son propias a menos que se indique lo contrario.

4.1 Adaptación de la programación genética para encontrar reglas de inversión

Uno de los principales objetivos de este proyecto es el diseño de un algoritmo que encuentre de forma automática reglas de inversión que indiquen cuándo comprar o vender un activo financiero, en concreto acciones. Para ello, se ha seguido el esquema propuesto por Allen y Karjalainen en su artículo *Using Genetic Algorithms to find Technical Trading Rules* (1993) [1], en el que utilizan la programación genética para obtener reglas similares.

Las reglas están basadas en el análisis técnico para decidir el momento de operar. Una regla de inversión consiste en un árbol cuyos nodos constan de diversas funciones seleccionadas, entre las que se incluyen funciones para obtener el precio de una determinada acción en diferentes días y algunos de los indicadores utilizados en el análisis técnico. El tipo de funciones que forman los nodos se detallarán en el apartado 4.2.1. El árbol que forma cada una de las reglas, al ser evaluado, da como resultado un valor booleano, que indica si se debe estar dentro o fuera del activo evaluado, es decir, tenerlo en cartera o no. Para cada día de mercado, se debe de evaluar cada regla, ya que puede tener nodos cuyas funciones evalúan el precio actual de la acción (precio de cierre del día) y funciones que utilizan el precio de un número determinado de días anteriores al actual para calcular un valor (funciones de análisis técnico), lo cual provoca que el resultado de la evaluación del árbol cambie según el día en el cual se evalúe:

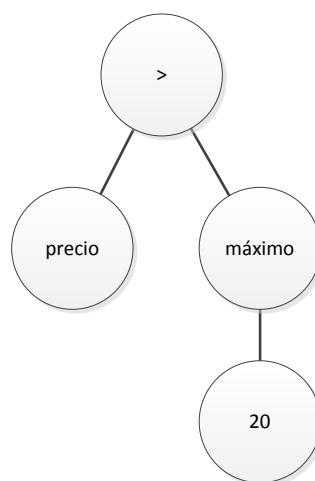


Figura 10: Ejemplo de una regla de inversión. En la figura, la regla mostrada indica estar dentro cuando el precio del día actual es mayor al máximo precio registrado en los 20 días anteriores al día actual.

La estrategia a seguir para invertir utilizando estas reglas es sencilla. Para cada día de mercado se evalúa una regla utilizando el histórico de los precios de la acción en cuestión como información de entrada para algunos de sus nodos (los que la necesiten). Hay cuatro situaciones posibles:

- Si no se poseía dicha acción en cartera y la regla indica estar dentro, entonces se procede a adquirirla.
- Si no se poseía dicha acción en cartera y la regla indica estar fuera, no se hace nada.
- Si ya se poseía la acción y la regla indica estar dentro, no se hace nada (se continúa con ella en cartera).
- Si ya se poseía la acción y la regla indica estar fuera, entonces se procede a venderla.

Los algoritmos genéticos y, por extensión, la programación genética, es técnica de inteligencia artificial adecuada para encontrar reglas de inversión que den beneficios. Como se ha comentado en el capítulo 3, no está demostrado que utilizando el análisis técnico se puedan obtener beneficios operando en los mercados financieros; es probable que en algunos períodos este tipo de análisis dé beneficios, pero en otros períodos puede no ser así. Como en el futuro no se puede saber si una regla dará los mismos resultados de beneficios que en el período utilizado para generarla, resulta complicado medir su calidad, y como consecuencia, la función de *fitness* solo puede dar un resultado aproximado de la calidad. Como se ha comentado en el capítulo 2, los algoritmos genéticos dan buenos resultados en este tipo de situaciones.

A pesar de este buen comportamiento de los algoritmos genéticos, es importante generar reglas que den beneficios con datos diferentes a los utilizados para generarlas. En otras palabras, las reglas deben estar generadas con el histórico de precios de una acción y deberían aprender los patrones que provocan las variaciones de precios observadas en el histórico de precios utilizado. Es de esperar que estos mismos patrones se repitan en el futuro, tal y como señala uno de los principios sobre los que se sustenta el análisis técnico, y que, por esta razón, las reglas anticipen cambios en el precio futuro de la acción.

Consecuentemente, hay que diseñar una estrategia que no sólo permita obtener beneficios con el histórico de precios utilizado para generar las reglas, sino que éstas también deben obtener beneficios con precios pertenecientes a otros períodos. Una estrategia adecuada es la propuesta por Allen y Karjalainen, la cual divide el histórico de precios en dos períodos contiguos pero no solapados [1]. El primero de ellos, llamado período de entrenamiento, es el que se utiliza para formar las reglas (los nodos que necesitan información sobre los precios, la extraen de este período), mientras que el segundo, llamado de validación, se utiliza para comprobar si la regla que genera mayores beneficios en el anterior período, también se comporta correctamente en el período actual (del cual la regla no dispone de ninguna información). Si este es el caso, entonces es presumible que la regla en cuestión haya detectado algún patrón común en ambos períodos que anticipa en la dirección adecuada cambios en el precio de la acción. Cuanto más extensos sean los períodos de entrenamiento y validación utilizados, es de esperar que sea más probable que estos patrones se repitan de nuevo en el futuro. La duración de estos dos períodos suele ser de 4 años para el de entrenamiento y de 2 años para el de

validación; de todos modos, la duración puede ser diferente, respetando que el de validación sea la mitad que el de entrenamiento. Para más información sobre esta estrategia de generación de las reglas, ver el apartado 4.3.

Para este proyecto ha sido adoptada la estrategia comentada en el párrafo anterior, ya que se utiliza en la mayoría de artículos que investigan la generación de reglas de inversión mediante algoritmos genéticos. Los pasos seguidos en esta estrategia para generar las reglas son los siguientes:

1. Crear la población inicial generando reglas de forma aleatoria hasta completar el número de individuos presentes en una población. Calcular el *fitness* de cada una de ellas en el periodo de entrenamiento.
2. Aplicar la mejor regla en el paso 1 al periodo de validación y calcular su *fitness* en éste. Guardar esta regla junto con su valor de *fitness* en el periodo de validación. Esta será la mejor regla inicial.
3. Escoger dos reglas de forma aleatoria, aunque favoreciendo a las que tengan mejor *fitness* en el periodo de entrenamiento (más detalles sobre la selección en el apartado 4.2.3).
4. Crear dos nuevas reglas cogiendo un subárbol aleatorio de cada una de las dos reglas seleccionadas en el paso 3 e intercambiándolos (operación de cruce).
5. Con una probabilidad dada, elegir un subárbol de manera aleatoria de cada una de las dos reglas obtenidas en el paso 4, y cambiarlos por sendos árboles generados aleatoriamente (mutación).
6. Calcular el *fitness* en el periodo de entrenamiento de las dos nuevas reglas obtenidas en el paso 5.
7. Reemplazar aleatoriamente una de las reglas escogidas en el paso 3, favoreciendo a aquella con peor *fitness* en el periodo de entrenamiento, por una de las reglas obtenidas en el paso 5, favoreciendo a aquella con mejor *fitness* en el periodo de entrenamiento (más detalles sobre la selección en el apartado 4.2.3).
8. Volver al paso 3 mientras el número de reglas que hayan sido escogidas sea menor que la probabilidad de cruce multiplicada por el número de reglas en la población.
9. Aplicar la mejor regla de la población en el periodo de entrenamiento al periodo de validación y calcular su *fitness*.
10. Si el *fitness* calculado en el paso 9 supera al de la mejor regla actual, guardar la nueva regla como la mejor.
11. Volver al paso 3 hasta que no se haya alcanzado el límite de generaciones o parar si no hay ninguna mejora en la mejor regla en un número determinado de generaciones.

El conjunto de pasos descritos se llama iteración. Al acabar una iteración, se guarda la mejor regla obtenida. Normalmente se llevan a cabo diversas iteraciones y se guarda la mejor regla obtenida en cada una. De esta forma se obtiene un conjunto amplio de reglas y el riesgo de que alguna no se comporte de forma adecuada queda oculto, ya que se puede comparar con el resto y observar el grado de consenso. De esta forma, el número de reglas que se obtienen es igual al número de iteraciones ejecutadas.

Es importante resaltar que los beneficios obtenidos siempre son relativos a los obtenidos por la estrategia *buy and hold* en el mismo periodo. La razón es que las reglas, cuya complejidad para obtenerlas es considerable, deben de aportar alguna ventaja con respecto a una estrategia de inversión pasiva y que prácticamente no supone ningún esfuerzo llevarla a cabo.

4.2 Detalles de la implementación

4.2.1 Tipos de nodos usados

Como se ha comentado en el apartado anterior, la decisión de qué funciones incluir en los nodos que forman las reglas es importante. De ello depende la información tenida en cuenta por las reglas y, consecuentemente, que los patrones observados en el histórico de precios puedan ser detectados.

Los precios que se utilizan en los nodos se encuentran normalizados con la media del precio de los 250 días anteriores a la jornada actual. La razón es que, con el tiempo, los precios de las acciones, como del resto de activos, tienden a elevarse al aumentar la cantidad de capital en circulación en la economía. De esta forma, los precios oscilan alrededor de la unidad y se elimina este efecto inflacionario.

Los nodos escogidos disponen de 0, 1, 2 o 3 parámetros. Un determinado nodo siempre tendrá un número fijo de éstos, es decir no existirán nodos con un número variable de parámetros. Los nodos sin ningún parámetro también son llamados hojas, ya que finalizan una rama del árbol.

A diferencia del método usado por Allen y Karjalainen en su artículo, en el cual no establecen ninguna limitación entre los tipos de datos que devuelve y acepta cada nodo, en la implementación hecha en este proyecto sí que limita cómo se pueden combinar diferentes nodos. En concreto se establecen tres tipos de datos: booleano, entero y decimal. Cada tipo de nodo devuelve un tipo de datos conocido de antemano, y acepta como parámetros valores pertenecientes a algún tipo de datos también conocido. De esta forma, los individuos forman árboles sintácticamente correctos que pueden ser interpretados de forma lógica. Esto también limita el espacio de búsqueda al eliminar combinaciones de nodos que no tendrían ningún sentido desde el punto de vista del análisis técnico ni de la lógica. Por ejemplo, una regla que indica estar dentro cuando el precio actual es mayor que cierto:

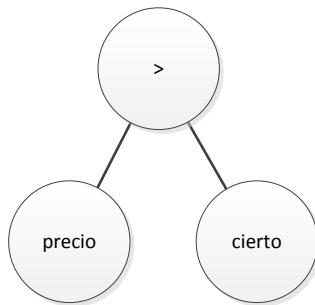


Figura 11: Ejemplo de una regla de inversión ambigua e incorrecta sintácticamente.

Sería ambigua ya que el valor devuelto por el nodo `>` podría ser cierto o falso cualquiera que fuese el valor del precio actual; todo dependería de la interpretación que se le diese a este caso. Crear árboles sintácticamente correctos elimina estas ambigüedades y da sentido lógico a las reglas creadas. El control de que los árboles que forman los individuos sean correctos se lleva a cabo en la inicialización de la población y al aplicar operadores en cada generación. El primer caso se comentará en el siguiente apartado; el último caso se comentará en el apartado 4.2.4.

A continuación se describen las funcionalidades implementadas en cada tipo de nodo y sus características:

- **Constante:** Es un nodo hoja. Representa a una constante, que puede ser de tipo booleano, entero o decimal. Una constante de tipo entero tiene un rango comprendido entre 0 y 250, y en la práctica representa a un número de días. Una constante de tipo decimal suele oscilar entorno a la unidad, y en la práctica representa a un precio.
- **Precio:** Es un nodo hoja. Representa al precio del día actual y por lo tanto devuelve un valor de tipo decimal. Este nodo necesita consultar el histórico de precios para dar su resultado.
- **AND, OR, NOT:** Los dos primeros tienen dos parámetros, mientras que el último tiene uno. Devuelven el resultado correspondiente a evaluar sus parámetros con la función que les da nombre. Sus parámetros deben de ser de tipo booleano.
- **Mayor, Menor, Igual:** Tienen dos parámetros. Devuelven un valor booleano resultado de aplicar la función que les da nombre a sus parámetros. El tipo de los parámetros puede ser decimal o entero, y, en el caso del nodo Igual, también booleano. El tipo de ambos parámetros ha de ser el mismo.
- **Suma, Resta:** Tienen dos parámetros. Ambos parámetros deben ser del mismo tipo: o bien enteros o bien decimales. Devuelven el resultado de aplicar la operación que da nombre al nodo a los dos parámetros. El tipo de valor devuelto es el mismo al que pertenecen sus parámetros. En caso de *overflow* por el límite superior, el nodo devuelve

el máximo valor válido (15,99998 para los decimales y 250 para los enteros); en caso de *overflow* por el límite inferior, el nodo devuelve el mínimo valor válido (-16,99998 para los decimales y 0 para los enteros).

- **If-Then-Else:** Es el único tipo de nodo que tiene tres parámetros. El primero de ellos siempre debe ser booleano y, dependiendo de su valor, el nodo devolverá como resultado su segundo (cierto) o su tercer (falso) parámetro. Estos dos últimos parámetros pueden ser de cualquier tipo, pero ambos deben de ser del mismo tipo.
- **Media Móvil, Máximo, Mínimo:** Tienen un parámetro, el cual debe de ser de tipo entero. Devuelven el resultado de aplicar la función que les da nombre sobre el número de días que indica su parámetro en el histórico de precios.

A causa de que los precios de cada jornada se encuentran normalizados con la media del precio de los 250 días anteriores al día actual, es necesario descartar las 250 primeras jornadas que aparecen en el histórico de precios, ya que son usadas para inicializar los datos. Adicionalmente, los nodos de tipo *Media Móvil*, *Máximo* y *Mínimo* pueden acceder a los datos de hasta 250 días previos a la jornada actual, con lo cual también hay que reservar este número de días para que puedan ser referenciados. Con lo cual, el histórico de precios ha de disponer de un mínimo de 501 jornadas de negocio para que el algoritmo pueda ser aplicado. Un año cuenta con alrededor de 250 días laborables, por lo tanto el histórico debe tener más de dos años de datos.

4.2.2 Representación e inicialización de los individuos

Es importante seleccionar un método compacto para guardar en memoria a los individuos y que permita acceder a su información de una forma rápida. De ello depende la velocidad con la que se ejecuta el algoritmo y la cantidad de memoria ocupada.

La representación de los individuos está basada en uno de los métodos estudiados en el artículo de Keith y Martin titulado *Genetic programming in C++: implementation issues* (1994) [5]. El método seleccionado es *virtual function tree*. Consiste en tener una clase abstracta "Nodo" y una clase concreta para cada tipo de nodo descrito en el apartado anterior. Estas clases concretas heredan de la abstracta "Nodo". En el artículo se compara la eficiencia de cinco métodos para representar los individuos en un algoritmo genético, tanto en tiempo de ejecución como en memoria ocupada. El método escogido es el segundo más rápido en tiempo de ejecución y el tercero en cantidad de memoria requerida. En la actualidad, la memoria resulta barata y las máquinas disponen de una gran cantidad de ella, con lo cual no es un factor importante en la elección del método. Consecuentemente, las razones por las que el método ha sido seleccionado, aparte del bajo tiempo de ejecución, son: su fácil entendimiento, la sencillez de su implementación y la capacidad de ampliación con más tipos de nodos. Estas ventajas son consecuencia de que dicho método explota el paradigma de la orientación a objetos, ya que trata a cada tipo de nodo como una clase diferente, con las mismas operaciones en todas ellas, deri-

vada de una clase común a todos los tipos que define la estructura (operaciones y atributos) que deben tener sus subclases.

Los individuos se guardan siguiendo el método *prefix* explicado en el mismo artículo. Este método consiste en tener para cada individuo un vector donde cada posición representa un nodo del árbol que forma el individuo. Cada nodo tiene su primer argumento en la posición siguiente, su segundo argumento justo donde acaba el subárbol encabezado por el primer argumento, el tercero donde acaba el subárbol del segundo argumento, y así sucesivamente:

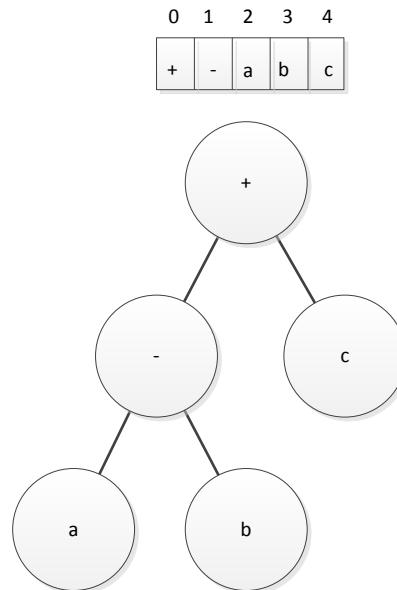


Figura 12: Representación de un árbol a partir de un vector con sus datos siguiendo el método "prefix".

Tal y como explica el artículo mencionado anteriormente, para determinar dónde acaba el subárbol que encabeza un determinado nodo, hay que recorrer el vector a partir de posición actual y sumar el número de argumentos de cada nodo menos uno. El subárbol terminará en el nodo en que la suma sea igual a -1. De esta forma, para evaluar el árbol de un individuo, hay que ir recorriendo secuencialmente el vector. Esto es lo que hace que esta implementación sea eficiente, ya que es más rápido acceder a la posición siguiente del vector que acceder a una posición aleatoria¹.

Para guardar la información de cada nodo, se ha decidido hacerlo en un campo de tipo entero. Este campo dispone de 32 bits en los cuales se guarda información diferente utilizando distintos bits del campo. Esta información es la siguiente: el valor de retorno del nodo y si dicho valor se ha de interpretar como booleano, entero o decimal. Inicialmente también se guardaba un dato más: la suma de los parámetros de cada nodo en posiciones anteriores del vector me-

¹ Un vector siempre tiene sus elementos en posiciones consecutivas de memoria, con lo cual es muy probable que, en un recorrido secuencial, la posición siguiente a acceder se encuentre en memoria cache y su tiempo de acceso sea prácticamente nulo.

nos uno, también llamado *stack count*. Sin embargo, este dato no es necesario para la ejecución del algoritmo, con lo cual, los bits ocupados por él pueden ser usados en el futuro para guardar otro tipo de información.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
reservado								tipo			valor																						
8 bits								3 bits			21 bits																						

Figura 13: Uso de los bits del único campo presente en cada nodo.

Si el nodo es de tipo booleano, entonces se interpretará cierto si la parte del valor tiene algo diferente de 0 y falso en caso contrario. En caso de ser entero, se interpretará el contenido de la parte del valor como un entero en complemento a dos, con lo cual puede representar números en el rango $+1.048.575 \dots -1.048.576$, que es suficiente para los que en la práctica contendrá (de 0 a 250). Por último, el valor de los nodos decimales será interpretado como un número en coma fija con 4 bits para la parte entera y 16 bits en la parte decimal, lo que da un rango de $+15,99998 \dots -16,99998$. La precisión es de $1,526 \cdot 10^{-5}$, que es suficiente para representar los precios normalizados con un error inferior a un 0,00153%. Adicionalmente, la representación de números decimales en coma fija tiene la ventaja de que las operaciones aritméticas son igual de rápidas que las realizadas con números enteros, ya que se operan de la misma forma.

En el tipo de valor de retorno se usan máscaras en vez de números enteros porque durante la inicialización de los individuos, en lugar del tipo de retorno, se guardan todos los tipos posibles que el nodo puede devolver. Esto es así porque la inicialización se hace mediante una función recursiva que necesita saber qué nodos hay disponibles para elegir en cada llamada y qué tipos pueden tener los nodos descendientes del actual. Una vez que el individuo está formado, ya se sabe con certeza qué tipo de valor va a devolver cada nodo, y estos bits pasan a indicar solo ese tipo. La cabecera de la función que inicializa cada nodo tiene el siguiente aspecto:

```
void inicializar(int nodos_disp, unsigned int posición, unsigned char& tipo_nodo);
```

Donde *nodos_disp* es el número máximo de nodos que puede tener el subárbol que encabeza el nodo a generar, *posición* es la posición que le corresponde al nodo a generar dentro del vector en el que se guardan los nodos, y *tipo_nodo* es un parámetro de entrada/salida e indica los tipos que puede devolver el nodo a generar y retorna el tipo de datos que finalmente devuelve el nodo a generar.

Es importante que cada nodo ocupe el mínimo espacio posible, ya que, el algoritmo se ejecuta con un número elevado de individuos (usando los parámetros recomendados, varios cientos), y cada uno de ellos estará formado por numerosos nodos (unos 50 de media). Así, además de ahorrar memoria para ejecutar el algoritmo, el tiempo de procesado disminuye, ya que los

datos de los individuos pueden estar en niveles de memoria cercanos, con lo cual es más rápido acceder a ellos².

Los individuos se inicializan nodo a nodo de forma recursiva y secuencialmente en el vector en el que se guardan. Esto quiere decir que primero se inicializa el nodo actual, después el primer argumento junto con el subárbol que encabeza, que se sitúa en el vector en la posición inmediatamente siguiente a la actual, después el segundo argumento, que comienza en la posición siguiente donde acaba el subárbol del primero, y así con el resto de argumentos que tenga el nodo actual. Primero se escoge el tipo de nodo del padre, y, según esta elección, se pasa a la función que genera los nodos qué tipos que pueden ser devueltos al padre. No obstante, la manera de inicializar los nodos difiere ligeramente según el tipo de nodo. A continuación se explican los detalles:

- **Constante:** Se genera un nodo constante del tipo requerido con un valor aleatorio. Según el tipo de nodo, los valores generados se encuentran en el siguiente rango:
 - Booleano: cierto o falso
 - Entero: 0 ... 250
 - Decimal: 0,5 ... 1,5
- Si el nodo a generar puede ser de diferentes tipos, se escoge uno de ellos y se genera una constante de ese tipo.
- **Precio:** Se genera un nodo de tipo decimal.
- **AND, OR, NOT:** El tipo de nodo ha de ser booleano, igual que sus descendientes. De esta forma se genera el nodo y después, al inicializar a sus descendientes, se les indica que han de ser de tipo booleano.
- **Mayor, Menor, Igual:** El tipo de nodo ha de ser booleano, aunque el tipo de los descendientes puede ser decimal o entero (en el caso del Igual, también booleano). Como el tipo de ambos descendientes ha de ser el mismo, se indica al primer descendiente los 2 (o 3) tipos que puede tener. Al acabar la inicialización del descendiente y del subárbol que encabeza, éste devuelve el tipo que finalmente ha escogido y entonces se inicializa el segundo descendiente indicándole que debe tener el tipo devuelto por el primero.
- **Suma, Resta, If-Then-Else:** El proceso es similar al caso anterior, aunque aquí el tipo del nodo lo determina el tipo pasado como argumento a la función que genera el nodo o, en caso de haber más de uno, el primer descendiente como ocurre en el punto an-

² Cuanto más cercano al procesador sea el nivel de memoria, más rápido es su acceso, aunque menos capacidad de almacenamiento tiene. Por ejemplo, los registros del procesador solo pueden guardar 4 bytes cada uno pero su tiempo de acceso es nulo (el dato se sirve en el mismo ciclo en el que se realiza una operación con él); la *cache* de nivel 1 tiene una capacidad de cientos de kilobytes y su tiempo de acceso está en 1 - 2 ciclos de procesador; la memoria principal puede guardar unos pocos gigabytes, con un tiempo de acceso de cientos de ciclos de procesador.

terior. En caso de que el primer descendiente determine el tipo de nodo, el tipo del nodo actual se conocerá una vez inicializado el primer descendiente y obtenido su tipo; en ese momento se establece el tipo del nodo actual y se inicializa el segundo descendiente con este tipo.

En el caso del *If-Then-Else*, el primer descendiente representa la condición y siempre tendrá tipo booleano, por lo que este método se aplica a su segundo y tercer descendientes.

- **Media Móvil, Máximo, Mínimo:** Estos nodos siempre serán de tipo decimal y con un único argumento de tipo entero.

La elección de un determinado nodo por parte de la función de inicialización se hace al azar, pero siempre escogiendo uno que pueda devolver alguno de los tipos indicados como parámetro y cuyo número de argumentos sea menor al número de posiciones disponibles en el vector. Para conseguir que los árboles generados sean completamente aleatorios y no tiendan a acumular demasiados nodos en los primeros subárboles creados, el número máximo de nodos para cada subárbol se pasa como parámetro a la función que genera los nodos. El número máximo de nodos asignados a cada descendiente es el número máximo de nodos asignados al nodo actual menos 1, dividido por el número de descendientes. De esta forma todos los descendientes tienen el mismo límite.

Para guardar los datos del histórico de precios de las acciones se utilizará una base de datos. Sin embargo, los nodos que necesitan acceder a los precios del histórico (*Máximo, Mínimo, Media Móvil, Precio*), tendrán disponibles todos los datos que puedan necesitar en memoria. Todo el histórico de precios de la acción a analizar se habrá copiado de la base de datos a memoria antes de aplicar el algoritmo, ya que realizar un acceso a la base de datos cada vez que se necesita un dato resultaría muy lento.

4.2.3 Evaluación

Los individuos se pueden evaluar durante cualquier número de días, siempre y cuando haya suficientes datos en el histórico de precios. El resultado de la evaluación es el beneficio absoluto durante ese período. A pesar de que el resultado de la evaluación debería de ser respecto a los resultados obtenidos por la estrategia *buy and hold* en el mismo periodo de tiempo, en la práctica no se lleva a cabo esta comparación durante la ejecución del algoritmo (periodos de entrenamiento y validación). La razón de esto es que para comparar el beneficio dado por diferentes reglas en el mismo periodo de tiempo, es suficiente con observar su beneficio absoluto, ya que el beneficio obtenido por *buy and hold* es una constante que se debería de restar a los beneficios de todas las reglas que se quieren comparar, resultando en un cálculo innecesario. Sin embargo, sí se resta el beneficio de *buy and hold* en los resultados obtenidos en los experimentos y en todas las estadísticas mostradas en el programa implementado para este proyecto, ya que el objetivo principal es batir a dicha estrategia.

Al evaluar las reglas, se tienen en cuenta los costes por cada transacción efectuada, es decir, los costes que implica vender o comprar acciones en la realidad. Concretamente, para evaluar una regla entre dos fechas dadas, se evalúa la regla con los datos del histórico de precios disponibles hasta el día actual. En caso de cambiar la posición (compra o venta), la transacción se hará efectiva al precio de apertura del día siguiente. En el primer día del periodo de evaluación, se evalúa la regla con los datos del histórico disponibles hasta el día anterior, y, si la regla indica comprar, el primer día de evaluación se compra al precio de apertura del día. Si en el último día del periodo de evaluación se está dentro del mercado, se vende al precio de apertura del día siguiente.

4.2.4 Operadores

En los siguientes apartados se comentará en detalle la implementación en el programa realizado para este proyecto de los operadores aplicados a los individuos de la población en cada generación. Cabe decir que solo se comentarán los que se aplican explícitamente y pueden ser ajustados: cruce y mutación.

Sin embargo, como se ha explicado en el capítulo 2, existe otro operador esencial en los algoritmos genéticos, conocido como reproducción y consistente en pasar sin cambios individuos a la siguiente generación. Éste también se aplica en el programa implementado, aunque no se pueda ajustar su probabilidad de forma explícita. Ella depende de la probabilidad con la que se aplica el operador de cruce. De esta forma, la probabilidad de aplicación del operador de reproducción es la siguiente:

$$p_{reproducción} = 1 - \frac{p_{cruce}}{2}$$

La fórmula viene dada porque la mitad de los individuos involucrados en la operación de cruce pasan sin cambios a la siguiente generación, tal y como se verá en el siguiente apartado. El resto de individuos de la población no afectados por la operación de cruce no sufren ningún cambio y pasan directamente a la siguiente generación.

4.2.4.1 Cruce

La operación de cruce intercambia dos subárboles de dos individuos y da lugar a dos nuevos individuos. Lo primero que se comprueba antes de realizar esta operación es que ambos individuos estén formados por más de un nodo; si los dos tienen un solo nodo, no tiene sentido realizar el intercambio, ya que daría lugar a dos individuos idénticos a sus progenitores. En segundo lugar, se comprueba que ambos sean compatibles, es decir, que sea posible intercambiar dos de sus subárboles teniendo en cuenta el tipo que devuelven sus nodos.

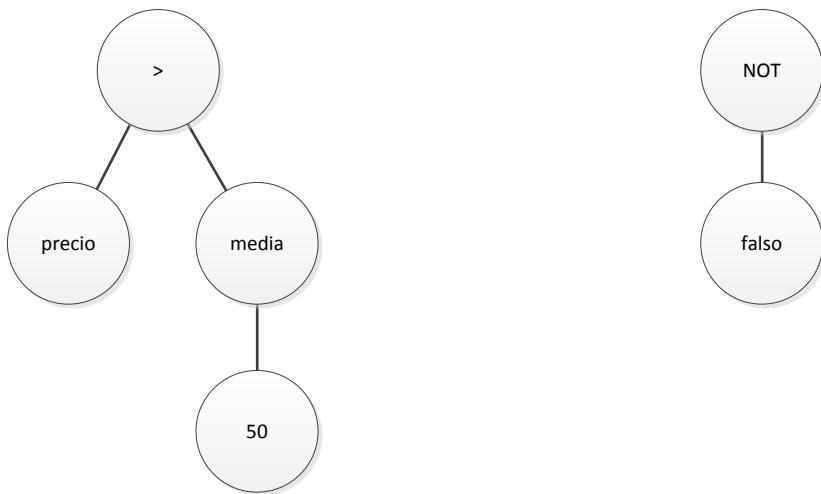


Figura 14: Ejemplo de dos individuos incompatibles.

Para comprobar si dos individuos son compatibles, hay que determinar si existe algún nodo de ambos individuos que devuelva el mismo tipo de datos. No se pueden intercambiar los nodos raíz de ambos individuos, ya que no cambiarían en nada los individuos generados. Para determinar su compatibilidad, primero se comprueba, para cada nodo que no sea raíz del primer individuo, si se puede intercambiar por cualquier otro nodo no raíz del segundo. Si no es posible, se comprueba si la raíz del primer individuo se puede intercambiar por otro nodo no raíz del segundo. Si los individuos continúan sin ser compatibles, se comprueba si la raíz del segundo individuo es intercambiable por cualquier otro nodo no raíz del primer individuo. Si aun así no existe ningún punto de intercambio entre ambos, se determina que son incompatibles y no pueden producir descendencia, con lo cual ambos pasan a la siguiente generación. En resumen, se comprueban todas las posibilidades para realizar el intercambio. Experimentalmente se ha comprobado que esta situación de incompatibilidad es infrecuente y no suele superar el 1% del número de individuos de la población.

Una vez asegurado que ambos individuos son compatibles, se escogen dos nodos al azar de sendos individuos como nodos que encabezarán los subárboles a intercambiar. Si en el primer individuo se escoge la raíz, en el segundo se prohíbe que pueda ser elegida. Si los dos nodos escogidos no son compatibles, se comprueba si el siguiente nodo (en el vector donde se guardan) del segundo individuo lo es con el escogido inicialmente en el primero. Si se han recorrido todos los nodos del segundo (de forma circular en el vector de nodos) y no hay ninguno compatible con el nodo seleccionado del primero, se comprueba con el siguiente nodo del primer individuo (recorriendo circularmente su vector de nodos) y se recorre de nuevo el segundo individuo buscando alguna compatibilidad.

Una vez realizada la operación de cruce se obtienen dos nuevos individuos. Consecuentemente, en ese momento la población cuenta con dos individuos más. Después de aplicar los operadores, la población ha de quedar con el mismo número de individuos que tenía anteriormente, y por ello, se han de descartar dos de los cuatro individuos involucrados en la operación. El descendiente con mayor *fitness* surgido de la operación de cruce realizada siempre remplazará

a uno de los progenitores. La elección de éste se hace teniendo en cuenta su *fitness*, aunque no siempre será remplazado el padre con menor *fitness*, sino que la probabilidad de que el segundo progenitor sea remplazado viene dada por la siguiente fórmula:

$$p_2 = \frac{\frac{\text{fitness}_1 - \text{fitness}_2}{\text{max_fitness} - \text{min_fitness}} + c}{2} + 1$$

Donde fitness_1 y fitness_2 son el *fitness* del primer y segundo progenitor respectivamente; *max_fitness* y *min_fitness* el mayor y menor *fitness* de la población; y *c* es una constante con un valor muy pequeño (en la práctica 0.00001) para prevenir una división entre 0 en caso de que en la población todos los individuos tengan el mismo *fitness* y para dar oportunidad (insignificante) a que, en caso de juntarse uno de los mejores individuos con uno de los peores, el peor de ellos pueda ser escogido. Lógicamente, la probabilidad de que sea el primer progenitor el remplazado es:

$$p_1 = 1 - p_2$$

Como se puede observar la fórmula tiene en cuenta el máximo y el mínimo *fitness* de la población, con lo cual, a medida que las generaciones van pasando y el *fitness* medio de los individuos aumenta, las probabilidades calculadas se ajustan a ello.

Selección

Para llevar a cabo la selección de los individuos a los que se aplicará el operador de cruce, se utilizará una estructura llamada *dis_aux*, que es un campo de la clase *Poblacion*. En el programa implementado, esta estructura es un vector de 16 posiciones; sin embargo, la longitud puede variar de acuerdo a la precisión deseada. Cada posición del vector representa a un conjunto de individuos con un *fitness* parecido (más parecido cuantas más posiciones tenga el vector). Los individuos de la población se clasifican en las distintas posiciones del vector según su *fitness*. Cada posición tiene un límite inferior y uno superior, y, todos los individuos que tengan su *fitness* entre esos dos límites, irán en esa posición. Ninguna posición del vector tiene el rango solapado con otra, ya que un individuo solo puede caer en una posición. La posición siguiente tendrá como límite inferior el límite superior de la posición actual, y la diferencia entre los dos límites es la misma para todas las posiciones:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
<i>dis_aux:</i>	1	0	0	2	11	43	4	16	9	20	33	0	5	0	2	4

Límite inferior: -50.00 -43.75 -37.50 -31.25 -25.00 -18.75 -12.50 -6.25 0.00 6.25 12.50 18.75 25.00 31.25 37.50 43.75

Fitness máximo: 50

Fitness mínimo: -50

Individuos en la población: 150

Figura 15: Esquema de la estructura utilizada para seleccionar los individuos que se cruzarán. Dentro del vector se muestra el número de individuos que caen en cada posición, según su "fitness".

Cada posición del vector contiene los siguientes campos:

```
float adj_fitness; //probabilidad de que la posición sea escogida
unsigned short num_ind; //número de individuos representados
vector<unsigned short> *pos_ind; //posición dentro del vector 'ind' de los individuos
```

Donde *num_ind* es el número de individuos que hay en esa posición y *pos_ind* guarda sus posiciones en el vector que guarda los individuos para poder localizarlos.

Todos los individuos que caen en la misma posición del vector, tienen la misma probabilidad de ser escogidos, la cual aumenta de forma lineal a medida que la posición aumenta. Por ejemplo, con un vector de 4 posiciones (en realidad se emplea uno de 16), las probabilidades que corresponden a cada posición son:

Posición	0	1	2	3
Probabilidad (%)	12.5	37.5	62.5	87.5

La probabilidad de escoger una posición del vector ha de ser mayor cuantos más individuos están representados en ella, por lo tanto, la probabilidad base, que se muestra en tabla anterior, se ha de multiplicar por el número de individuos representados, *num_ind*. La suma de probabilidades de todas las posiciones del vector ha de sumar 1, para que sea independiente del número de individuos de la población. De esta forma, para hallar la probabilidad que le corresponde a cada posición del vector, se ha de aplicar la siguiente fórmula, cuyo resultado se guarda en el campo *adj_fitness*:

$$\text{adj_fitness}_i = \frac{p_{\text{base}}_i \cdot \text{num_ind}_i}{\sum_{j=1}^{\text{num_pos}} p_{\text{base}}_j \cdot \text{num_ind}_j}$$

Donde *num_pos* es el número de posiciones del vector, *num_ind_i* el número de individuos que caen en la posición *i* del vector y *p_base_i* es la probabilidad base de la posición *i* del vector.

Para calcular la posición de cada individuo dentro del vector, se utiliza la siguiente formula:

$$\text{posición} = \left\lfloor \frac{\text{num_pos} \cdot (\text{fitness}_i - \text{min_fitness})}{\text{max_fitness} - \text{min_fitness}} \right\rfloor$$

Donde *num_pos* es el número de posiciones del vector, *fitness_i* es el *fitness* del individuo actual, *min_fitness* es el *fitness* del peor individuo de la población y *max_fitness* es el *fitness* del mejor individuo de la población.

De esta manera, se garantiza que en la primera y la última posición del vector siempre habrá algún individuo representado y que no influye cómo está distribuido el *fitness* de la población, ya que se ajusta a su distribución. Dicho de otro modo, este método logra acentuar las pequeñas diferencias en el *fitness* de los individuos en una población donde todos son similares. Como se ha comentado en el capítulo 2, esto también sucede en la naturaleza, con lo cual este método se ajusta bien a la realidad.

A pesar de haber una alta probabilidad de seleccionar al mejor individuo de la población de la manera descrita, puede no ser así. De todos modos, este individuo siempre será elegido, ya que este algoritmo sigue el denominado modo elitista.

En cualquier caso, siempre habrá un número par de individuos seleccionados, eliminando al individuo con menor *fitness* de entre los seleccionados en caso de ser impares. Una vez seleccionados a los individuos que se cruzarán, las parejas entre ellos se forman de manera aleatoria. El número de individuos seleccionados depende de la probabilidad escogida para aplicar el operador de cruce, y viene dado por la siguiente fórmula:

$$\# \text{individuos seleccionados} = p_{\text{cruce}} \cdot \# \text{individuos de la población}$$

Normalmente p_{cruce} es de un 90%. La fórmula mostrada implica que el número de individuos cruzados será siempre el mismo en todas las generaciones.

4.2.4.2 Mutación

La operación de mutación se realiza con una probabilidad determinada sobre el individuo descendiente de un cruce que remplaza a uno de sus progenitores. La probabilidad de realizar esta operación habitualmente es muy baja, no superior a un 5% (aunque de hecho, Koza [6] recomienda que para problemas genéricos no se aplique el operador).

Siempre se realiza justo después de obtener el nuevo individuo resultante de un cruce, y se aplica sobre este. Consiste en elegir un nodo al azar de dicho individuo y remplazar el subárbol que encabeza el nodo por otro generado aleatoriamente. La forma de generar el nuevo subárbol es la misma que la utilizada para generar los individuos de la población inicial, aunque con dos pequeñas diferencias: el nodo raíz del árbol generado debe de devolver el mismo tipo de datos que el nodo remplazado y el número de nodos disponibles para generar el árbol es menor, concretamente:

$$\# \text{nodos disponibles} = \# \text{máximo} - \# \text{nodos individuo} + \# \text{nodos subárbol reemplazado}$$

4.3 Sobreentrenamiento

Es probable que el mercado se comporte de forma diferente en periodos de tiempo distintos, es decir, que los patrones de análisis técnico que en un determinado periodo de tiempo dan beneficios, en otro periodo den pérdidas. De hecho, algunos autores que utilizan programación genética para generar reglas de inversión, [1] y [3], señalan que a medida que el tiempo avanza, las reglas pierden la efectividad que tenían en periodos cercanos a los utilizados para generarlas. Si esto resulta cierto, se puede deducir que el mercado también varía su comportamiento cada cierto periodo de tiempo de forma paulatina.

Si los patrones que permiten obtener beneficios en el mercado cambian cada cierto tiempo, no sería acertado utilizar un periodo de tiempo para generar reglas (entrenamiento) y otro para comprobar que dichas reglas funcionan (validación), ya que las reglas que se comportan bien

en uno, podrían no hacerlo en el otro. Volviendo al objetivo inicial de los algoritmos genéticos, el cual es buscar la mejor solución posible a un problema con datos relevantes al problema, no deberían de utilizarse otros datos ocultos (del periodo de validación) para guiar el algoritmo.

A pesar de lo comentado, utilizar solo un periodo para guiar la solución en este problema no daría buenos resultados si se aplican las reglas a otros periodos de tiempo distintos al utilizado para generarlas. Con un único periodo, cada generación transmite a la siguiente información privilegiada sobre cómo debería de comportarse para obtener los máximos beneficios en ese periodo, o dicho de otro modo, transmite en cierta forma cómo se comportará en un futuro el histórico de precios. Ésta es una información que no está disponible en la realidad en el momento de operar en cualquier mercado y, consecuentemente, las reglas generadas de este modo se encontrarían sobreentrenadas para ese periodo de tiempo. Por ello, las reglas deberían de aprender a dar beneficios cuando no disponen de información sobre la futura evolución de los precios.

Existe la posibilidad de que los patrones en los periodos de entrenamiento y validación sean diferentes; de todas formas probablemente guardarán características comunes si la duración de ambos periodos es la adecuada. La elección de esta duración es una decisión importante, ya que el elegir periodos muy extensos puede dar lugar a que los patrones que funcionan en uno no funcionen en el otro; por el contrario, si se eligen con una duración corta, la información proporcionada por ambas muestras podría no ser suficiente para detectar patrones que den buenos resultados en el futuro. En este sentido, la duración normalmente elegida por los autores es de cuatro años para el periodo de entrenamiento y de dos años para el de validación.

Este método es similar al *cross-validation*, utilizado en estadística para determinar si un conjunto de datos se puede ajustar a un modelo concreto. En este método se trabaja con dos muestras pertenecientes a la misma población. Este punto difiere ligeramente al histórico de precios de una acción en diferentes periodos de tiempo, el cual podría considerarse que es de una población más o menos diferente si los patrones que generan beneficios no lo hacen en los dos periodos. No obstante, es la mejor forma de aproximarse al *cross-validation*, ya que se trata de la misma acción y es de esperar que su comportamiento tenga similitudes en dos periodos de tiempo no demasiado lejanos uno de otro. Este es un motivo adicional por el cual los periodos de entrenamiento y validación deben tener la duración adecuada y ser consecutivos.

Referencias utilizadas en el capítulo: [1], [2], [3], [4], [5], [6].

PARTE 2: CONSTRUCCIÓN DE UNA APLICACIÓN PARA EXPERIMENTACIÓN

5. ANÁLISIS Y ESPECIFICACIÓN

En este capítulo se describirán las etapas de análisis de requisitos y la especificación del sistema software construido para este proyecto. También se detallará la metodología seguida durante la creación del software, que se incluirá dentro de la etapa de análisis.

En la etapa de análisis se concretarán las funcionalidades que deberá tener el sistema (requisitos funcionales) y los factores que se deberán tener en cuenta en la implementación de dichas funcionalidades (requisitos no funcionales). También se determinará el proceso de trabajo que se utilizará en la construcción del software, junto con las herramientas empleadas para generar los diagramas surgidos en dicho proceso. Los apartados dedicados a la etapa de análisis son: 5.1 y 5.2.

En la etapa de especificación se detallarán qué bloques componen la lógica del sistema y cómo se comportará externamente, es decir, cómo interactuará con los actores y las entradas y salidas de datos que producirá. En esta etapa no se tienen en cuenta las tecnologías empleadas en la implementación del sistema. Los apartados dedicados a la especificación son: 5.3, 5.4 y 5.5.

5.1 Metodología de trabajo

La metodología utilizada para diseñar el software es el modelo propuesto por Craig Larman. Este es un proceso de trabajo que establece que en cada etapa del flujo de trabajo (análisis, especificación, diseño e implementación) deben existir unos diagramas y documentos concretos que muestren claramente lo que ocurre en cada una. Sin embargo, en esta memoria no se mostrarán todos los documentos que propone el proceso de Larman, a causa de su gran número. Este proceso está orientado a documentar sistemas software muy grandes, e incluso en estos casos, algunos de los documentos de la metodología no son necesarios para comprender y poder modificar el sistema. Por este motivo, se mostrarán únicamente los documentos más importantes que permitan la comprensión del sistema software. Es importante no documentar en exceso, ya que en este caso existirán diferentes diagramas redundantes e innecesarios que añaden complejidad innecesaria y provocan confusión en la persona que intenta comprender el sistema.

El programa utilizado para construir los diagramas es *Umbrello*. Este es un programa de código abierto y licencia GLP para Linux, y tiene una gran cantidad de funcionalidades relacionadas con la ingeniería del software, entre ellas la realización de los diagramas mostrados en esta parte de la documentación.

5.2 Requisitos del sistema

Para determinar las funcionalidades que debe ofrecer el sistema software y qué factores tener en cuenta al implementarlas, se han llevado a cabo reuniones con el director de este proyecto, Argimiro Arratia, que ha desarrollado el papel de cliente final de la aplicación.

5.2.1 Funcionales

El programa implementado en este proyecto grosso modo debe de proporcionar una interfaz que permita ejecutar el algoritmo de programación genética (también implementado en el proyecto), gestionar los datos de los históricos de precios de los índices y acciones guardados en la base de datos, gestionar las reglas guardadas en la base de datos y realizar experimentos con las reglas generadas por el algoritmo. Estos experimentos deben permitir realizar el conjunto de experimentos descritos en la última parte de este proyecto, así como llevar cabo en un futuro otro tipo de experimentos con las reglas generadas, especialmente orientados al estudio de la influencia de los costes de transacción en las reglas. Por este motivo, la interfaz del programa debe proporcionar la mayor libertad posible al usuario para manejar reglas de inversión y observar sus características.

Del listado de requisitos funcionales se derivan los casos de uso, ya que ambos tratan las funcionalidades que debe tener el sistema. A continuación se concretan estas funcionalidades, agrupadas según los cuatro criterios generales citados en el párrafo anterior:

- Programación genética:
 - Generar un conjunto de reglas con unos determinados parámetros. Tal y como se ha explicado en el capítulo 4, una ejecución del algoritmo genera tantas reglas de inversión como iteraciones realice. Todas las reglas generadas en una ejecución del algoritmo se agruparán en un conjunto indivisible que las contendrá, al cual se llamará de ahora en adelante *conjunto de reglas de élite*.
 - Cambiar los parámetros usados para generar los conjuntos de reglas de élite.
- Gestión de datos de los históricos:
 - Consultar el histórico de precios de los índices y valores guardados.
 - Eliminar el histórico de precios de algún índice o valor.
 - Añadir o actualizar los datos del histórico de precios de los índices y valores guardados, automáticamente y manualmente:
 - Automático: Los datos se descargan de Internet para todos los valores e índices presentes en la base de datos.
 - Manual: El usuario elige un valor o índice presente en la base de datos y un fichero con todos los datos que deberá contener el histórico.
 - Introducir un nuevo índice en la base de datos, y dar la opción a introducir también valores pertenecientes a él.
 - Introducir un nuevo valor perteneciente a un índice existente en la base de datos.
- Gestión de los conjuntos de reglas:

- Cambiar el nombre de un conjunto de reglas.
- Quitar de la interfaz un conjunto de reglas
- Guardar un conjunto de reglas en la base de datos.
- Cargar un conjunto de reglas de la base de datos.
- Eliminar un conjunto de reglas de la base de datos.
- Crear un conjunto de reglas personalizado, en el que se pueden agregar individualmente reglas pertenecientes a conjuntos de élite. El propósito de tener estos conjuntos personalizados es dar libertad al usuario para que pueda mezclar diferentes reglas de inversión provenientes de distintos conjuntos de élite y poder observar el resultado que da su aplicación conjunta (los experimentos muestran información de cada regla de forma individual y del conjunto, tal y como se comentará en el siguiente punto).
- Añadir una regla a un conjunto personalizado.
- Eliminar una regla de un conjunto personalizado.
- Experimentación:
 - Obtener información sobre la generación de un conjunto de reglas. Para los conjuntos de reglas de élite, la información mostrada es la siguiente: índice, valor, fechas entre las que está comprendido el periodo de entrenamiento y el de validación, beneficio máximo, mínimo y medio de los periodos de entrenamiento y validación, número de iteraciones, número de individuos en la población, número máximo de generaciones, número de generaciones sin mejora, número máximo de nodos por individuo, probabilidad de cruce y mutación, costes por transacción. Aparte de estos parámetros, también se muestra cómo ha evolucionado el *fitness* medio de la población y del mejor individuo en los periodos de entrenamiento y validación durante la generación. En el caso de los conjuntos de reglas personalizados, la información mostrada es la misma, aunque los datos pertenecen a la población de élite con la que ha sido generada cada regla, a excepción del beneficio máximo, mínimo y medio, que son sustituidos por el beneficio dado por la regla en cuestión en los periodos de entrenamiento y validación. Todos los datos de beneficios son mostrados con respecto a los obtenidos mediante la estrategia *buy and hold*.
 - Mostrar los beneficios y las estadísticas generales³ de un conjunto de reglas durante un determinado periodo según diversos costes de transacción.
 - Mostrar los beneficios y las estadísticas generales de un conjunto de reglas durante un periodo y con un coste de transacción determinados.
 - Mostrar qué porcentaje de reglas de un conjunto se encuentran dentro del mercado para cada día de mercado de un periodo dado.

³ Media, desviación estándar, mediana, máximo y mínimo de los beneficios dados por el conjunto de reglas. También se incluyen los resultados de operar mediante el voto mayoritario del conjunto de reglas, es decir, estar dentro del mercado cuando más del 50% de ellas así lo indican y estar fuera del mercado en caso contrario.

5.2.2 No funcionales

Los requisitos no funcionales, también llamados atributos de calidad, definen aquellos factores en los que se debe de poner énfasis al desarrollar el sistema. Mientras que los requisitos funcionales determinan qué es lo que debe hacer el sistema, los no funcionales describen cómo debería ser el sistema. Algunos ejemplos de requisitos no funcionales son: seguridad, fiabilidad, tolerancia a fallos, escalabilidad, tiempo de respuesta, portabilidad o usabilidad.

Para el sistema construido en este proyecto se han considerado los siguientes requisitos no funcionales:

- Eficiencia: La ejecución del algoritmo de programación genética en un escenario real se realiza con una configuración de parámetros que implica un gran volumen de procesado y, consecuentemente, dicha ejecución puede llevar fácilmente varias horas en una máquina de gama media actual. El algoritmo es la parte más importante del proyecto, y, por este motivo, en su diseño se ha tenido en cuenta la eficiencia de su implementación.
- Facilidad de ampliación: El algoritmo tiene en cuenta que en un futuro puedan ser añadidos más tipos de nodos, ya que los que hay en la versión final son solo un subconjunto de los más importantes usados en el análisis técnico. Por ello, puede ser interesante en un futuro investigar con más tipos de nodos. Adicionalmente, en el resto del programa también se ha tenido en cuenta la facilidad de ampliación futura, posiblemente añadiendo más funcionalidades para realizar más tipos de experimentos.
- Estabilidad: Debido a que la ejecución del algoritmo puede ser larga y a que las reglas generadas no son guardadas en la base de datos hasta que dicha ejecución finaliza, es importante que el programa no finalice inesperadamente mientras dure la ejecución del algoritmo. Este es el principal motivo por el cual considerar la estabilidad del programa.
- Facilidad de aprendizaje: Se ha considerado que la interfaz gráfica del programa tiene que ser intuitiva, ya que el grupo de usuarios a los que está dirigido el sistema, inversores e investigadores en finanzas, no tienen por qué estar familiarizados con la informática y no desean complicaciones innecesarias en los datos mostrados. Esta interfaz intuitiva posibilita que los usuarios aprendan rápidamente a usar el programa correctamente.

5.3 Modelo de casos de uso

El modelo de casos de uso detalla con qué entidades interactúa el sistema, sean usuarios o componentes externos inanimados, y qué funcionalidades puede llevar a cabo cada entidad. Estas funcionalidades se obtienen a partir de los requisitos funcionales explicados en el apartado 5.2.1.

5.3.1 Actores

Existirán tres actores en el sistema: usuario genérico, usuario investigador y servidor de datos:

- Usuario genérico: Representa a un inversor cuyo objetivo es utilizar el sistema para obtener conjuntos de reglas y utilizarlas para tomar sus decisiones de inversión. Utiliza todas las funcionalidades del sistema a excepción de aquellas que tienen que ver con la realización de experimentos.
- Usuario investigador: Es un tipo de usuario que pretende utilizar el sistema para realizar experimentos con las reglas generadas. Tiene acceso a todas las funcionalidades que permite el sistema.
- Servidor de datos: El sistema necesita tener acceso a los datos de los históricos de precios de los índices y valores para poder actualizar su base de datos. Este actor representa a un servidor que posee datos actualizados y que permite su acceso a través de Internet.

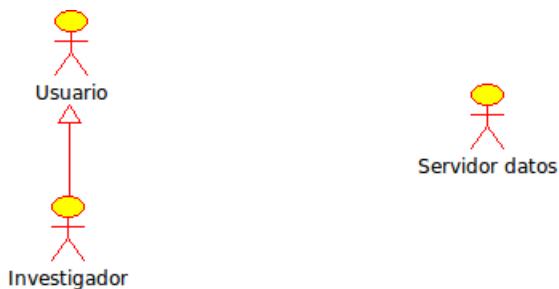


Figura 16: Diagrama de actores participantes en el sistema.

5.3.2 Diagramas

Los diagramas de los casos de uso muestran la relación que existe entre los actores y los casos de uso. Del mismo modo que se ha hecho con los requisitos funcionales, los diagramas de los casos de uso se han separado según el tipo de funcionalidad para ofrecer mayor claridad. También se han sacado comportamientos comunes dentro de algunos casos de uso, identificables porque tienen una flecha con el texto <<include>> apuntando hacia ellos. El motivo es que en el siguiente apartado, en el cual se describe paso a paso cada caso de uso, los comportamientos comunes se describirán una sola vez y no en cada caso de uso que los utiliza, con lo cual harán más clara la comprensión y más corta la descripción global de los casos de uso.

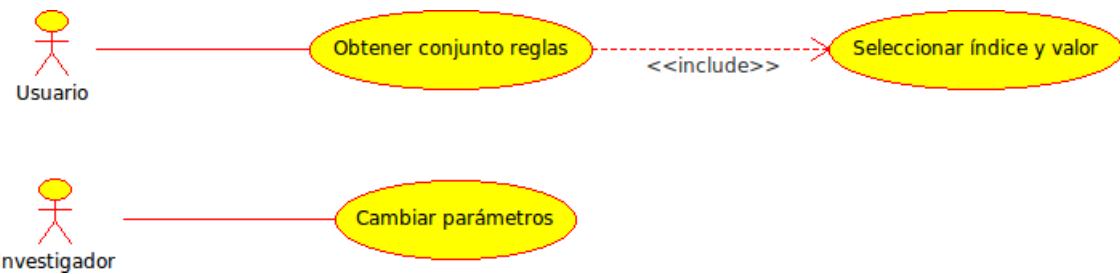


Figura 17: Casos de uso relacionados con la programación genética.

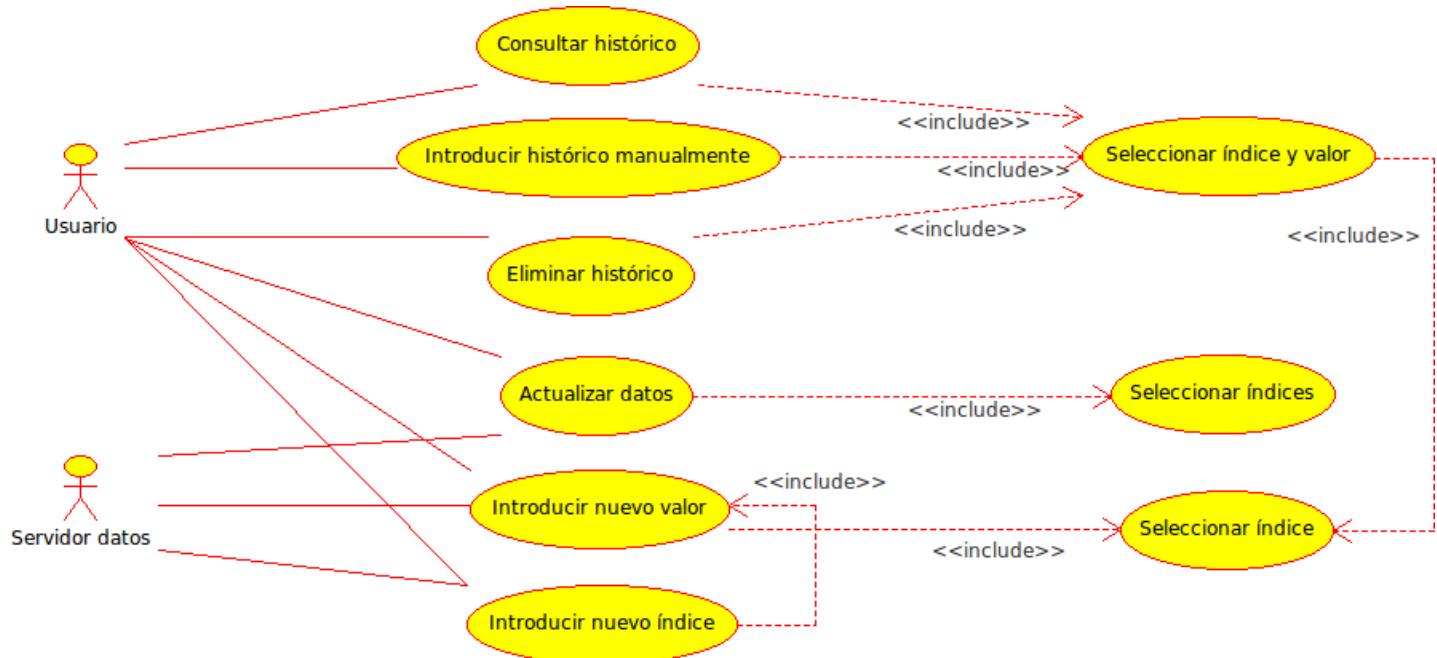


Figura 18: Casos de uso para manejar los datos de los históricos de precios.

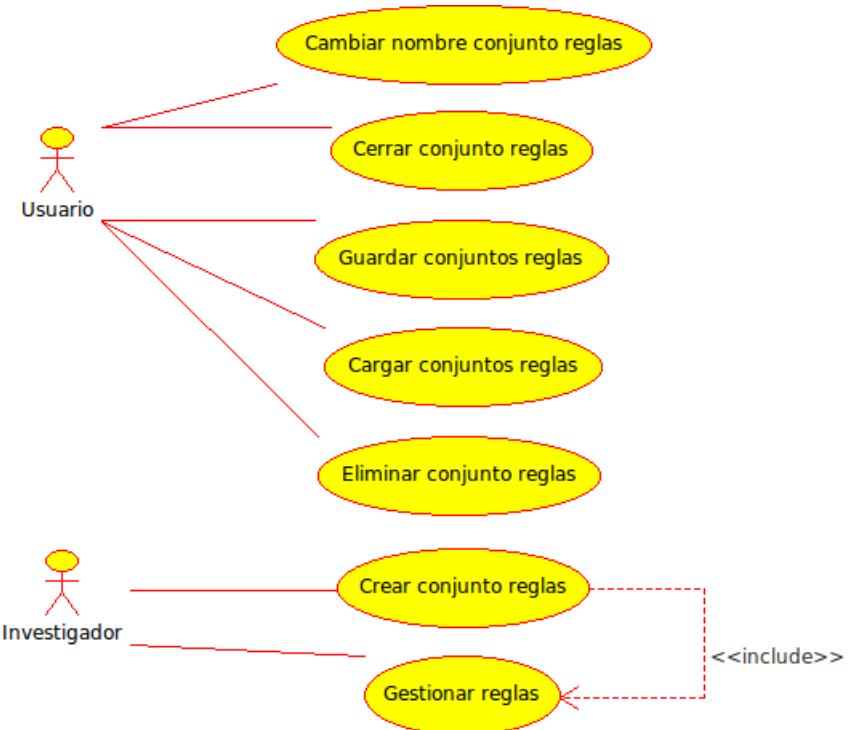


Figura 19: Casos de uso para manejar conjuntos de reglas.

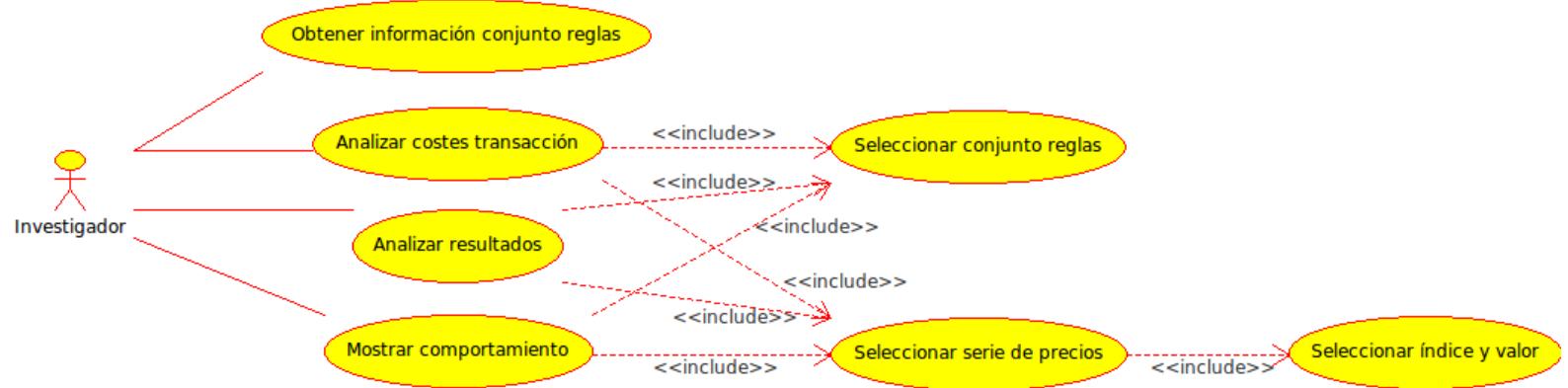


Figura 20: Casos de uso relacionados con la experimentación con las reglas.

5.3.3 Descripción

A continuación se detallan las características de cada caso de uso mostrado en los diagramas del apartado anterior. Concretamente estas características son: breve descripción, actores participantes, condiciones necesarias para poder iniciar el caso de uso (precondiciones), cambios producidos en el sistema al acabar el caso de uso (postcondiciones), flujo normal de acontecimientos y flujos alternativos (errores).

5.3.3.1 Obtener conjunto de reglas

Descripción

Permite generar conjuntos de reglas tomando los parámetros por defecto, aunque el usuario puede ajustar los costes de transacción usados en el proceso.

Actores

Usuario.

Precondiciones

- 1. El valor con el que se quiere obtener el conjunto de reglas está actualizado.**

Postcondiciones

- 1. El sistema obtiene un conjunto de reglas aplicando el algoritmo de programación genética sobre el histórico de precios del valor especificado.**
- 2. El sistema agrega el conjunto de reglas obtenido a la sesión actual.**
- 3. La sesión actual no dispone de dos conjuntos de reglas con el mismo nombre.**

Curso típico de acontecimientos

<p>1. El usuario pide al sistema obtener conjuntos de reglas.</p> <p>4. El usuario rellena el formulario con los costes de transacción deseados.</p>	<p>2. Ejecutar Seleccionar índice y valor.</p> <p>3. El sistema muestra un formulario con un campo para introducir los costes de transacción que se utilizarán en la ejecución del algoritmo.</p> <p>5. El sistema genera un conjunto de reglas con los parámetros por defecto (especificados en Sesión), los costes de transacción indicados en el paso 4 y el valor seleccionado en el paso 2.</p> <p>6. El sistema añade a la sesión actual el conjunto de reglas obtenido en el paso 5 con un nombre diferente a cualquier otro conjunto de reglas que ya haya en la sesión.</p> <p>7. El caso de uso acaba.</p>
Cursos alternativos	-

<i>5.3.3.2 Cambiar parámetros</i>	
Descripción	Permite a un usuario investigador cambiar los parámetros de la sesión actual usados, por el algoritmo de programación genética para generar conjuntos de reglas.
Actores	
Investigador.	
Precondiciones	
-	
Postcondiciones	

<p>1. El sistema modifica los parámetros de la sesión.</p>	
Curso típico de acontecimientos	
<p>1. El usuario pide al sistema cambiar parámetros.</p>	<p>2. El sistema muestra un formulario con los siguientes campos:</p> <ul style="list-style-type: none"> • Número de generaciones. • Tamaño de la población. • Número máximo de nodos de cada regla. • Probabilidad de cruce. • Probabilidad de mutación. • Número de iteraciones a ejecutar. • Escoger los periodos de entrenamiento y validación de dos formas: <ul style="list-style-type: none"> Opción 1: • Especificar fechas iniciales y finales de los períodos. Opción 2: • Longitud total en días de los períodos. En este caso, el periodo de entrenamiento y validación serán consecutivos, el segundo posterior al primero, y relativos al día actual.
<p>3. El usuario rellena los campos del formulario que deseé modificar.</p>	<p>4. El sistema modifica parámetros indicados en los campos que han sido llenados por el usuario en el paso anterior.</p> <p>5. El caso de uso acaba.</p>
Cursos alternativos	
-	

<i>5.3.3.3 Consultar histórico</i>
Descripción

<p>Permite a un usuario consultar los datos de un valor entre una fecha de inicio y de fin. Los datos mostrados para cada día de cotización del valor entre las fechas indicadas serán los siguientes: fecha, apertura, máximo, mínimo, cierre, volumen y cierre ajustado.</p>		
Actores		
Usuario.		
Precondiciones		
<ol style="list-style-type: none"> 1. El valor deseado ha de existir en la base de datos. 		
Postcondiciones		
-		
Curso típico de acontecimientos		
<ol style="list-style-type: none"> 1. El usuario pide al sistema consultar histórico. 	<ol style="list-style-type: none"> 2. Ejecutar Seleccionar índice y valor. 3. El sistema muestra un formulario con dos campos para escoger la fecha de inicio y de final. 4. El usuario introduce una fecha de inicio y una fecha de fin. 5. El sistema muestra todos los datos disponibles del valor seleccionado en el paso 2 entre las fechas de inicio y final seleccionadas en el paso 4, ambas inclusive. 6. El caso de uso acaba. 	
Cursos alternativos		
-		

<p><i>5.3.3.4 Introducir histórico manualmente</i></p>
Descripción
<p>Permite actualizar los datos de un valor a partir de un fichero en formato <i>comma separated values (csv)</i>. Este fichero ha de contener en cada línea, excepto en la primera, los siguientes datos: fecha (AAAA/MM/DD), apertura, máximo, mínimo, cierre, volumen, cierre ajustado.</p>

Actores	
Usuario.	
Precondiciones	
1. El valor deseado ha de existir en la base de datos.	
Postcondiciones	
1. El sistema guarda los datos del fichero leído en la base de datos como pertenecientes al valor deseado. Si para ese valor y una fecha del fichero ya existe un registro en la base de datos, se sobrescriben los datos de ese registro con los indicados en el fichero.	
Curso típico de acontecimientos	
1. El usuario pide al sistema introducir histórico.	2. Ejecutar Seleccionar índice y valor.
4. El usuario introduce la ruta al fichero.	3. El sistema muestra un formulario con un campo para introducir la ruta al fichero que contiene los datos.
	5. El sistema actualiza los datos del valor seleccionado en el paso 2 con los datos del fichero que se encuentra en el lugar especificado en el paso 4.
	6. El sistema muestra el número de registros actualizados en la base de datos.
	7. El caso de uso acaba.
Cursos alternativos	
Paso 5: El fichero tiene un formato incorrecto, no existe o no se puede abrir o leer:	
	1. El sistema notifica el error y no actualiza la base de datos.
	2. El curso continúa en el paso 4.

<i>5.3.3.5 Eliminar histórico</i>
Descripción

Permite eliminar un valor de la base de datos y todos sus datos.	
Actores	
Usuario.	
Precondiciones	
1. El valor deseado ha de existir en la base de datos.	
Postcondiciones	
1. El sistema elimina todos los datos del valor seleccionado de la base de datos.	
Curso típico de acontecimientos	
<p>1. El usuario pide al sistema eliminar un valor.</p>	<p>2. Ejecutar Seleccionar índice y valor.</p> <p>3. El sistema borra de la base de datos todos los datos pertenecientes al valor seleccionado en el paso 2.</p> <p>4. El caso de uso acaba</p>
Cursos alternativos	
-	

5.3.3.6 Actualizar datos
Descripción
Permite actualizar la información de todos los valores existentes en la base de datos accediendo a un servidor de datos y descargando los históricos de precios.
Actores
Usuario, servidor de datos.
Precondiciones
-
Postcondiciones
1. El sistema actualiza todos los valores de la base de datos con los disponibles en el servidor desde la fecha del registro más reciente en la base de datos hasta la fecha actual.

Curso típico de acontecimientos	
1. El usuario pide al sistema actualizar datos.	<p>2. Ejecutar Seleccionar índices.</p> <p>3. El sistema accede al servidor de datos y descarga, para cada valor de cada índice seleccionado en el paso anterior, un fichero csv con los datos un formato determinado. Con este fichero se actualiza el valor al cual pertenece en la base de datos. Solo se actualizan los valores que ya existen en la base de datos.</p> <p>4. El sistema muestra un resumen con el número de valores actualizados para cada índice.</p> <p>5. El caso de uso acaba.</p>
Cursos alternativos	
Paso 3: No hay conexión a Internet o el servidor de datos no está disponible:	
2. El usuario realiza las acciones necesarias para solucionar el problema.	<p>1. El sistema notifica el error.</p> <p>3. El curso continúa en el paso 3.</p>
Paso 3: Un fichero descargado no tiene el formato adecuado o no existe ningún fichero en el servidor para el valor solicitado entre su fecha más reciente en la base de datos y la fecha actual:	
	<p>1. El sistema no actualiza el valor actual.</p> <p>2. El curso continúa en el paso 3 con el siguiente valor a actualizar.</p>

5.3.3.7 Introducir nuevo valor
Descripción
Permite introducir en la base de datos un nuevo valor, y que, a continuación, sea actualizado con los datos más recientes en el servidor de datos.
Actores

Usuario, servidor de datos.	
Precondiciones	
1. El índice al que pertenece el valor a introducir ya existe en la base de datos.	
Postcondiciones	
1. El sistema crea el valor en la base de datos y lo clasifica como perteneciente a un determinado índice. 2. El sistema actualiza el valor con todos los datos históricos disponibles en el servidor para ese valor.	
Curso típico de acontecimientos	<p>1. El usuario pide al sistema introducir nuevo valor.</p> <p>2. Ejecutar Seleccionar índice.</p> <p>3. El sistema muestra un formulario para introducir el identificador del nuevo valor.</p> <p>4. El usuario introduce el identificador (<i>ticker</i>) del nuevo valor, tal como aparece en el servidor de datos.</p> <p>4. El sistema comprueba que el identificador introducido en el paso anterior no forma parte del índice seleccionado en el paso 2.</p> <p>5. El sistema accede al servidor y descarga todos los datos del nuevo valor en un fichero csv con un formato determinado.</p> <p>6. El sistema guarda en la base de datos los datos del fichero descargado en paso anterior como pertenecientes al valor e índice seleccionados.</p> <p>7. El sistema muestra un resumen con el número de registros (días de cotización) guardados en la base de datos.</p> <p>8. El caso de uso acaba.</p>
Cursos alternativos	
Paso 5: Ya hay un valor con el mismo identificador en el índice:	
	1. El sistema notifica el error.

	2. El curso continúa en el paso 4.
Paso 6: No hay conexión a Internet o el servidor no está disponible:	
2. El usuario realiza las acciones necesarias para solucionar el problema.	<ol style="list-style-type: none"> 1. El sistema notifica el error. 3. El curso continúa en el paso 6.
Paso 6: Un fichero descargado no tiene el formato adecuado o no existe ningún fichero en el servidor para el valor en cuestión:	
	<ol style="list-style-type: none"> 1. El sistema notifica que el valor introducido no existe. 2. El curso continúa en el paso 4.

<i>5.3.3.8 Introducir nuevo índice</i>
Descripción
Permite introducir en la base de datos un nuevo índice y a continuación sus componentes, que serán actualizados con los datos más recientes disponibles en el servidor.
Actores
Usuario, servidor de datos.
Precondiciones
-
Postcondiciones
<ol style="list-style-type: none"> 1. El sistema crea el índice en la base de datos. 2. El sistema añade los valores especificados al índice creado. 3. El sistema actualiza el índice y los valores que le han sido añadidos con todos los datos históricos disponibles en el servidor.
Curso típico de acontecimientos

<ol style="list-style-type: none"> 1. El usuario pide al sistema crear un nuevo índice. 3. El usuario introduce el identificador (<i>ticker</i>) del nuevo índice, tal como aparece en el servidor de datos. 	<ol style="list-style-type: none"> 2. El sistema muestra el formulario para crear un índice. 4. El sistema comprueba que el índice introducido en el paso 3 no existe ya en la base de datos. 5. El sistema accede al servidor y descarga todos los datos del nuevo índice en un fichero csv con un formato determinado. 6. El sistema guarda los datos del fichero descargado en el paso anterior en la base de datos como pertenecientes al nuevo índice. Para ello se crea un nuevo valor con el mismo nombre que el índice que contendrá los datos de dicho índice. 7. Se ejecuta <i>Introducir nuevo valor</i> a partir del paso 3. 8. Volver al paso 7 mientras haya valores por añadir. 9. El caso de uso acaba.
Cursos alternativos	
Paso 4: Ya hay un índice con el mismo identificador en la base de datos:	
	<ol style="list-style-type: none"> 1. El sistema notifica el error. 2. El curso continúa en el paso 3.
Paso 5: No hay conexión a Internet o el servidor no está disponible:	
<ol style="list-style-type: none"> 2. El usuario realiza las acciones necesarias para solucionar el problema. 	<ol style="list-style-type: none"> 1. El sistema notifica el error. 3. El curso continúa en el paso 5.
Paso 5: El fichero descargado no tiene el formato adecuado o no existe ningún fichero en el servidor para el índice en cuestión:	
	<ol style="list-style-type: none"> 1. El sistema notifica que el índice introducido no existe. 2. El curso continúa en el paso 3.

<p><i>5.3.3.9 Cambiar nombre conjunto de reglas</i></p>	
Descripción	<p>Permite cambiar el nombre de un conjunto de reglas de la sesión actual.</p>
Actores	<p>Usuario.</p>
Precondiciones	<ol style="list-style-type: none"> Hay algún conjunto de reglas en la sesión actual.
Postcondiciones	<ol style="list-style-type: none"> Cambia el nombre de un conjunto de reglas de la sesión actual. En la sesión actual no hay dos conjuntos de reglas con el mismo nombre.
Curso típico de acontecimientos	<ol style="list-style-type: none"> El usuario selecciona algún conjunto de reglas de la sesión actual. El usuario pide al sistema cambiar el nombre del conjunto de reglas seleccionado en el paso anterior. El sistema muestra un formulario con un campo para introducir el nuevo nombre del conjunto de reglas. El usuario introduce el nuevo nombre. El sistema cambia el nombre del conjunto de reglas por el introducido en el paso anterior. El caso de uso acaba.
Cursos alternativos	<p>Paso 4: Ya hay un conjunto de reglas con el mismo nombre en la sesión actual:</p>
	<ol style="list-style-type: none"> El sistema notifica la situación. El curso continúa en el paso 3.

5.3.3.10	<i>Cerrar un conjunto de reglas</i>
Descripción	Permite quitar un conjunto de reglas de la sesión actual.
Actores	Usuario.
Precondiciones	<ol style="list-style-type: none"> Hay algún conjunto de reglas en la sesión actual.
Postcondiciones	<ol style="list-style-type: none"> Se elimina de la sesión actual un conjunto de reglas.
Curso típico de acontecimientos	<ol style="list-style-type: none"> El usuario selecciona algún conjunto de reglas de la sesión actual. El usuario pide al sistema cerrar el conjunto de reglas seleccionado en el paso anterior. El sistema elimina de la sesión actual el conjunto de reglas seleccionado. El caso de uso acaba.
Cursos alternativos	
<p>Paso 2: El conjunto de reglas no fue guardado previamente en la base de datos:</p>	
<ol style="list-style-type: none"> El usuario responde a la pregunta formulada. 	<ol style="list-style-type: none"> El sistema notifica la situación y pregunta si guardar el conjunto de reglas. Si el usuario responde afirmativamente, el conjunto de reglas se guarda en la base de datos. El curso continúa en el paso 3.

5.3.3.11	<i>Guardar conjuntos de reglas</i>
Descripción	Permite guardar en la base de datos los conjuntos de reglas de la sesión actual.
Actores	Usuario.
Precondiciones	-
Postcondiciones	<p>1. El sistema guarda en la base de datos los conjuntos de reglas de la sesión que no existen en ella o bien han sido modificados.</p>
Curso típico de acontecimientos	<p>1. El usuario pide al sistema guardar los conjuntos de reglas.</p> <p>2. El sistema guarda en la base de datos todos los conjuntos de reglas presentes en la sesión que han sido modificados o creados.</p> <p>3. El caso de uso acaba.</p>
Cursos alternativos	<p>Paso 2: Ya existe en la base de datos algún conjunto de reglas con el mismo nombre que algún otro que se intenta guardar:</p> <p>2. El usuario responde a la pregunta formulada.</p> <p>1. El sistema notifica la situación y pregunta si se desea reemplazar uno de los conjuntos de reglas existente en la base de datos.</p> <p>3. Si el usuario responde afirmativamente, se reemplaza el conjunto de reglas por el que se ha preguntado en el paso 1.</p> <p>4. Volver al paso 1 mientras haya conflictos de nombres.</p>

	5. El curso continúa en el paso 2.
--	------------------------------------

5.3.3.12	<i>Cargar conjuntos de reglas</i>				
Descripción					
Permite cargar en la sesión actual uno o más conjuntos de reglas.					
Actores					
Usuario.					
Precondiciones					
-					
Postcondiciones	<ol style="list-style-type: none"> 1. La sesión dispone de los conjuntos de reglas cargados. 2. La sesión no tiene ningún conjunto de reglas con el mismo nombre. 				
Curso típico de acontecimientos	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="vertical-align: top; width: 50%;"> <ol style="list-style-type: none"> 1. El usuario pide al sistema cargar conjuntos de reglas. </td> <td style="vertical-align: top; width: 50%;"> <ol style="list-style-type: none"> 2. El sistema accede a la base de datos y obtiene todos los conjuntos de reglas previamente guardados. 3. El sistema muestra los conjuntos de reglas obtenidos en el paso 2 junto con opciones de filtrado de los conjuntos mostrados según el índice y el valor con los que fueron generados: carga un menú con todos los índices disponibles en la base de datos. </td> </tr> <tr> <td style="vertical-align: top;"> <ol style="list-style-type: none"> 4. El usuario escoge uno o más conjuntos de reglas mostrados. </td> <td style="vertical-align: top;"> <ol style="list-style-type: none"> 5. El sistema añade a la sesión actual los conjuntos de reglas seleccionados en el paso 4. 6. El caso de uso acaba. </td> </tr> </table>	<ol style="list-style-type: none"> 1. El usuario pide al sistema cargar conjuntos de reglas. 	<ol style="list-style-type: none"> 2. El sistema accede a la base de datos y obtiene todos los conjuntos de reglas previamente guardados. 3. El sistema muestra los conjuntos de reglas obtenidos en el paso 2 junto con opciones de filtrado de los conjuntos mostrados según el índice y el valor con los que fueron generados: carga un menú con todos los índices disponibles en la base de datos. 	<ol style="list-style-type: none"> 4. El usuario escoge uno o más conjuntos de reglas mostrados. 	<ol style="list-style-type: none"> 5. El sistema añade a la sesión actual los conjuntos de reglas seleccionados en el paso 4. 6. El caso de uso acaba.
<ol style="list-style-type: none"> 1. El usuario pide al sistema cargar conjuntos de reglas. 	<ol style="list-style-type: none"> 2. El sistema accede a la base de datos y obtiene todos los conjuntos de reglas previamente guardados. 3. El sistema muestra los conjuntos de reglas obtenidos en el paso 2 junto con opciones de filtrado de los conjuntos mostrados según el índice y el valor con los que fueron generados: carga un menú con todos los índices disponibles en la base de datos. 				
<ol style="list-style-type: none"> 4. El usuario escoge uno o más conjuntos de reglas mostrados. 	<ol style="list-style-type: none"> 5. El sistema añade a la sesión actual los conjuntos de reglas seleccionados en el paso 4. 6. El caso de uso acaba. 				
Cursos alternativos					
Paso 4: El usuario desea filtrar las reglas que el sistema muestra:					

<p>1. El usuario selecciona un índice</p>	<p>2. El sistema muestra los conjuntos de reglas que pertenecen solo al índice seleccionado.</p> <p>3. El sistema carga una lista con los valores del índice seleccionado que existen en la base de datos.</p>
<i>Si el usuario desea filtrar las reglas también por el valor con el que fueron generadas:</i>	
<p>4. El usuario selecciona un valor.</p>	<p>5. El sistema muestra los conjuntos de reglas que pertenecen solo al valor seleccionado.</p>
<i>Fin de la condición</i>	
<p>6. El usuario escoge uno o más conjuntos de reglas mostrados.</p>	<p>7. El curso continúa en el paso 5.</p>
<i>Paso 5: En la sesión actual ya existe algún conjunto de reglas con el mismo nombre que alguno de los seleccionados para ser cargado:</i>	
<p>2. El usuario responde a la pregunta del paso anterior.</p>	<p>1. El sistema notifica la situación y pregunta renombrar los conjuntos existentes en la sesión que tengan el mismo nombre que los que se van a cargar.</p>
<i>Si la respuesta del paso 2 es afirmativa:</i>	
	<p>3. El sistema renombra todos los conjuntos de reglas de la sesión actual que presenten conflictos con los que van a ser cargados.</p> <p>4. El curso continúa en el paso 5.</p>
<i>Si la respuesta del paso 2 es negativa:</i>	
	<p>5. El sistema añade a la sesión actual los conjuntos de reglas seleccionados en el paso 4 del curso principal que no tengan el mismo nombre que los que ya hay en la sesión actual.</p> <p>6. El curso continúa en el paso 6.</p>

<i>5.3.3.13</i>	<i>Eliminar un conjunto de reglas</i>
Descripción	Permite eliminar un conjunto de reglas de la base de datos del sistema.
Actores	Usuario.
Precondiciones	<ol style="list-style-type: none"> En la sesión actual hay algún conjunto de reglas.
Postcondiciones	<ol style="list-style-type: none"> El sistema elimina de la sesión actual un conjunto de reglas, y, si se hubiese guardado previamente en la base de datos, también lo elimina de ella.
Curso típico de acontecimientos	<ol style="list-style-type: none"> El usuario selecciona un conjunto de reglas de la sesión actual. El usuario pide al sistema eliminar el conjunto de reglas seleccionado en el paso 1. El sistema elimina de la sesión y de la base de datos, en caso de que exista en ella, el conjunto de reglas seleccionado. El caso de uso acaba.
Cursos alternativos	-

<i>5.3.3.14</i>	<i>Crear conjunto de reglas</i>
Descripción	Permite crear un nuevo conjunto de reglas personalizado en la sesión actual a partir de reglas existentes en otros conjuntos.
Actores	Investigador.

Precondiciones	
1. En la sesión actual hay algún conjunto de reglas de élite.	
Postcondiciones	
1. El sistema crea un nuevo conjunto de reglas personalizado, lo añade a la sesión actual y lo guarda en la base de datos. 2. El sistema añade reglas individuales seleccionadas por el usuario, pertenecientes a conjuntos de reglas de élite de la sesión actual, al nuevo conjunto.	
Curso típico de acontecimientos	<p>1. El usuario pide al sistema crear un nuevo conjunto de reglas.</p> <p>2. El sistema muestra un formulario con un campo para introducir el nombre del conjunto de reglas.</p> <p>3. El usuario introduce el nombre deseado.</p> <p>4. El sistema lee el nombre y comprueba que sea diferente al de cualquier otro conjunto de la sesión actual y de los que están guardados en la base de datos.</p> <p>5. El sistema crea en la sesión un nuevo conjunto de reglas personalizado con el nombre especificado en el paso 3 y lo guarda en la base de datos.</p> <p>6. Ejecutar Gestionar reglas a partir del paso 3.</p> <p>7. El caso de uso acaba.</p>
Cursos alternativos	
Paso 4: Ya hay un conjunto de reglas en la sesión con el mismo nombre:	
	<p>1. El sistema notifica el error.</p> <p>2. El curso continúa en el paso 3.</p>

<p><i>5.3.3.15 Gestionar reglas</i></p>	
<p>Descripción</p>	
<p>Permite añadir reglas pertenecientes a conjuntos de reglas de élite presentes en la sesión actual a un conjunto personalizado. También permite eliminar reglas de un conjunto personalizado.</p>	
<p>Actores</p>	
<p>Investigador.</p>	
<p>Precondiciones</p>	<ol style="list-style-type: none"> 1. En la sesión actual hay algún conjunto de reglas de élite. 2. En la sesión actual está presente el conjunto de reglas personalizado que se desea modificar.
<p>Postcondiciones</p>	<ol style="list-style-type: none"> 1. El sistema modifica un conjunto de reglas personalizado añadiendo reglas pertenecientes a conjuntos de reglas de élite de la sesión actual y/o eliminando reglas del conjunto personalizado.
<p>Curso típico de acontecimientos</p>	<ol style="list-style-type: none"> 1. El usuario selecciona un conjunto de reglas personalizado de la sesión actual. 2. El usuario pide al sistema gestionar las reglas del conjunto seleccionado en el paso 1. 3. El sistema muestra un formulario que lista todas las reglas presentes en el conjunto personalizado seleccionado en el paso 1. 4. El usuario selecciona las reglas que quiere eliminar del conjunto personalizado. 5. El sistema elimina del conjunto personalizado las reglas seleccionadas en el paso 4. 6. El sistema muestra todos los conjuntos de reglas de élite presentes en la sesión actual.

<p>7. El usuario selecciona uno de los conjuntos de reglas de élite, en el cual existen reglas que desea añadir al conjunto personalizado.</p> <p>9. El usuario selecciona todas las reglas que desea añadir al conjunto personalizado.</p> <p>10. Mientras queden reglas por añadir, volver al paso 6.</p>	<p>8. El sistema muestra todas las reglas que componen el conjunto de élite seleccionado en el paso 6 y que no estén presentes ya en el conjunto personalizado.</p> <p>11. El sistema añade al conjunto personalizado todas las reglas seleccionadas en el paso 9.</p> <p>12. El caso de uso acaba.</p>
Cursos alternativos	
-	

<p><i>5.3.3.16 Obtener información de un conjunto de reglas</i></p>
Descripción
Permite a un usuario investigador obtener información sobre los parámetros con los que fue generado un conjunto de reglas.
Actores
Investigador.
Precondiciones
1. En la sesión actual hay algún conjunto de reglas.
Postcondiciones
-
Curso típico de acontecimientos

- | | |
|--|--|
| <ol style="list-style-type: none"> 1. El usuario selecciona uno de los conjuntos de reglas de la sesión actual. 2. El usuario pide al sistema mostrar información sobre el conjunto de reglas seleccionado en el paso 1. | |
|--|--|

Si el conjunto de reglas seleccionado es de élite:

- | | |
|--|--|
| | <ol style="list-style-type: none"> 3. El sistema muestra una tabla con los parámetros con los que fue generado el conjunto de reglas: valor, índice, períodos de entrenamiento y de validación, iteraciones, tamaño de la población, número de generaciones, número máximo de nodos, probabilidad de cruce y de mutación, costes de transacción. También se muestran las siguientes estadísticas: beneficio medio y de la mejor regla en los períodos de entrenamiento y validación, y evolución del <i>fitness</i> medio y del mejor individuo en cada generación para ambos períodos. 4. El caso de uso acaba. |
|--|--|

Si el conjunto de reglas seleccionado es personalizado:

- | | |
|---|--|
| <ol style="list-style-type: none"> 6. El usuario selecciona una de las reglas mostradas. 8. Mientras haya reglas de las cuales el usuario desee ver su información, volver al paso 6. | |
|---|--|

- | | |
|--|---|
| | <ol style="list-style-type: none"> 5. El sistema muestra una lista con todas las reglas que componen el conjunto. 7. El sistema muestra una tabla con los parámetros con los que fue generado el conjunto de reglas que dio lugar a la regla seleccionada en el paso 6 y sus estadísticas. Esta información es la misma que la indicada en el paso 3. 9. El caso de uso acaba. |
|--|---|

Cursos alternativos
-

5.3.3.17	<i>Analizar costes de transacción</i>
Descripción	<p>Permite a un usuario investigador visualizar si la aplicación de un conjunto de reglas a una serie de precios durante un periodo bate a la estrategia <i>buy and hold</i>. Los resultados aparecen clasificados según un conjunto de precios por transacción. Concretamente, se mostrará para cada regla del conjunto de reglas, su beneficio teniendo en cuenta diferentes precios de transacción. También se mostrará el beneficio según el voto mayoritario del conjunto de reglas para cada coste de transacción. Todos los beneficios mostrados serán en relación a los obtenidos siguiendo la estrategia <i>buy and hold</i>.</p>
Actores	
Investigador.	
Precondiciones	<ol style="list-style-type: none"> 1. En la sesión actual hay algún conjunto de reglas.
Postcondiciones	<ol style="list-style-type: none"> 1. Se actualiza la información del conjunto de reglas que indica cuál ha sido el último periodo de test que ha utilizado.
Curso típico de acontecimientos	<ol style="list-style-type: none"> 1. El usuario pide al sistema analizar costes de transacción. 2. Ejecutar <i>Seleccionar serie de precios</i>. 3. Ejecutar <i>Seleccionar conjunto de reglas</i>. 4. El sistema muestra un formulario para introducir diversos costes de transacción. 5. El usuario introduce los costes por transacción con los cuales deseé ver resultados. 6. El sistema aplica el conjunto de reglas seleccionado en el paso 3 a la serie de precios y periodo escogidos en el paso 2, para todos los costes de transacción introducidos en el paso 5.

	<ol style="list-style-type: none"> 7. El sistema muestra una tabla con los beneficios respecto a <i>buy and hold</i> obtenidos en el paso 6 para cada regla y coste de transacción, y también el beneficio que resulta de operar según el voto mayoritario del conjunto. 8. El sistema actualiza la asociación “test” del conjunto con los registros utilizados para aplicar las reglas. 9. El caso de uso acaba.
Cursos alternativos	
-	

<i>5.3.3.18 Analizar resultados</i>			
Descripción	<p>Permite a un usuario investigador visualizar los beneficios resultantes de la aplicación de un conjunto de reglas a una serie de precios durante un periodo. Los resultados son relativos a los obtenidos mediante la estrategia <i>buy and hold</i>.</p>		
Actores			
Investigador.			
Precondiciones	<ol style="list-style-type: none"> 1. En la sesión actual hay algún conjunto de reglas. 		
Postcondiciones	<ol style="list-style-type: none"> 1. Se actualiza la información del conjunto de reglas que indica cuál ha sido el último periodo de test que ha utilizado. 		
Curso típico de acontecimientos	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 5px; vertical-align: top;"> <ol style="list-style-type: none"> 1. El usuario pide al sistema analizar resultados. </td> <td style="padding: 5px; vertical-align: top;"> <ol style="list-style-type: none"> 2. Ejecutar <i>Seleccionar serie de precios</i>. 3. Ejecutar <i>Seleccionar conjunto de reglas</i>. 4. El sistema muestra un formulario para introducir el coste de transacción. </td> </tr> </table>	<ol style="list-style-type: none"> 1. El usuario pide al sistema analizar resultados. 	<ol style="list-style-type: none"> 2. Ejecutar <i>Seleccionar serie de precios</i>. 3. Ejecutar <i>Seleccionar conjunto de reglas</i>. 4. El sistema muestra un formulario para introducir el coste de transacción.
<ol style="list-style-type: none"> 1. El usuario pide al sistema analizar resultados. 	<ol style="list-style-type: none"> 2. Ejecutar <i>Seleccionar serie de precios</i>. 3. Ejecutar <i>Seleccionar conjunto de reglas</i>. 4. El sistema muestra un formulario para introducir el coste de transacción. 		

<p>5. El usuario introduce el coste por transacción con el cual deseé aplicar las reglas.</p>	<p>6. El sistema aplica el conjunto de reglas seleccionado en el paso 3 a la serie de precios y periodo escogidos en el paso 2, con el coste por transacción introducido en el paso 5.</p> <p>7. El sistema muestra una tabla con los beneficios respecto a <i>buy and hold</i> obtenidos en el paso 6 para cada regla del conjunto seleccionado en el paso 3, y también el beneficio que resulta de operar según el voto mayoritario del conjunto.</p> <p>8. El sistema actualiza la asociación “test” del conjunto con los registros utilizados para aplicar las reglas.</p> <p>9. El caso de uso acaba.</p>
Cursos alternativos	-

<p><i>5.3.3.19</i></p>	<p><i>Mostrar comportamiento</i></p>
Descripción	<p>Permite a un usuario investigador visualizar para cada jornada de un periodo perteneciente a una serie de precios, la proporción de reglas de un conjunto que se posicionan dentro.</p>
Actores	
Investigador.	
Precondiciones	<p>1. En la sesión actual hay algún conjunto de reglas.</p>
Postcondiciones	<p>1. Se actualiza la información del conjunto de reglas que indica cuál ha sido el último periodo de test que ha utilizado.</p>
Curso típico de acontecimientos	

<p>1. El usuario pide al sistema mostrar comportamiento.</p>	<p>2. Ejecutar Seleccionar serie de precios.</p> <p>3. Ejecutar Seleccionar conjunto de reglas.</p> <p>4. El sistema aplica el conjunto de reglas seleccionado en el paso 3 a la serie de precios y periodo escogidos en el paso 2.</p> <p>5. El sistema muestra, para cada día de la serie de precios escogida en el paso 2, la proporción de reglas posicionadas dentro del mercado.</p> <p>6. El sistema actualiza la asociación “test” del conjunto con los registros utilizados para aplicar las reglas.</p> <p>7. El caso de uso acaba.</p>
Cursos alternativos	-

<p><i>5.3.3.20 Seleccionar índice</i></p>	
Descripción	
Permite a un usuario seleccionar un índice existente en la base de datos.	
Actores	
Usuario.	
Precondiciones	
-	
Postcondiciones	
-	
Curso típico de acontecimientos	
	<p>1. El sistema muestra una lista con todos los índices existentes en la base de datos.</p>

2. El usuario selecciona el índice deseado.	3. El caso de uso acaba.
Cursos alternativos	
-	

<i>5.3.3.21 Seleccionar índices</i>	
Descripción	
Permite a un usuario seleccionar varios índices existentes en la base de datos.	
Actores	
Usuario.	
Precondiciones	
-	
Postcondiciones	
-	
Curso típico de acontecimientos	
2. El usuario selecciona todos los índices deseados.	<p>1. El sistema muestra una lista con todos los índices existentes en la base de datos.</p> <p>3. El caso de uso acaba.</p>
Cursos alternativos	
-	

<i>5.3.3.22 Seleccionar índice y valor</i>	
Descripción	
Permite a un usuario seleccionar un índice existente en la base de datos y a continuación un valor perteneciente al índice. El valor ha de existir en la base de datos.	
Actores	

Usuario.	
Precondiciones	
-	
Postcondiciones	
-	
Curso típico de acontecimientos	<ol style="list-style-type: none"> 1. Ejecutar Seleccionar índice. 2. El sistema muestra una lista con todos los valores existentes en la base de datos pertenecientes al índice seleccionado en el paso 1. <p>3. El usuario selecciona el valor deseado.</p> <ol style="list-style-type: none"> 4. El caso de uso acaba.
Cursos alternativos	
-	

5.3.3.23	<i>Seleccionar serie de precios</i>
Descripción	Permite a un usuario seleccionar una serie de precios, perteneciente a algún valor o cargada de un fichero, y un periodo de dicha serie.
Actores	
Usuario.	
Precondiciones	
-	
Postcondiciones	
-	
Curso típico de acontecimientos	<ol style="list-style-type: none"> 1. El sistema muestra dos opciones para seleccionar la serie de precios:

	<ul style="list-style-type: none"> • Precios de un valor presente en el sistema. • Serie de precios cargada de un fichero.
2. El usuario escoge una de las dos opciones.	
<i>Si la opción escogida es “precios presentes en el sistema”:</i>	3. Ejecutar Seleccionar índice y valor.
<i>Si la opción escogida es “precios de un fichero”:</i>	<p>4. El sistema muestra un formulario con un campo para entrar la ruta del fichero que contiene la serie de precios en formato csv.</p> <p>5. El usuario introduce la ruta del fichero que contiene los datos</p> <p>6. El sistema lee el fichero con la serie de precios, y crea un índice y valor temporales para contener los datos del fichero.</p>
<i>Fin de la condición</i>	<p>7. El sistema muestra un formulario con dos campos para entrar las fechas de inicio y de final del periodo a seleccionar.</p> <p>8. El usuario rellena el formulario con las fechas deseadas.</p> <p>9. El caso de uso acaba.</p>
Cursos alternativos	
Paso 7: El fichero no tiene un formato adecuado o no se puede abrir:	
	<p>1. El sistema notifica el error ocurrido.</p> <p>2. El curso continúa en el paso 5.</p>

<i>5.3.3.24</i>	<i>Seleccionar conjunto de reglas</i>
Descripción	Permite a un usuario seleccionar un conjunto de reglas de élite existente en la sesión actual.
Actores	Usuario.
Precondiciones	-
Postcondiciones	-
Curso típico de acontecimientos	<ol style="list-style-type: none"> 1. El sistema muestra un formulario con todos los conjuntos de élite existentes en la sesión. 2. El usuario selecciona un conjunto de reglas. 3. El caso de uso acaba.
Cursos alternativos	-

5.4 Modelo conceptual

Los conceptos que son relevantes en el sistema para poder realizar los casos de uso planeados se muestran en el modelo conceptual. Éste consta del diagrama de clases y las restricciones de integridad del diagrama.

5.4.1 Diagrama de clases

El diagrama de clases proporciona una visión general de los objetos principales del sistema, junto con sus atributos y relaciones. A continuación se muestra el diagrama correspondiente al sistema implementado en este proyecto.

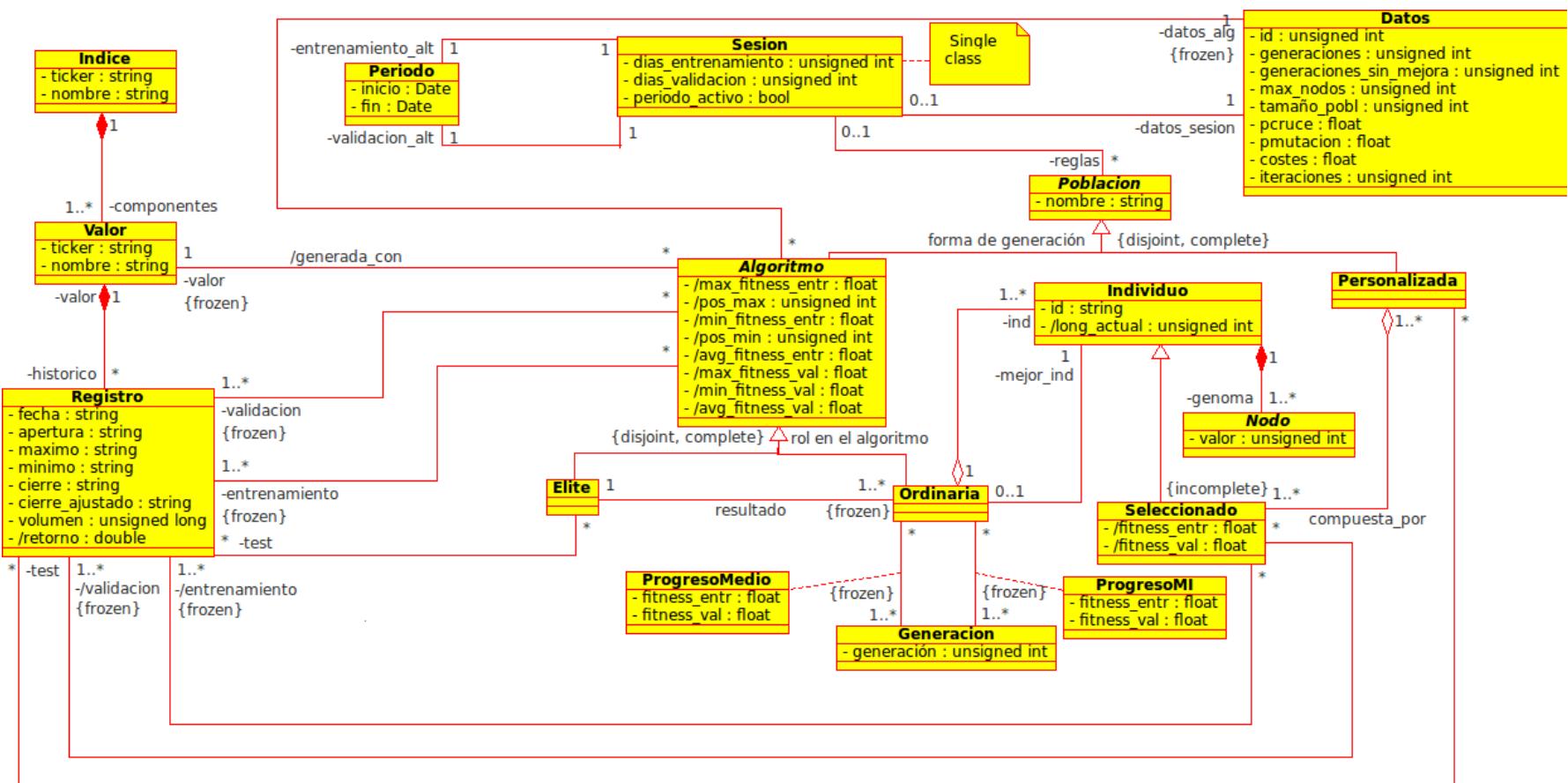


Figura 21: Diagrama de clases.

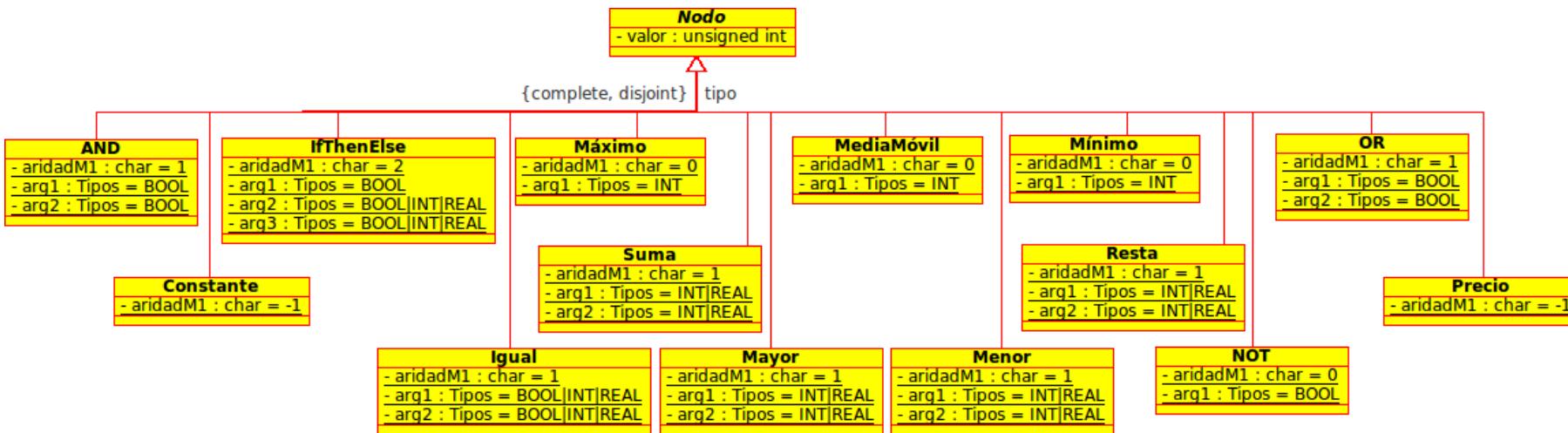


Figura 22: Diagrama de clases (cont.).

Los diagramas mostrados cuentan con las siguientes restricciones, que no pueden ser representadas gráficamente:

- Una instancia de *Algoritmo* tiene todas las instancias de *Registro* con las que está asociado mediante “entrenamiento” y “validación” pertenecientes a días de mercado consecutivos.
- Una instancia de *Algoritmo* tiene todas las instancias de *Registro* con las que está asociado mediante “entrenamiento” y “validación” pertenecientes a una misma instancia de *Valor*.
- Una instancia de *Elite* tiene todas las instancias de *Registro* con las que está asociado mediante “test” pertenecientes a una misma instancia de *Valor*.
- Una instancia de *Elite* tiene todas las instancias de *Registro* con las que está asociado mediante “test” pertenecientes a días consecutivos de mercado.
- La asociación “mejor_ind” de *Ordinaria* se corresponde con alguna de las instancias de *Individuo* de la asociación “ind”.
- Todas las instancias de *Elite* y *Ordinaria* unidas mediante la asociación “resultado”, tienen las mismas asociaciones “valor”, “entrenamiento”, “validación” y “datos_alg”.
- Una instancia de *Seleccionado* solo puede ser creada si hay alguna instancia de *Ordinaria* que apunta a ella mediante la asociación “mejor_ind”.

Los atributos derivados de los objetos de los diagramas se obtienen de la siguiente manera:

- Algoritmo:
 - max_fitness_entr: Si es de tipo *Ordinaria*, es el resultado de aplicar la función de *fitness* en el periodo definido por la asociación “entrenamiento” sobre todas las instancias de *Individuo* de la asociación “ind” y quedarse con el valor máximo dado. Si es de tipo *Elite*, las instancias de *Individuo* sobre las que se aplica son las presentes en la asociación “mejor_ind” de cada instancia de *Ordinaria* de la asociación “resultado”.
 - pos_max: Si es de tipo *Ordinaria*, conserva la posición en la que se encuentra la instancia de *Individuo* que da valor al atributo max_fitness_entr en la asociación “ind”. Si es de tipo *Elite*, la posición guardada es la de la instancia de *Ordinaria* de la asociación “resultado” que posee la instancia de *Individuo* que da valor al atributo max_fitness_entr.
 - min_fitness_entr: Análogo a max_fitness_entr, aunque con el mínimo valor de *fitness*.
 - pos_min: Análogo a pos_max, aunque aplicado al atributo min_fitness_entr.
 - avg_fitness_entr: Análogo a max_fitness_entr, aunque con el valor medio del *fitness* de todas las instancias de *Individuo* relevantes.
 - max_fitness_val: Análogo a max_fitness_entr, aunque con la función de *fitness* aplicada sobre el periodo definido por la asociación “validación”.
 - min_fitness_val: Análogo a max_fitness_val, aunque con el mínimo valor de *fitness*.

- avg_fitness_entr: Análogo a max_fitness_val, aunque con el valor medio del *fitness* de todas las instancias de *Individuo* relevantes.
 - valor: Apunta a la instancia de *Valor* a la que pertenecen todas las instancias de *Registro* apuntadas por “entrenamiento” y “validación”.
- Registro:
 - retorno: Es el resultado de calcular la variación del precio de cierre de la instancia actual con respecto a la instancia con fecha inmediatamente anterior y perteneciente a las mismas instancias de *Valor* e *Índice*. En otras palabras, es la variación de precio del día actual con respecto al anterior.
- Individuo:
 - long_actual: Número de instancias de *Nodo* que posee la asociación “genoma”.
- Seleccionado:
 - fitness_entr: Es el resultado de aplicar la función de *fitness* en el periodo definido por la asociación “entrenamiento” de la instancia de *Ordinaria* a la que está asociado.
 - fitness_val: Es el resultado de aplicar la función de *fitness* en el periodo definido por la asociación “validación” de la instancia de *Ordinaria* a la que está asociado.
 - entrenamiento: Contiene las mismas instancias de *Registro* que la asociación “entrenamiento” de la instancia de *Ordinaria* a la que está asociado.
 - validación: Contiene las mismas instancias de *Registro* que la asociación “validación” de la instancia de *Ordinaria* a la que está asociado.

Los atributos que hacen de claves primarias de cada objeto se muestran en la siguiente tabla:

Clase	Atributos
Índice	nombre
Valor	nombre (<i>Índice</i>), ticker
Registro	nombre (<i>Índice</i>), ticker (<i>Valor</i>), fecha
Población	nombre
Individuo	id
Generación	número
ProgresoMedio	nombre (<i>Ordinaria</i>), número (<i>Generación</i>)
ProgresoMI	nombre (<i>Ordinaria</i>), número (<i>Generación</i>)
Datos	id

Nótese que solo se muestran las claves primarias de las clases que no son *singleton* (de las cuales solo hay una instancia de ellas en todo momento), ya que no es necesario tenerlas identificadas. Tampoco se muestran las de las subclases, ya que se identifican del mismo modo que la superclase de la que heredan. Las instancias de la clase *Nodo* se identifican por la instancia de *Individuo* a la que pertenecen y la posición que ocupan en la asociación “genoma”, ya que son una composición de la clase *Individuo* y solo pueden existir asociadas a alguna instancia de *Individuo*.

Las características de los objetos más importantes mostrados en el diagrama mostrado se explican a continuación:

- Registro: Representa a los datos de un día de mercado de un determinado valor e índice.
- Población: Representa a cualquiera de las poblaciones que pueden ser creadas.
- Algoritmo: Representa a una población, que, para ser creada, se debe aplicar el algoritmo de programación genética sobre ella. Este tipo de poblaciones, una vez construidas, no pueden cambiar los individuos que forman parte de ellas, pero sí que pueden variar de individuos mientras el algoritmo es aplicado.
- Elite: Representa a una población compuesta por diversos individuos surgidos de la aplicación del algoritmo de programación genética previamente y cada uno de ellos ha sido el mejor de su población. Las instancias de *Individuo* pueden ser accedidas mediante la asociación “resultado” y a continuación “mejor_ind”.
- Ordinaria: Representa a una población de individuos (instancias de *Individuo* accesibles mediante la asociación “ind”) sobre los que se aplica el algoritmo de programación genética para evolucionarlos. Una vez finalizada la aplicación del algoritmo, la relación “ind” puede no ser conservada.
- Personalizada: Representa a una población a la que se pueden agregar un número indeterminado de individuos que ya forman parte de la asociación “mejor_ind” alguna instancia de *Ordinaria*. Estos individuos, representados por instancias de *Seleccionado*, pueden ser agregados y eliminados de la población en todo momento.
- Individuo: Representa a cualquier individuo utilizado durante o después de la aplicación del algoritmo de programación genética. Las instancias a las que se ha terminado de aplicar el algoritmo y que no cuentan con una instancia de *Ordinaria* apuntando hacia ellos mediante la relación “mejor_ind”, no es necesario conservarlos.
- Seleccionado: Representa a una instancia de *Individuo* que forma parte de alguna instancia de *Personalizada*.
- Nodo: Representa a un nodo del árbol de una instancia de *Individuo*.
- ProgresoMedio: Guarda las estadísticas sobre el *fitness* medio de una población en una generación determinada.
- ProgresoMI: Guarda las estadísticas sobre el *fitness* del mejor individuo de la población en una generación determinada.
- Datos: Representa a los parámetros con los que ha sido generada una población, excluyendo los períodos de entrenamiento y validación.
- Sesión: Conserva los parámetros por defecto que se utilizarán para aplicar el algoritmo de programación genética en las instancias de *Ordinaria* que van a ser creadas. Si “periodo_activo” es cierto, se utilizarán los períodos de tiempo de las asociaciones “entrenamiento_alt” y “validacion_alt”. Si “periodo_activo” es falso, se utilizarán períodos relativos al día actual: tantos días como “días_validacion” anteriores a la actualidad para el periodo de validación y tantos como “días_entrenamiento” anteriores al primer día de validación para el periodo de entrenamiento.

Referencias utilizadas en el capítulo: [11], [12].

6. DISEÑO

Este capítulo detallará cómo actúa internamente el sistema software desarrollado en el proyecto. En concreto, se partirá del resultado obtenido en la etapa de especificación, tratada en el anterior capítulo y que principalmente indica qué es lo que tiene que hacer el sistema y su relación con los usuarios, los requisitos no funcionales, también detallados en el capítulo anterior, y la tecnología con la que realizar la implementación, es decir, los recursos hardware y software que serán utilizados en la etapa de implementación.

El diseño describe la arquitectura utilizada en el sistema, la organización de la base de datos, los detalles de la interfaz de comunicación con el sistema y la lógica utilizada en el dominio del sistema. En definitiva, se produce la documentación necesaria que permite implementar el sistema software.

6.1 Arquitectura del sistema

La arquitectura define las relaciones entre componentes del sistema. Estos componentes son los bloques que constituyen la arquitectura y están compuestos por un conjunto de módulos, clases de programación (en caso de que el lenguaje de implementación sea orientado a objetos, como es el caso de este proyecto), funciones relacionadas entre sí, etc. Cada componente realiza unas determinadas funciones y tiene una interfaz a través de la cual otros componentes se pueden comunicar para pedir que éste realice una acción.

El sistema software realizado para este proyecto tiene una complejidad moderada. Ha de disponer de funcionalidades poco relacionadas entre sí, aunque dependientes unas de otras, y cada una de ellas requiere una planificación importante (interfaz de comunicación con el usuario, integración del algoritmo de programación genética, control de una base de datos). También se desea que el sistema sea fácilmente ampliable en el futuro (requisito no funcional). Por estos motivos se ha decidido adoptar una arquitectura en tres capas.

La arquitectura escogida es ampliamente utilizada para proyectos software de tamaño mediano y grande. Cada capa representa a un componente arquitectónico y la división de las funciones del sistema en tres capas permite separar tareas que no tienen nada que ver entre ellas y agrupar las que tienen cometidos relacionados. Ello permite reducir la dificultad para entender el sistema. En esta arquitectura, las capas están relacionadas jerárquicamente, de tal forma que una capa solo se puede comunicar con capas adyacentes. Normalmente existen tres capas: presentación, dominio y datos, aunque el número puede variar según las características del proyecto.

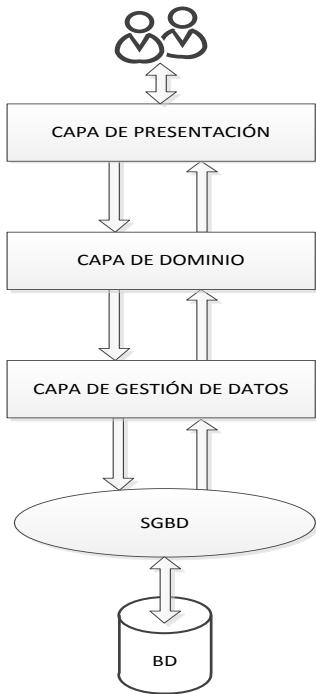


Figura 23: Esquema de la arquitectura en 3 capas.

6.1.1 Capa de presentación

La capa de presentación es la responsable de interaccionar con el usuario, es decir, recibir sus peticiones, mostrar el resultado de estas peticiones y hacer lo necesario para resolverlas. Para llevar a cabo este último paso, esta capa utiliza a la capa de dominio, en la cual delega la resolución de las peticiones. La capa de presentación únicamente controla que el formato de los datos introducidos por el usuario sea el correcto (por ejemplo, que el contenido de un campo numérico sea un número o que un determinado campo en el que se debe introducir un número fijo de caracteres, tenga ese número exacto), para el resto de tareas utiliza las funciones que ofrece la capa de dominio.

El diseño de esta capa debe tener en cuenta el requisito no funcional de facilidad de aprendizaje, apuntado en el apartado 5.2.2. Por ello, se utilizará una interfaz gráfica, que permite al usuario llevar el control de las acciones a realizar y la información se puede presentar de forma clara. Las guías seguidas para conseguir que la interfaz posea el requisito no funcional comentado son las siguientes: agrupar funcionalidades relacionadas en menús, tener pocos elementos en la vista de la interfaz, realizar pocos pasos para llevar a cabo un caso de uso, ser consistente en el formato de los elementos, y ser lo más claro, orientativo y explícito posible en los mensajes mostrados.

Al final de este apartado se muestran diagramas de navegación con todos los diálogos del sistema y sus relaciones. El significado de los componentes de los diagramas mostrados es el siguiente:

- Recuadros con bordes en ángulo recto: Representan diálogos.

- Recuadros ovalados: Representan a menús y sus opciones.
- Círculos con contorno sin oscurecer: Inicio de una acción.
- Círculos con contorno oscurecido: Fin de una acción. Implica regresar a la pantalla principal y cerrar todos los diálogos abiertos.
- Recuadros con bordes en ángulo recto y dos barras verticales a los lados: Indican una acción que contiene varios pasos y diálogos, la cual se detalla en otro sitio del diagrama.

El texto que se muestra en las transiciones entre componentes sigue el siguiente formato: *evento (argumentos) [condición] / acción a realizar*. El *evento* es la acción que realiza el usuario que provoca la transición; *argumentos* es información que necesitarán otros componentes que suceden al actual; *condición* indica qué tiene que cumplirse para poder llevarse a cabo la transición; *acción a realizar* es lo que sucede en el sistema al producirse la transición. Normalmente no se detallan en los diagramas todos los parámetros explicados, ya que no son necesarios o resultan obvios. Tampoco se muestra la acción para cancelar la acción en curso, ya que se puede realizar desde todos los diálogos. Como se ha comentado en la etapa de especificación, mostrar la mínima información que permita entender adecuadamente el sistema facilita la comprensión del mismo.

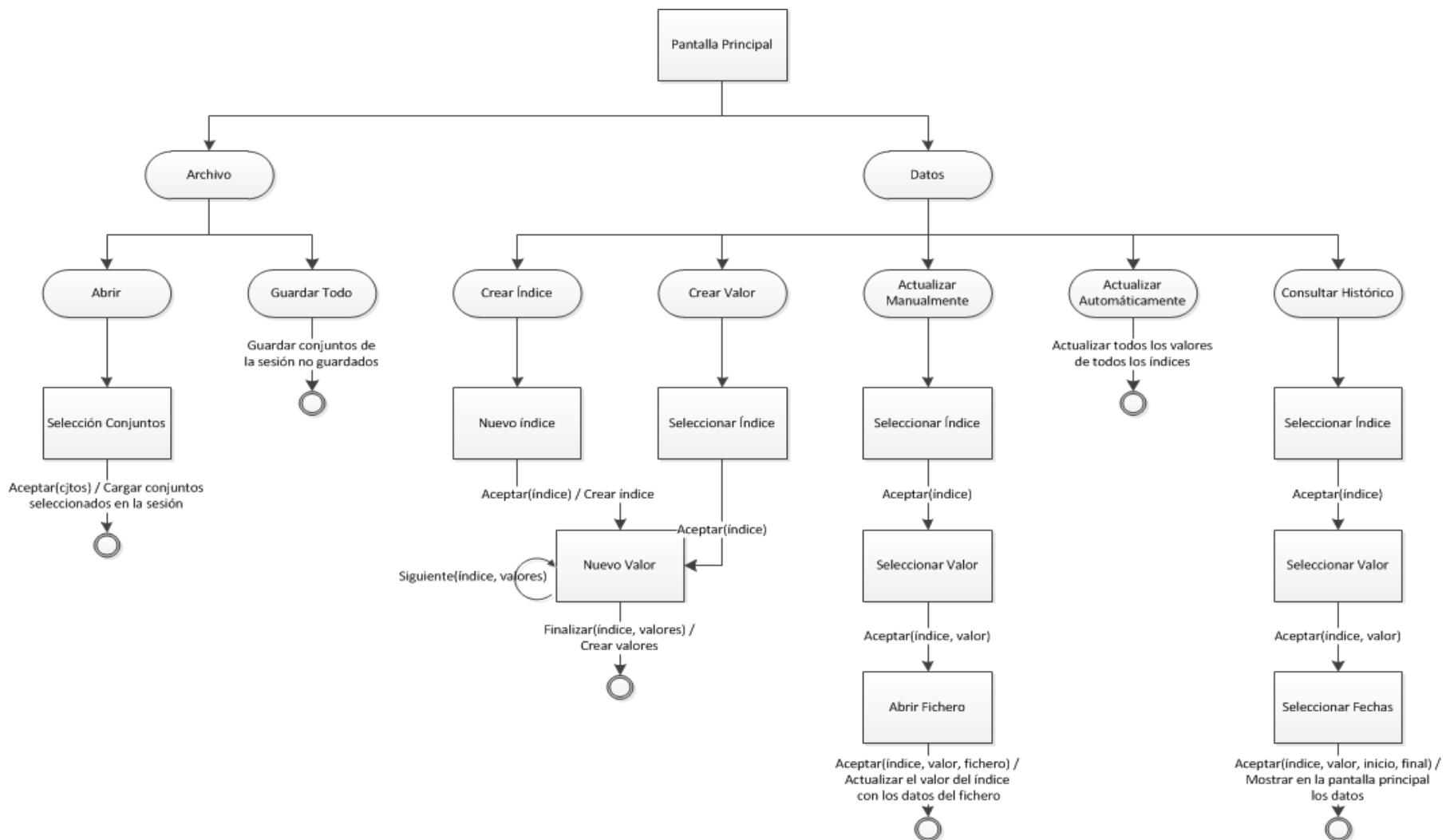


Figura 24: Mapa de navegación de la capa de presentación.

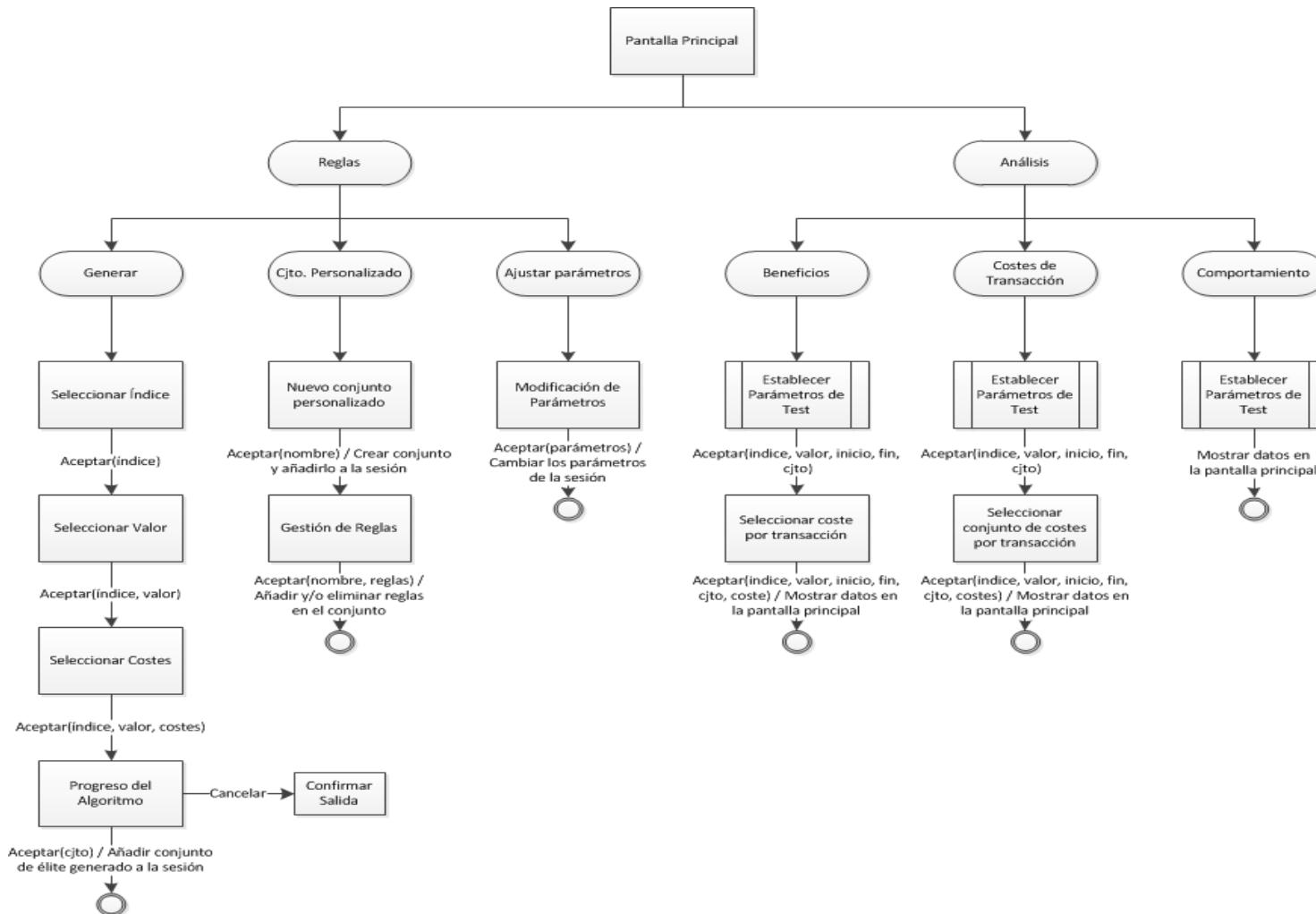


Figura 25: Mapa de navegación de la capa de presentación (cont.)

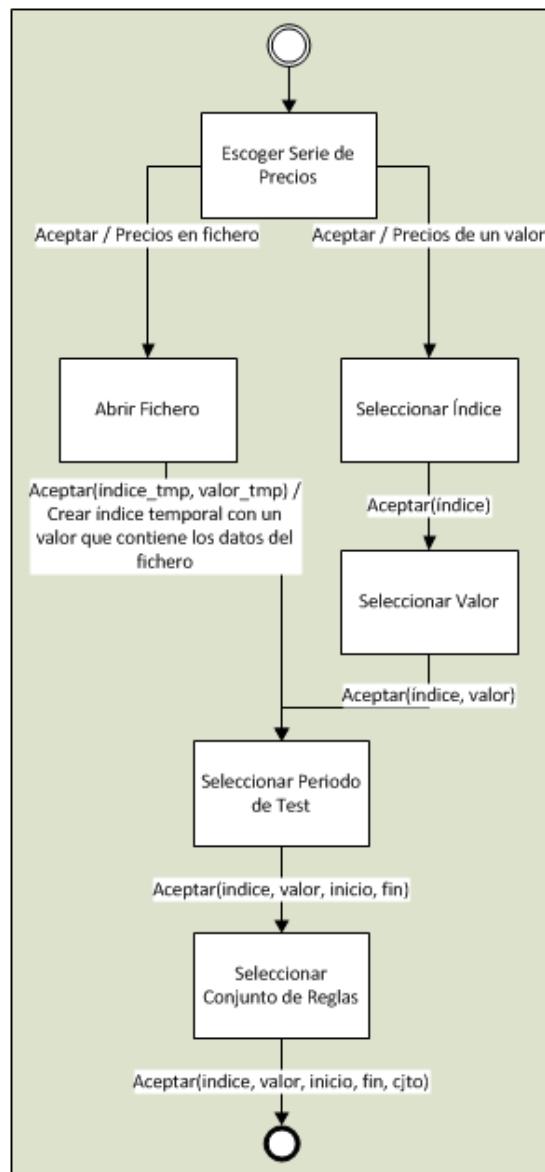
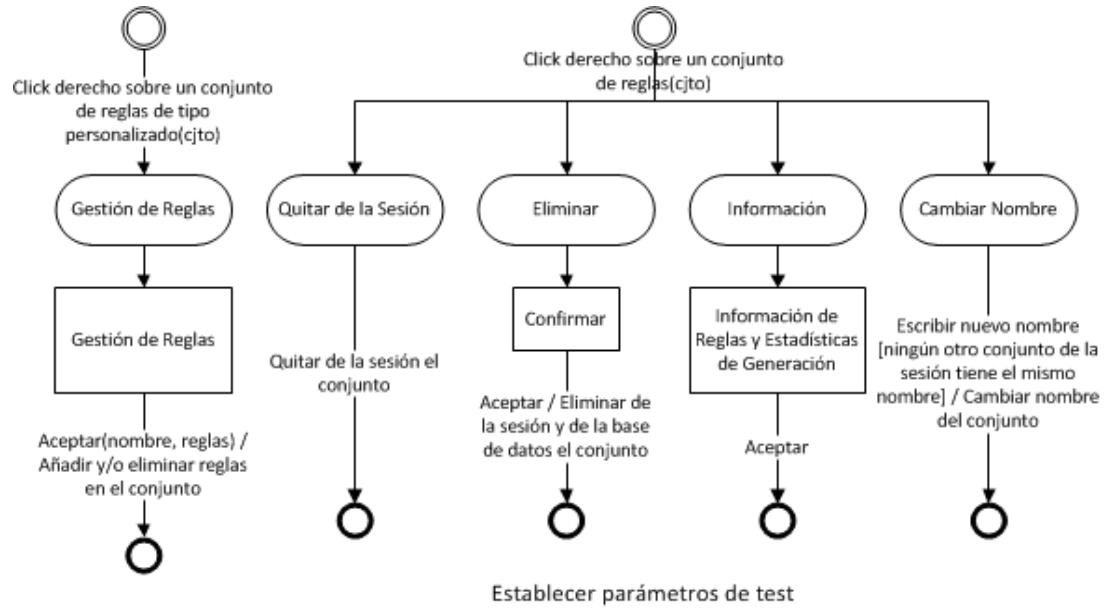


Figura 26: Mapa de navegación de la capa de presentación (cont. 2).

6.1.2 Capa de dominio

La capa de dominio contiene la lógica del sistema y se encarga de realizar sus funcionalidades. Es por ello que para el diseño de esta capa se requiere la documentación surgida de la etapa de especificación, que detalla qué es lo que debe hacer el sistema.

El diseño de la capa de dominio implica normalizar el diagrama de clases obtenido en la especificación (apartado 5.4.1). Para normalizarlo, es necesario tratar las asociaciones con más de dos objetos, las clases asociativas, la información derivada y las restricciones de integridad. Este paso permite la implementación directa del sistema en un lenguaje de programación orientado a objetos. Adicionalmente, al diagrama resultante existe la posibilidad de aplicar patrones de diseño. Estos patrones facilitan la implementación de una determinada situación, proveyendo una solución genérica que es fácilmente adaptable a una situación concreta. Por ejemplo, la especificación indica que una instancia de un objeto debe variar la subclase a la que pertenece durante su ciclo de vida. La programación orientada a objetos no permite a un objeto cambiar de subclase, pero existe un patrón de diseño, llamado patrón estado, que describe una forma práctica de emular este comportamiento en un objeto en cualquier lenguaje orientado a objetos.

En el diagrama de clases de la etapa de especificación mostrado en el apartado 5.4.1 no existe ninguna asociación con más de dos objetos involucrados, con lo cual no es necesario normalizar nada en este aspecto. Hay dos clases asociativas, *ProgresoMedio* y *ProgresoMI*, que necesitan ser transformadas a clases con asociaciones comunes. Esto también añade una restricción textual por cada una de las dos clases.

Todos los atributos derivados de la clase *Algoritmo* se materializarán, ya que su cálculo resulta muy costoso y necesitan ser consultados de forma rápida (son mostrados al consultar la información de una población). En cambio, la asociación derivada *generada_con*, se calculará porque resulta sencillo hacerlo. Los atributos y asociaciones derivados de *Seleccionado* se materializarán porque se desea conservar esa información aunque la instancia de *Ordinaria* que contiene dicha información se elimine del sistema en un momento dado. La información derivada de *Individuo* se materializará porque, aunque su cálculo no sea demasiado costoso, durante la ejecución del algoritmo el atributo es accedido con una frecuencia importante, y, como indica uno de los requisitos no funcionales, el algoritmo debe de ser lo más eficiente posible. En cuanto al atributo derivado de *Registro*, se calculará al iniciar la ejecución del algoritmo de programación genética, ya que, aunque su cálculo sea ligeramente costoso, el número de instancias de *Registro* existentes en la base de datos puede llegar a ser muy elevado, con lo cual guardar un atributo más para cada instancia aumentaría considerablemente el espacio necesario para hacer persistentes dichas instancias.

Se aplicará el patrón controlador fachada para definir las operaciones que se podrán utilizar desde fuera de la capa de dominio. Este patrón consiste en disponer de una clase *singleton*, la cual tiene todas las funciones a las que se puede llamar externamente junto con atributos para guardar el estado del dominio. Las operaciones que contiene esta clase se encargan de llamar

a las funciones internas de la capa para llevar a cabo las acciones que se solicitan desde el exterior. Este patrón es ampliamente usado en la capa de dominio con arquitectura de tres capas en sistemas que no disponen de un gran número de funcionalidades, ya que separa de forma sencilla la interfaz de comunicación de la capa y la estructura interna. Para sistemas complejos se utilizan otro tipo de controladores que permiten separar funcionalidades externas relacionadas entre ellas en clases distintas. Por ejemplo, el controlador papel agrupa en una clase todas las operaciones que puede realizar un actor, con una clase para cada actor del sistema; el controlador caso de uso junta en una clase todas las operaciones realizables en un caso de uso junto con los atributos para mantener el estado del caso de uso, con una clase para cada caso de uso.

A continuación se muestran las modificaciones realizadas al modelo conceptual comentadas en los anteriores párrafos. No se muestra el diagrama de los subtipos de *Nodo*, ya que no sufre ningún cambio. Para mejorar la legibilidad del diagrama no se han incluido las operaciones de cada clase.

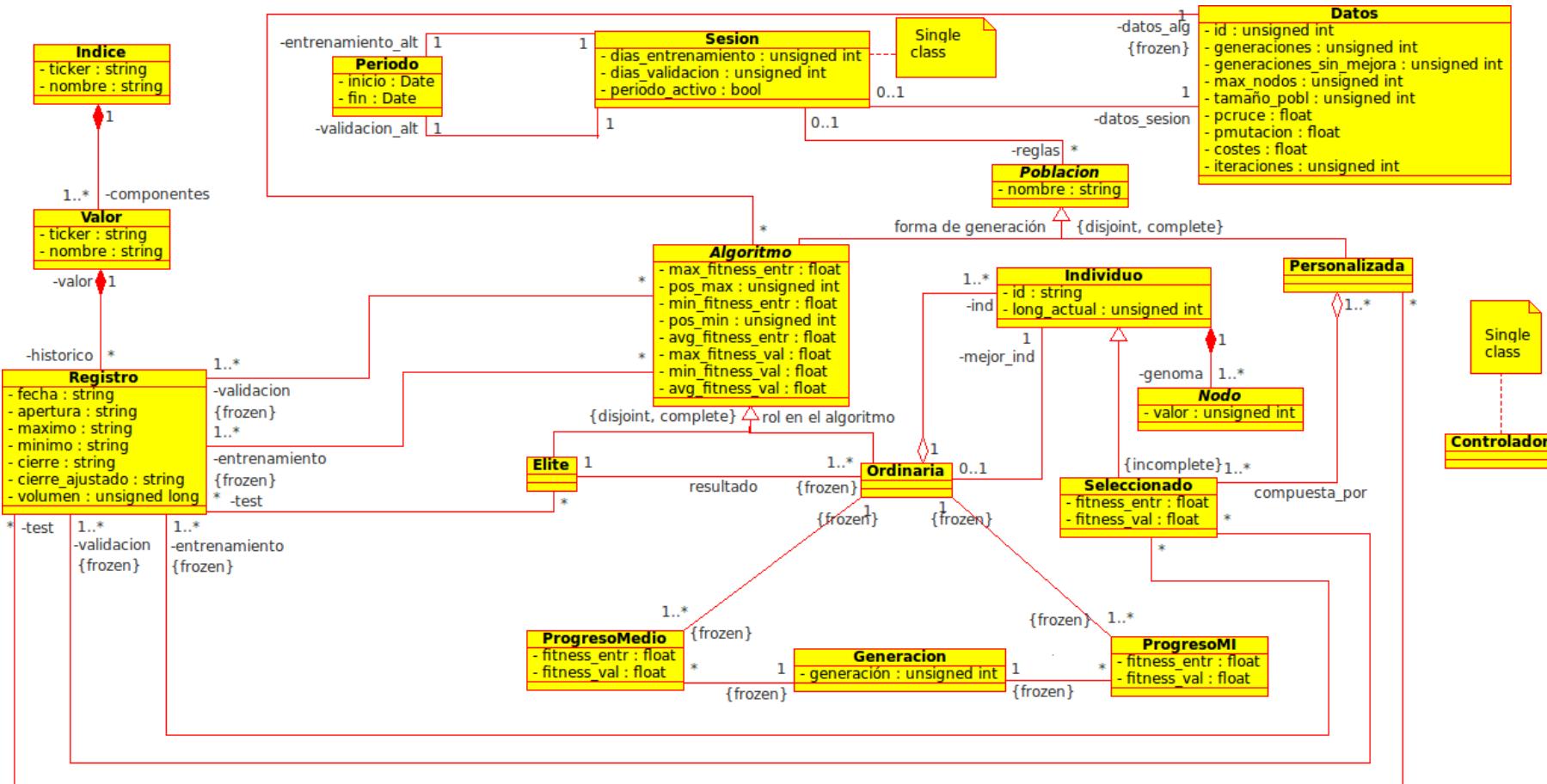


Figura 27: Diagrama de clases de la capa de dominio.

Las restricciones textuales que hay que añadir son las siguientes:

- No puede haber dos instancias de *ProgresoMedio* asociadas a las mismas instancias de *Ordinaria* y *Generación*.
- No puede haber dos instancias de *ProgresoMI* asociadas a las mismas instancias de *Ordinaria* y *Generación*.

Además de lo explicado, cabe decir que para ejecutar el algoritmo de programación genética, que se encuentra en la clase *Ordinaria*, se han realizado algunas modificaciones a la clase en la que se encuentra con el fin de aumentar su eficiencia y reducir su tiempo de ejecución. Las más importantes son las siguientes:

- Las instancias de *Individuo* involucradas en el algoritmo se guardan en un vector (en el que los objetos se encuentran en posiciones consecutivas de memoria) dentro de la misma clase, en lugar de tener un vector de punteros a los objetos, tal como se haría normalmente en un lenguaje orientado a objetos.
- Lo mismo se ha hecho con todas las instancias de *Registro* utilizadas en las etapas de entrenamiento y validación, las cuales se copian a un vector dentro de la clase. Además, cada posición de este vector no contendrá los datos que hay en una instancia de *Registro* en el mismo formato, sino que se guardarán ocupando en mínimo espacio posible y con un formato específico para poder ser usados eficientemente por el algoritmo. Esto se consigue guardando cada dato con un tipo simple de datos de la siguiente forma:
 - *fecha*: Utiliza el tipo *unsigned int*, y su contenido representa a una fecha en formato AAAAMMDD. De esta forma, comparar fechas resulta inmediato.
 - *apertura, máximo, mínimo, cierre, cierre_ajustado*: Utiliza el tipo *int*, y representan a un número en coma fija con 16 bits en la parte decimal, que es suficiente para representar precios con la precisión correcta. Además, estos precios ya se encuentran normalizados con la media de los 250 días anteriores, con lo cual el algoritmo tiene los datos en el formato adecuado.
- La estructura que permite escoger a los individuos, comentada en el apartado 4.2.4.1, es un atributo de la clase. No se muestra en el diagrama de clases para no dificultar la comprensión del sistema desde el punto de vista de la ingeniería del software, ya que únicamente es utilizada por el algoritmo.

6.1.3 Capa de gestión de datos

La capa de gestión de datos permite obtener y modificar datos almacenados de manera persistente, así como guardar nuevos datos. Esta capa interacciona directamente con el Sistema Gestor de la Base de Datos (SGBD) para poder llevar a cabo sus acciones. Por ello, la capa debe utilizar el lenguaje del SGBD (variante SQL que utiliza, si es que utiliza este lenguaje) para poder interactuar con él.

Para diseñar la capa de gestión de datos es necesario decidir cómo se guardarán en la base de datos los objetos resultantes del diagrama de clases de la capa de dominio y la estructura de la

base de datos. Siguiendo de forma estricta la teoría, cada objeto debería tener su tabla correspondiente en la base de datos. No obstante, el sistema construido para este proyecto no consta con un gran número de clases, lo cual permite relajar este concepto en algunas ocasiones, explicadas a lo largo de este apartado.

Las clases *singleton*, *Sesión*, *Periodo* y *Controlador*, no se guardan en la base de datos, ya que siempre que se carga el programa sus atributos contienen los mismos valores porque se encuentran fijados en el código.

Los objetos *Datos*, *Índice*, *Valor* y *Registro* cuentan cada uno de ellos con una tabla. El objeto *Ordinaria* no se guarda, ya que es un contenedor temporal para ejecutar el algoritmo de programación genética y no tiene sentido conservar los datos específicos usados en la ejecución. Los datos relevantes de este objeto ya son guardados por el objeto *Elite* que lo posee.

Las poblaciones se guardan en dos tablas diferentes, según sean *Elite* o *Personalizada*. Cada una de estas dos tablas cuenta, además de los atributos propios de la clase, con los atributos de las clases de las que hereda. Este método, consistente en guardar cada clase concreta en una tabla distinta y no hacer lo mismo con las clases abstractas, se conoce como *Concrete Table Inheritance*. La tabla que guarda los objetos *Elite*, tiene una relación con los objetos *Individuo* de la asociación “mejor_ind” de todos los objetos *Ordinaria* que tiene en la asociación “resultado”. Esto es así porque, como se ha comentado, los objetos *Ordinaria* no se guardan; solo se conserva la información relevante de éstos.

Cada objeto *Individuo* tiene una tabla, y, si es de tipo *Seleccionado*, también estará presente en otra tabla cuya clave primaria es la misma que en la primera tabla. La tabla de los objetos *Seleccionado* guarda las estadísticas de la población con la que fue generado el individuo, ya que es posible borrar la tabla que las conserva (donde se guardan los objetos *Elite*) sin borrar el objeto *Seleccionado*, e interesa mantener estos datos.

Una modificación importante es la realizada al guardar el historial de un individuo durante su generación, cuyos objetos en el diagrama del dominio son: *ProgresoMedio*, *ProgresoMI* y *Generación*. En la base de datos existe una tabla para los tres objetos, llamada *historial-individuo*. Además, al no haber ninguna tabla que guarde objetos *Ordinaria*, la tabla anterior está relacionada con el objeto *Individuo* de su asociación “mejor_ind”. Hay que tener en cuenta también que en la base de datos no se guarda ningún objeto *Individuo* que no pertenezca o haya pertenecido a una asociación “mejor_ind” de un objeto *Ordinaria*.

Los objetos *Nodo* se guardan en la misma tabla en la que se guarda el objeto *Individuo* del que forman parte. Un objeto *Nodo* debe pertenecer a un solo *Individuo* y no tendría sentido de otra forma, por ello es correcto hacerlo así. La razón de esta modificación es que, hay una gran cantidad de objetos *Nodo* que componen un objeto *Individuo*. Si se utilizase una tabla para los objetos *Nodo*, se provocaría que en la base de datos fuese sencillo alcanzar un número de entradas en esta tabla muy elevado, lo que introduciría un *overhead* importante al guardar los datos, tanto en tiempo como en espacio de disco, y al recuperarlos. Hay que tener en cuenta que habría que introducir un identificador para cada *Nodo* y un atributo que guarde el objeto

Individuo al que pertenece, lo cual aumentaría aún más el espacio requerido y el tiempo de inserción al tener que mantener claves foráneas.

El formato en el que se guarda un objeto *Nodo* en uno *Individuo* se muestra en la siguiente figura. Todos los objetos *Nodo* que pertenecen a un objeto *Individuo* se guardan en un mismo campo binario de este último, de forma consecutiva y con dos bytes al principio del campo que indican cuántos bytes tiene el campo (estos dos bytes del principio quedan excluidos de la cuenta):

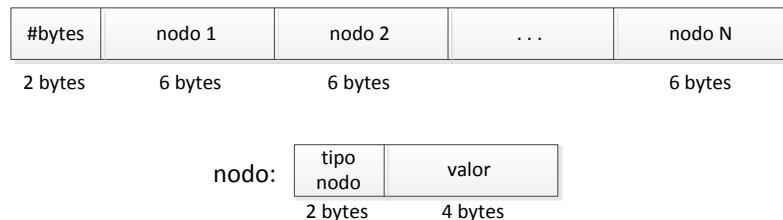


Figura 28: Formato en el que se guardan los objetos Nodo de un mismo Individuo en la base de datos.

Para poder acceder a las tablas guardadas en la base de datos y modificar su contenido, la capa de gestión de datos dispone de controladores encargados de transferir información entre los objetos mostrados en el diagrama de clases de la capa de dominio y las tablas de la base de datos. Estos controladores disponen por defecto de las siguientes operaciones, las cuales tienen como parámetros y devuelven como resultado objetos de la capa de dominio (cada controlador una sola clase de objeto):

- Nuevo: Crea el objeto pasado como parámetro en la base de datos. Si el objeto ya existe, devuelve error.
- Actualizar: Modifica los datos del objeto pasado como parámetro en la base de datos. Si el objeto no existe, devuelve error.
- Eliminar: Elimina los datos del objeto pasado como parámetro de la base de datos.
- Obtener: Devuelve un vector con todos los objetos cuyos atributos tengan unos valores específicos, indicados en los parámetros.
- Obtener todos: Devuelve un vector con todos los objetos presentes en la base de datos.

Es posible que algún controlador también tenga alguna otra funcionalidad, en cuyo caso se trata de alguna acción específica a las tablas que maneja e implementada en esta capa en vez de hacerlo en la de dominio para mejorar la eficiencia de la operación⁴. Los controladores que

⁴ Los SGBD disponen de un gran número de instrucciones, las cuales pueden ser aprovechadas para mejorar extraordinariamente la eficiencia de algunas operaciones realizadas sobre la base de datos, las cuales, utilizando las operaciones que ofrece por defecto el controlador de la capa de datos, resultarían muy costosas.

se han decidido que formen parte de esta capa corresponden a los siguientes objetos del diagrama de clases de la capa de dominio: *Elite*, *Personalizada*, *Índice*, *Valor* y *Registro*.

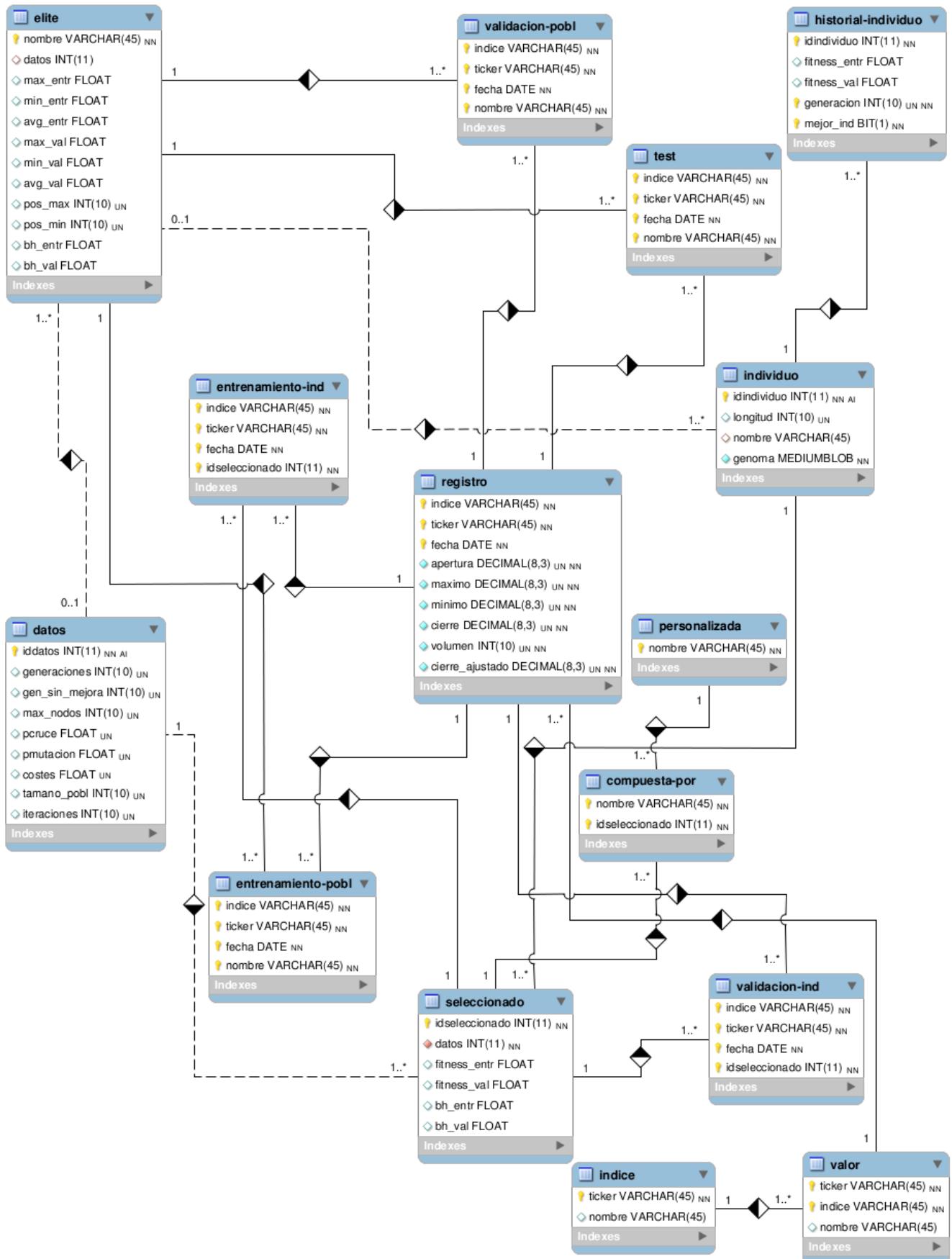


Figura 29: Modelo de datos de la base de datos.

Los atributos con una llave de color amarillo a la izquierda representan las claves primarias; los que tienen un rombo de color rojo son claves foráneas. Los símbolos que aparecen a la derecha de los atributos representan sus características y pueden ser los siguientes: NN (*Not Null*), UN (*UNsigned*), AI (*Auto Increment*).

Referencias utilizadas en el capítulo: [11], [12].

7. IMPLEMENTACIÓN

En este capítulo se detallará el entorno en el que se ha desarrollado la implementación del proyecto, su instalación, las librerías externas utilizadas, los lenguajes de programación y la base de datos empleada.

7.1 Lenguajes utilizados

7.1.1 C++

El lenguaje escogido para realizar la implementación del proyecto ha sido C++. Este lenguaje está basado principalmente en C y fue desarrollado como una extensión a él. Por ello es posible incluir código C en un proyecto implementado en C++, aunque no a la inversa; además ambos lenguajes comparten gran parte de su sintaxis. C++ es un lenguaje ampliamente utilizado para el desarrollo de todo tipo de aplicaciones, desde drivers hasta videojuegos, ya que posee una gran variedad de características.

El desarrollo del lenguaje comenzó en 1979 por Bjarne Stroustrup en AT&T Bell Labs. El proceso consistía en la implementación de mejoras al lenguaje de programación C, tales como clases, funciones virtuales, sobrecarga de operadores, herencia, plantillas de datos o excepciones. La primera versión comercial fue lanzada al mercado en 1985 y múltiples versiones la sucedieron en los años siguientes, hasta que en 1998 el primer estándar oficial del lenguaje apareció de la mano de *International Organization for Standardization (ISO)*, conocido como C++98 o formalmente como *ISO/IEC 14882:1998*. Otras revisiones del estándar fueron publicadas posteriormente, siendo la última hasta la fecha de esta documentación la C++11 o *ISO/IEC 14882:2011*, aprobada el 12 de Agosto de 2011. Es esta última versión la utilizada en la implementación de este proyecto.

C++ es un lenguaje orientado a objetos cuyas características más importantes son las siguientes:

- **Genérico:** Puede ser usado para implementar soluciones a problemas de distinta índole.
- **Multiparadigma:** Soporta diferentes formas de representar un sistema: orientado a objetos, procedural o una combinación de ambas.
- **Compilado:** Las instrucciones en lenguaje ensamblador, que son las que serán ejecutadas en última instancia por el procesador de la máquina, son generadas antes de ejecutar el programa (durante la etapa de compilación).
- **Estáticamente tipado:** Las variables y funciones del programa deben tener un tipo definido, el cual es comprobado en la etapa de compilación si resulta correcto.
- **Nivel intermedio:** Posee características de lenguajes de alto y de bajo nivel. Los primeros se caracterizan por tener instrucciones abstractas y alejadas de las que ejecutará el procesador, mientras que los segundos disponen de instrucciones que permiten controlar de forma cercana la ejecución de los datos en el procesador.

Este lenguaje, al ser compilado, provoca que la rapidez en la ejecución sea de las más elevadas que pueda ofrecer cualquier otro lenguaje de programación, a excepción de ensamblador. Adicionalmente, al disponer de características tanto de alto como de bajo nivel, facilita la tarea de implementar una interfaz gráfica de usuario, y al mismo tiempo, realizar optimizaciones de bajo nivel y controlar el valor de los bits de una variable de forma muy rápida. Junto a esto, el ser un lenguaje utilizado ampliamente, lo que implica que dispone de abundante documentación, y orientado a objetos, que es el paradigma que permite implementar de forma inmediata el diseño del sistema explicado en el capítulo 6, han sido los motivos por los cuales se ha tomado la decisión de implementar el software de este proyecto en C++.

Existen inconvenientes en este lenguaje, relacionados principalmente con la claridad del código escrito y la gran complejidad del lenguaje. Especialmente en proyectos grandes, el código puede resultar menos entendible que el mismo proyecto implementado en otro lenguaje. Además, el lenguaje dispone de una gran cantidad de instrucciones y conceptos, lo cual puede inducir a confusión en el programador. No obstante, se ha considerado que éstos inconvenientes son menores en comparación con las ventajas comentadas en el anterior párrafo, ya que no se trata de un proyecto excesivamente grande ni se utilizan conceptos complicados de forma recurrente.

7.1.2 El framework Qt

Para implementar la parte de la interfaz gráfica se ha utilizado Qt, un *framework* que proporciona diferentes utilidades que facilitan considerablemente la implementación de interfaces gráficas de usuario, el manejo de una base de datos o la transmisión de datos a través de una red de datos, entre otras cosas. En este proyecto solo se usan las utilidades que permiten construir interfaces gráficas, que por otra parte son las más populares de entre el resto de utilidades ofrecidas.

Este *framework* está implementado en C++, aunque existen conectores con otros muchos lenguajes que permiten usarlo en casi cualquiera. Sin embargo, si el proyecto en el que se emplea Qt está implementado en C++, se gana eficiencia y simplicidad al poder tener acceso a sus funcionalidades de forma directa sin tener que disponer de una librería específica para el lenguaje utilizado. Además, algunas de las funcionalidades que ofrece Qt no son soportadas por algunos conectores de algunos lenguajes.

Haavard Nord y Eirik Chambe-Eng, director general y presidente respectivamente de la empresa noruega Trolltech, comenzaron el desarrollo de Qt en 1991. La primera versión fue lanzada al año siguiente. En Qt existen las versiones mayores, identificables por el número en primer lugar (por ejemplo, Qt X.Y tendría como versión mayor la número X), las cuales difieren considerablemente de otras versiones mayores y no son compatibles entre ellas. Aparte están las versiones menores, que son revisiones de las versiones mayores y se identifican por el número en segundo lugar. La versión principal utilizada en la implementación de este proyecto es Qt 4 y fue lanzada en Junio de 2005, mientras que la versión menor es Qt 4.7 y se lanzó en Septiembre de 2010.

Qt es soportado en multitud de plataformas y sistemas operativos, de hecho es utilizado ampliamente en programación de dispositivos móviles, como por ejemplo Symbian, Meego, Windows CE o Embedded Linux. Actualmente, se está trabajando en que también sea soportado en otros sistemas muy utilizados, tales como iPhone, Android o OpenSolaris. Por ser usado tan ampliamente, la documentación disponible sobre el *framework* es extensa. Además, al ser independiente de la plataforma sobre la que se utiliza, permite que la documentación sea la misma para todos los sistemas, ganando en claridad.

7.1.3 Structured Query Languaje (SQL)

El programa implementado en C++ requiere comunicarse con el SGBD, para lo cual se necesita enviarle las órdenes en un lenguaje que sepa interpretar. El SGBD utilizado en este proyecto tiene SQL como único lenguaje soportado.

SQL son las siglas en inglés de *Lenguaje de Consulta Estructurado*, y es un lenguaje específico para operar con datos de bases de datos relacionales. Está basado en el álgebra relacional e inspirado por el modelo relacional descrito por Edgar F. Codd en 1970 en su artículo *A Relational Model of Data for Large Shared Data Banks*.

Este lenguaje surgió en 1974 a partir del trabajo realizado en IBM por Donald D. Chamberlin y Raymond F. Boyce con el objetivo de controlar el almacenamiento de datos en un SGBD similar a uno relacional. No obstante, no sucedió hasta junio de 1979 que la empresa Relational Software Inc., cuyo nombre en la actualidad es Oracle Corporation, comercializó la primera versión en sus productos. La primera estandarización del lenguaje fue efectuada por *American National Standards Institute (ANSI)* en 1986, con nombre *SQL-86*, que fue seguida por la de *ISO* en 1987, con nombre *SQL-87*. El último estándar existente data de 2008 y es conocido como *SQL:2008 (ANSI) o SQL 2008 (ISO)*.

Este lenguaje se diferencia de otros como C++ en que se programa de forma declarativa. Esto quiere decir que el programador debe indicar qué es lo que desea obtener, pero no el procedimiento seguido para conseguir el resultado. Sin embargo, hay extensiones al estándar que añaden instrucciones de la programación imperativa, típica de lenguajes orientados a objetos como C++, tales como control del flujo de ejecución (bucles, condicionales).

No obstante, cada SGBD dispone de su propia implementación del lenguaje, cuya sintaxis es igual que en el estándar en la mayor parte de las instrucciones pero con ligeras diferencias en algunas dependiendo del SGBD.

En la actualidad, la mayor parte de las bases de datos son relacionales y usan SQL como lenguaje para manejar los datos.

7.2 Sistema gestor de la base de datos

Para el sistema software realizado en este proyecto, se ha escogido a *MySQL 5.5* como SGBD. Éste tiene la particularidad de actuar como servidor en el sistema en el que se instala, de forma que se debe acceder a él por medio de una conexión de red, que puede ser desde el mismo sistema utilizando la dirección de *localhost*. Esto permite que la base de datos pueda ser accedida de manera concurrente por diferentes clientes conectados remotamente al sistema. De todos modos, el acceso concurrente no se tratará en este proyecto, ya que el SGBD será accedido desde el mismo sistema en el que se encuentra ejecutando por una única instancia del programa.

MySQL es uno de los SGBDs más populares a causa de que es robusto, eficiente y de código abierto. Por ello ha sido adoptado por gran cantidad de aplicaciones para sus bases de datos. Ejemplos de programas que usan *MySQL* como SGBD incluyen Joomla, TYPO3, WordPress o Drupal. Incluso es utilizado en sistemas que requieren gran capacidad de procesamiento de datos, tales como Google, Facebook, Twitter o Wikipedia.

A pesar de que MySQL dispone de funcionalidades avanzadas, también en la versión libre del software, en este proyecto se utilizarán un subconjunto reducido de funcionalidades simples, explicadas a continuación:

- **Integridad referencial:** La base de datos debe soportar referencias entre tablas, al existir claves foráneas. Por ello se ha escogido *InnoDB* como tecnología de almacenamiento de datos. Al mismo tiempo, esta tecnología proporciona soporte para transacciones, siguiendo el estándar *ACID* (atomicidad, consistencia, aislamiento, durabilidad), aunque no es importante en este proyecto.
- **Sub consultas:** En algunas ocasiones se utilizan sub consultas al comunicarse con el SGBD. Esto es, dentro de una operación de consulta de datos, hay una condición que depende del resultado de otra consulta, también escrita en el mismo código SQL de la operación. Es importante tener presente que estas operaciones son relativamente lentas y deben ser empleadas en la menor medida posible.
- **Codificación *UTF-8*:** La codificación de los campos de texto de las tablas, es decir, con cuantos bytes se representará cada carácter y cómo se interpretará, será siguiendo el estándar *UTF-8*. Éste permite guardar en un único byte por carácter todos los caracteres que se utilizan en español y otros que puedan estar en los identificadores de las acciones.
- **Actualización de tablas mediante ficheros:** Al actualizar la base de datos, más concretamente al actualizar el histórico de precios de las acciones, se introduce un gran volumen de datos provenientes de un fichero temporal descargado de Internet. Para que esta operación se realice con rapidez, en la implementación se usa una instrucción que proporciona el SGBD para actualizar una tabla a partir de un fichero [24]. Esto permite que la velocidad de inserción, en comparación con la ejecución de múltiples inserciones individuales, aumente unas 20 veces, ya que se elimina el *overhead* que se produce en la ejecución de las inserciones múltiples [25].

7.3 Entorno de desarrollo

El software se ha implementado bajo el sistema operativo *Ubuntu 10.04*, una distribución muy conocida de Linux. Es por esto que está construido para ejecutarse sobre dicha plataforma.

Las razones para elegir este sistema operativo son varias. Una de ellas es que resulta sencillo configurar el entorno de programación si se tiene una mínima experiencia (se explicará cómo hacerlo a lo largo del capítulo). Otra de las razones es que, en caso de producirse algún error durante la configuración, la cantidad de información que hay disponible en Internet es muy elevada y es probable encontrar la solución. Esto sucede porque es el sistema operativo más usado para tareas de programación, ya que su código ha sido público desde su creación y es posible tener un conocimiento de su estructura y de la forma en que realiza las tareas, lo que permite a los programadores tener más transparencia. De alguna forma, esto a su vez causa que el sistema operativo sea eficiente, ya que programadores de todo el mundo pueden colaborar en su diseño y aportar sus ideas para mejorarlo. Esta eficiencia es otra de las razones por las que elegir este sistema.

7.3.1 Librerías externas

Para la implementación del sistema se ha decidido utilizar algunas funcionalidades proporcionadas por librerías externas, es decir, que no vienen instaladas por defecto en el sistema operativo. De estas librerías, las que son dinámicas también se necesitan al ejecutar el software, ya que sus funciones se enlazan dinámicamente con el ejecutable del programa al empezar su ejecución y no las contiene en su código.

Boost es el nombre de un conjunto de librerías descargables desde la página web <http://www.boost.org/> e instalables sin ninguna restricción para el lenguaje de programación C++, utilizado para la implementación del proyecto. Estas librerías están desarrolladas por una comunidad abierta de usuarios, a la que cualquier programador puede unirse, y por ello su código es público. No obstante, para que una librería desarrollada por alguien pueda formar parte de *Boost*, es necesario que pase un filtro para comprobar que sea útil y estable. Existen un número elevado de librerías en *Boost* que proporcionan un gran número de utilidades, no obstante, se utilizará un conjunto reducido de ellas que será suficiente para proporcionar las utilidades deseadas.

Es importante distinguir entre las librerías necesarias en la implementación del programa y las requeridas durante la ejecución. En el segundo caso son necesarias las librerías dinámicas, con extensión *.so*⁵, las cuales ya están compiladas y permiten que el programa utilice sus funciones una vez compilado, sin tener que incluir el código de las funciones que ofrece la librería en el código del ejecutable. Esto ofrece la ventaja de reducir el tamaño del programa, aunque perjudica el rendimiento porque la dirección de las funciones de la librería se determina durante

⁵ En sistemas operativos basados en Unix, entre los que se incluyen todas las distribuciones Linux y Mac OS, la extensión es *.so*, y posiblemente, el número de versión de la librería a continuación, por ejemplo *librería.so.1.55*. En sistemas operativos Windows la extensión de las librerías dinámicas es *.dll*.

la ejecución. No obstante, ninguna librería dinámica es utilizada mientras el algoritmo de programación genética se está ejecutando, con lo cual no supone una pérdida importante de eficiencia en el programa. En el caso de las librerías necesarias en la implementación, además de las librerías dinámicas, son necesarias aquellas que son incluidas en el código del ejecutable, y que por tanto se requieren en la etapa de compilación del sistema. Este tipo de librerías únicamente contienen funciones *inline*, que se caracterizan por ser funciones con pocas líneas de código y que su código es incluido en los lugares donde se llama a la función (no se realiza la llamada habitual a función) durante la etapa de compilación.

Las librerías requeridas en la implementación y no en la ejecución del software construido en este proyecto son las siguientes: *asio.hpp* y *lexical_cast.hpp*. Ambas pertenecen al conjunto de librerías *Boost*, y como se puede comprobar, su extensión indica que son ficheros de cabecera de C++, es decir, código fuente sin compilar. Además, las funciones se encuentran implementadas en los mismos ficheros, lo cual indica que se trata de funciones *inline*⁶. La primera de estas librerías, *asio.hpp*, es utilizada para facilitar la comunicación con el servidor de datos a través de Internet. La segunda, *lexical_cast.hpp*, se emplea para convertir un tipo de datos en otro, en concreto de *string* a *double*.

En cuanto a las librerías utilizadas en la implementación y ejecución del sistema, es decir, las librerías dinámicas, han sido escogidas las siguientes: *libboost_date_time*, *libboost_system*, *libmysqlcppconn5*, *libqgui4*, *libqtc4* y *libmysqlclient16-dev*. Las dos primeras pertenecen al conjunto *Boost*, y son utilizadas respectivamente para facilitar las operaciones con fechas y para tratar casos de error en la comunicación con el servidor de datos. La librería *libmysqlcppconn5* proporciona una interfaz para la comunicación con el SGBD desde C++. La librería *libmysqlclient16-dev* también dispone de una interfaz de comunicación con el SGBD aunque desde C. Esta librería dispone de más funciones que la anterior, no obstante la anterior está específicamente construida para C++ y debe utilizarse siempre que sea posible si el lenguaje de desarrollo es C++ (como es el caso). En este proyecto, la librería construida en C es necesaria en un caso, en el que la librería en C++ no dispone de la función deseada (*escapeString*). Por último están las librerías que posibilitan la interfaz gráfica, *libqgui4* y *libqtc4*, y que disponen de los elementos y mecanismos gráficos utilizados en Qt, un *framework* de C++ para implementar interfaces de usuario gráficas (ver apartado 7.1.2).

Los pasos a seguir para la instalación en el sistema de todas las librerías comentadas en este apartado se detallarán en el apéndice A, aunque en el caso de las librerías de Qt y de la base de datos, se explicarán en los apartados 7.3.2 y 7.3.3 respectivamente.

⁶ Por convenio en C++, las funciones se suelen definir en ficheros de cabecera e implementar en ficheros de código fuente. Existe también la posibilidad de implementar las funciones en los mismos ficheros de cabecera, en cuyo caso, las funciones serán *inline*.

7.3.2 NetBeans

La implementación se realizará en un entorno de desarrollo integrado (*IDE*, por sus siglas en inglés), el cual dispone de todas las herramientas necesarias para editar código, compilar, enlazar, ejecutar y depurar el sistema durante su implementación en el mismo programa. Esto supone ahorrar tiempo en la construcción del software, ya que no es necesario un terminal en el que entrar los comandos de compilación, enlace y ejecución cada vez que se vaya a probar el programa, sino que mediante un clic se puede realizar todo el proceso. El IDE escogido ha sido *NetBeans*, ya que soporta el lenguaje de programación C++, tiene un depurador de código incorporado que permite ejecutar instrucción a instrucción, cuenta con una interfaz intuitiva y dispone de abundante documentación para su configuración. Una guía completa de configuración de *NetBeans* para trabajar en C y C++ y utilizar Qt (mediante *Qt Designer*) puede encontrarse en [22]. Lo más importante explicado en la guía mencionada es que debe activarse el módulo de C/C++ en *NetBeans* e instalar *Qt Designer*. Una forma de realizar este último paso es ejecutar los siguientes comandos en un terminal:

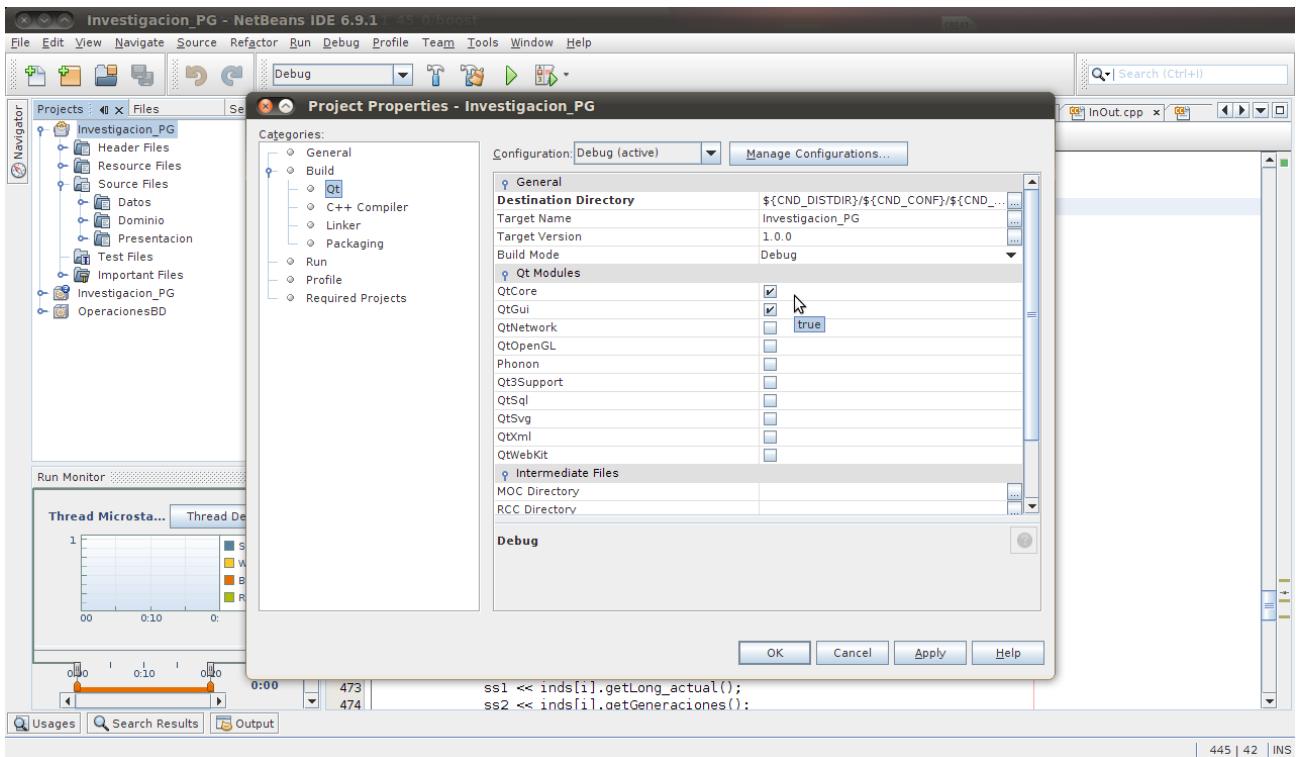
```
sudo apt-get install qt4-qmake  
sudo apt-get install qt4-designer
```

La versión de *NetBeans* utilizada para construir el software de este proyecto ha sido la 6.9.1. Una vez configurado el IDE para trabajar en C y C++, creado un proyecto de C++ con Qt tal como se menciona anteriormente y tener instaladas en el sistema las librerías explicadas en el anterior apartado en los directorios por defecto, se puede indicar al IDE las opciones de compilación y enlace que deberá tener en cuenta. Fundamentalmente estas opciones están relacionadas con las librerías a incluir en cada etapa. A continuación se detallan los pasos para realizar esta configuración:

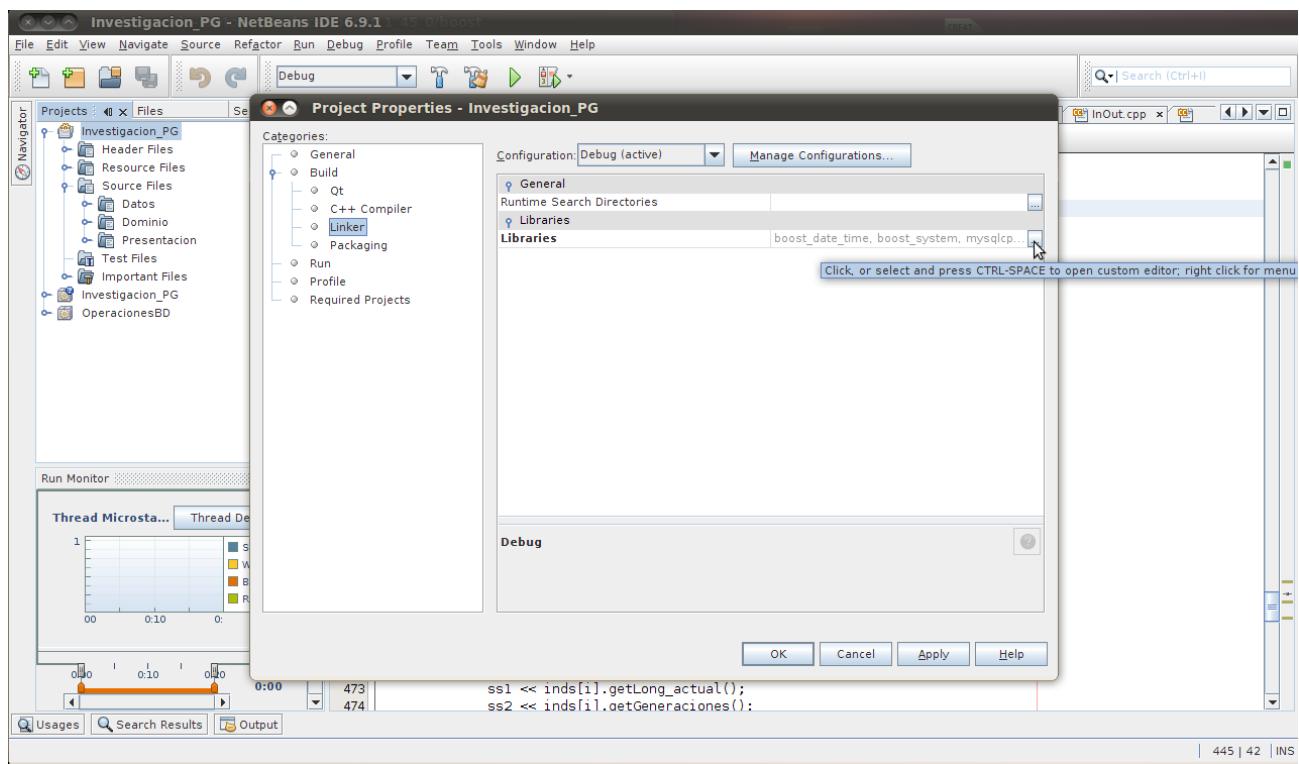
1. En el panel de navegación, clic derecho sobre el proyecto creado y selección de la opción *Properties*:

The screenshot shows the NetBeans IDE interface. The title bar reads "Investigacion_PG - NetBeans IDE 6.9.1". The menu bar includes File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, Help. The toolbar has icons for New, Open, Save, Cut, Copy, Paste, Find, Replace, and Run. The Projects panel on the left shows a project named "Investigacion_PG" with various source files like CtrlElite.cpp, CtrlPersonalizada.h, CtrlPersonalizada.cpp, Individuo.h, Individuo.cpp, InOut.cpp, and InOut.h. The code editor window displays C++ code related to database connections and buffer management. Below the code editor is the "Run Monitor" tool, which shows a thread timeline for "Microtask..." with four threads: Sleeping, Waiting, Blocked, and Running. The status bar at the bottom right shows "445 | 42 | INS".

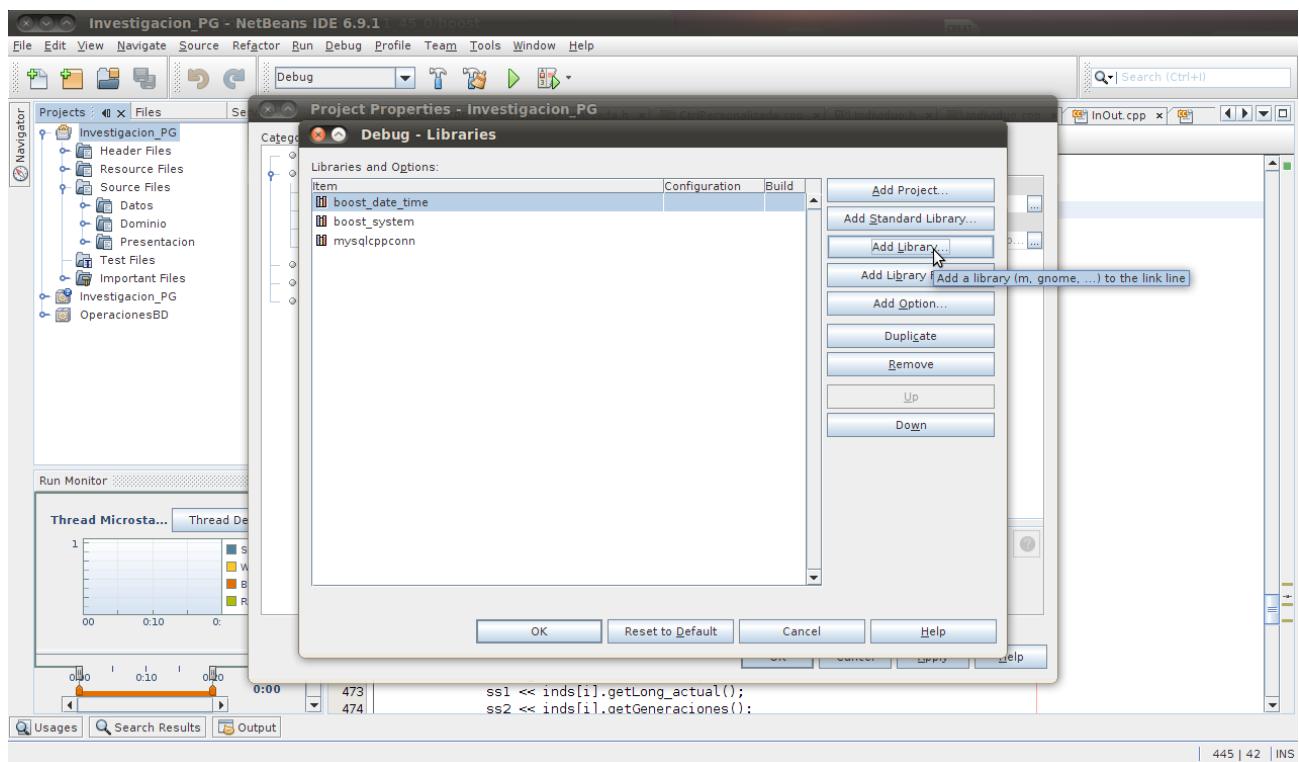
- En la categoría *Build* -> *Qt* del panel izquierdo y apartado *Qt Modules*, comprobar que *QtCore* y *QtGui* están seleccionados (por defecto lo están):



- En la categoría *Build* -> *Linker* del panel izquierdo y apartado *Libraries*, clic sobre el ícono que permite añadir librerías:

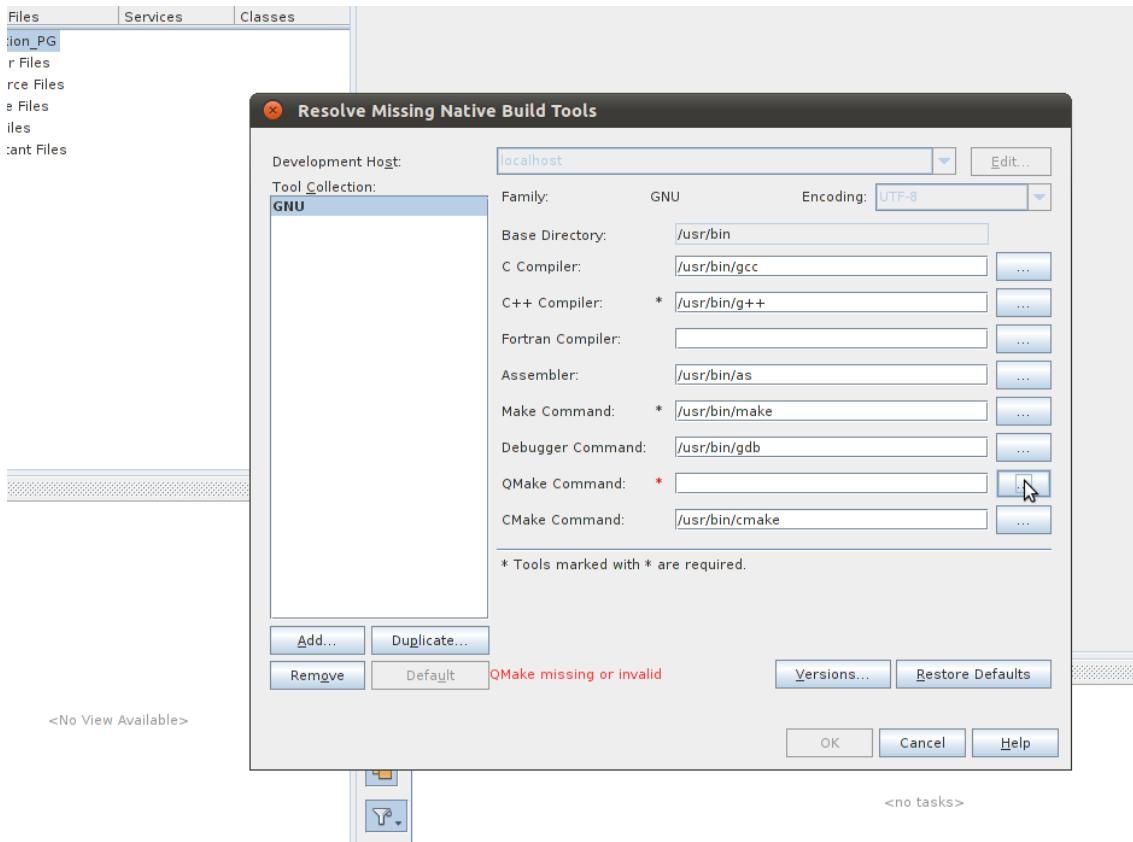


4. Clic sobre el botón *Add Library...*, y a continuación seleccionar las librerías */usr/local/lib/libboost_date_time.so*, */usr/local/lib/libboost_system.so* y */usr/local/lib/libmysqlcppconn.so* (el directorio mostrado es en el que se instalan por defecto):



Como se puede comprobar, solo se añaden las librerías dinámicas con las cuales enlazar el código del ejecutable. No se indican las librerías no dinámicas porque las librerías de *Boost* ya se instalan por defecto en uno de los directorios en los que el compilador de C++ busca ficheros para incluir al compilar: `/usr/local/include`.

Es posible que al construir el proyecto, aparezca un diálogo como el que se muestra:



Esto indica que el comando utilizado para generar el *Makefile* que contiene las reglas para compilar los ficheros con código Qt, llamado *qmake*, no ha sido detectado por *NetBeans*. En este caso hay que introducir la ruta donde se encuentra, que por defecto es `/usr/bin/qmake-qt4`. También puede suceder que la versión por defecto de *qmake* que adopta *NetBeans* no es la correcta, lo cual provoca que el proyecto no pueda ser compilado. La razón es que, en ocasiones, *NetBeans* coge por defecto la versión de *qmake* a la que apunta el enlace `/usr/bin/qmake` y que podría no ser la adecuada. Para solucionar esto, hay que seleccionar en la barra de menú *Tools -> Options -> C/C++* e introducir en el campo "QMake Command" lo siguiente: `/usr/bin/qmake-qt4`.

Una vez realizados estos pasos, ya debería ser posible construir el proyecto desde *NetBeans*.

7.3.3 MySQL Workbench

El programa utilizado para crear la base de datos y las tablas que la componen ha sido *MySQL Workbench 5.2.30*. Este programa dispone de una interfaz gráfica para manejar las bases de

datos del sistema cuyo SGBD es *MySQL*. También permite crear modelos que muestran la estructura de una base de datos (como ejemplo, ver modelo mostrado en el apartado 6.1.3) entre otras funcionalidades relacionadas.

Para poder utilizar correctamente este programa, es necesario tener instalada en el sistema (o en un sistema remoto y accesible mediante una red) una base de datos *MySQL*. Para instalar la base de datos mencionada en el sistema es necesario tener conexión a Internet y ejecutar el siguiente comando en un terminal:

```
sudo apt-get install mysql-server mysql-client libmysqlclient16-dev
```

Al ejecutar el comando mostrado, el sistema pedirá introducir una contraseña para el usuario “root”, que tiene permisos de administrador de la base de datos. IMPORTANTE: El código del programa creado para este proyecto tiene grabados en el código (fichero *Datos/ConexionBD.cpp*) los parámetros de acceso a la base de datos siguientes:

- URL: `tcp://127.0.0.1:3306`
- Usuario: root
- Contraseña: root
- Nombre de la base de datos: PFC_Programacion_Genetica

Si se configura la base de datos con otros parámetros, es necesario modificar el código del programa para que funcione correctamente.

Los pasos a seguir para configurar la base de datos utilizada en este proyecto desde *MySQL Workbench* son los siguientes:

1. Descargar el programa de <http://www.mysql.com/downloads/workbench/5.2.html>. Es necesario seleccionar la versión correcta, en este caso sería *Ubuntu Linux* y versión 10.04 de 32 bits:

MySQL Workbench 5.2.37

Select Platform:

Ubuntu Linux

Ubuntu Linux ver. 11.04 (x86, 64-bit), DEB	5.2.37	18.2M	Download
(mysql-workbench-gpl-5.2.37-1ubu1104-amd64.deb)		MD5: 2d910400ce1a5c86f7f1a4123156a9a2	
Ubuntu Linux ver. 11.04 (x86, 32-bit), DEB	5.2.37	18.0M	Download
(mysql-workbench-gpl-5.2.37-1ubu1104-i386.deb)		MD5: 8c690cb92ef6715ac91fea0e6526ef06	
Ubuntu Linux ver. 10.04 (x86, 32-bit), DEB	5.2.37	17.9M	Download
(mysql-workbench-gpl-5.2.37-1ubu1004-i386.deb)		MD5: 6ef662146d662133d036dd06f65b30a3	
Ubuntu Linux ver. 10.04 (x86, 64-bit), DEB	5.2.37	18.1M	Download
(mysql-workbench-gpl-5.2.37-1ubu1004-amd64.deb)		MD5: 2a3d981715af487ef268743f35c21a1c	



We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download.

2. Una vez descargado, abrir el archivo e instalarlo siguiendo los pasos.
3. A continuación, abrir el programa y crear una nueva instancia del servidor de la base de datos:

Welcome to MySQL Workbench

→ [What's New in This Release?](#)
Read about all changes in this MySQL Workbench release.

Workspace

SQL Development
Connect to existing databases and run SQL Queries, SQL scripts, edit data and manage database objects.

Data Modeling
Create and manage models, forward & reverse engineer, compare and synchronize schemas, report.

Server Administration
Configure your database server, setup user accounts, browse status variables and server logs.

New Connection
Add a new database connection for querying.

Edit Table Data
Select a connection and schema table to edit.

Edit SQL Script
Open an existing SQL Script file for editing.

Manage Connections
Modify connection settings or add connections.

Create New EER Model
Create a new EER Model from scratch.

Create EER Model From Existing Database
Create by connecting and reverse engineering.

Create EER Model From SQL Script
Import an existing SQL file.

New Server Instance
Register a new server instance to manage.

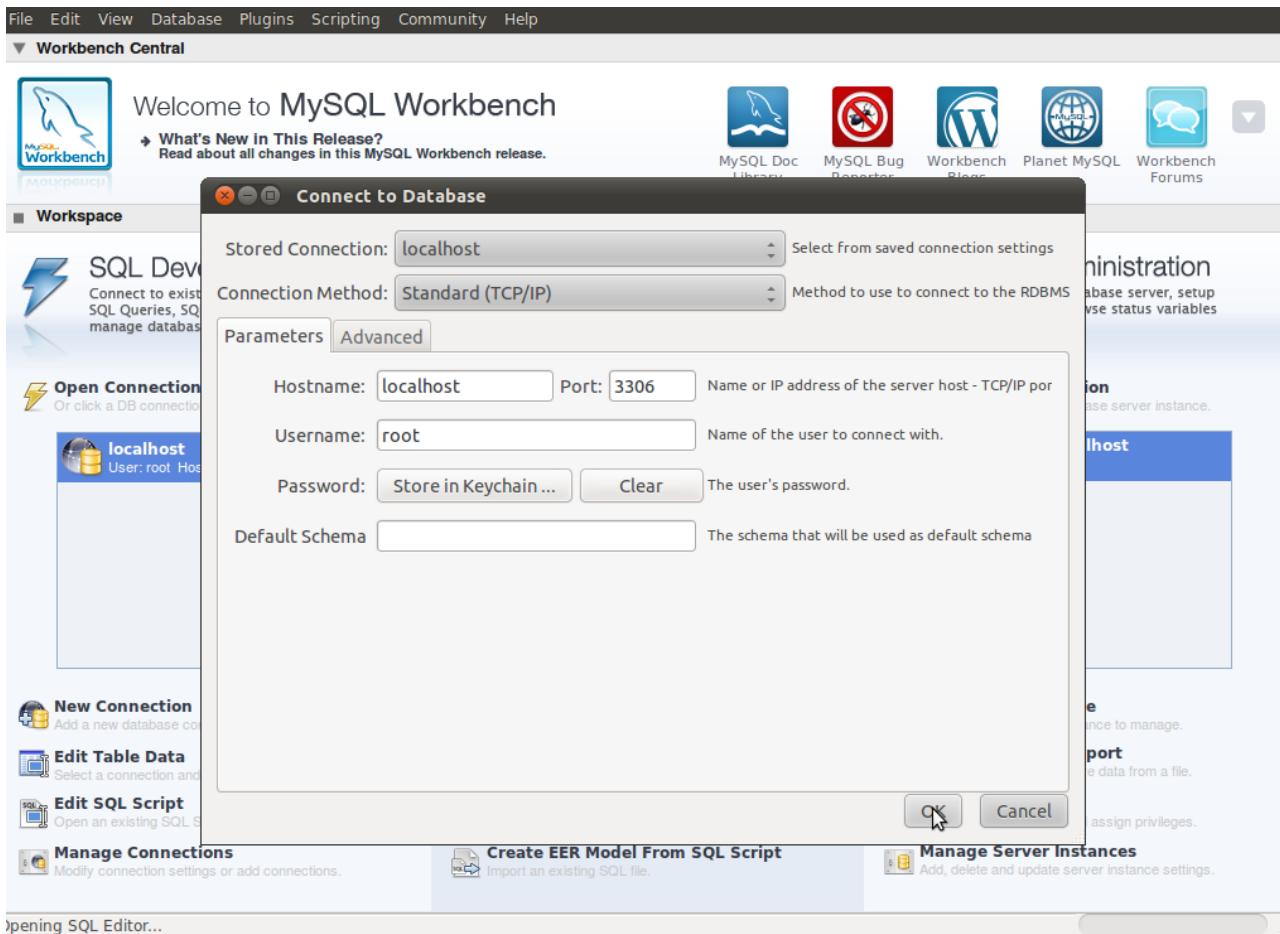
Manage Import / Export
Create a dump file or restore data from a file.

Manage Security
Manage user accounts and assign privileges.

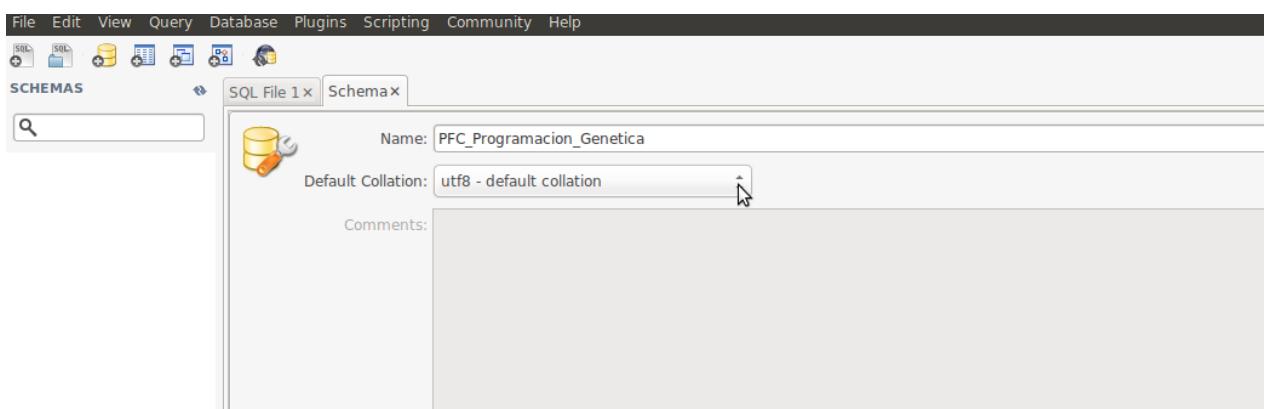
Manage Server Instances
Add, delete and update server instance settings.

new server instance creation cancelled

4. Seguir los pasos para crear la nueva instancia. El nombre que se le dé a la instancia no es importante.
5. Abrir una conexión con la instancia creada:



6. Crear un nuevo *schema* o base de datos con nombre *PFC_Programacion_Genetica* y codificación de caracteres *utf8 – default collation*:



7. En este momento ya se pueden crear las tablas que componen la base de datos. Hay que copiar el contenido del fichero CreacionBD.sql, proporcionado con los ficheros que contienen el programa implementado, en el editor SQL y ejecutar las instrucciones copiadas para crear todas las tablas:

The screenshot shows the MySQL Workbench interface. The top menu bar includes File, Edit, View, Query, Database, Plugins, Scripting, Community, and Help. Below the menu is a toolbar with various icons. On the left, a 'SCHEMAS' panel shows the 'PFC_Programacion_Gen' schema with Tables, Views, and Routines. The main area is a 'SQL File 1*x' editor containing the following SQL code:

```

157 PRIMARY KEY (`indice`, `ticker`, `fecha`, `idseleccionado`),  

158 KEY `fk_validacion-ind_registro`(`ticker`, `fecha`),  

159 KEY `fk_validacion-ind_seleccionado`(`idseleccionado`),  

160 CONSTRAINT `fk_validacion-ind_registro` FOREIGN KEY  

161 CONSTRAINT `fk_validacion-ind_seleccionado` FOREIGN KEY  

162 ) ENGINE=InnoDB DEFAULT CHARSET=utf8$$  

163  

164 • CREATE TABLE `validacion-pobl` (  

165     `indice` varchar(45) NOT NULL,  

166     `ticker` varchar(45) NOT NULL,  

167     `fecha` date NOT NULL,  

168     `nombre` varchar(45) NOT NULL,  

169     PRIMARY KEY (`indice`,`ticker`,`fecha`,`nombre`),  

170     KEY `fk_validacion-pobl_registro`(`ticker`, `fecha`),  

171     KEY `fk_validacion-pobl_elite`(`nombre`),  

172     CONSTRAINT `fk_validacion-pobl_elite` FOREIGN KEY  

173     CONSTRAINT `fk_validacion-pobl_registro` FOREIGN KEY  

174 ) ENGINE=InnoDB DEFAULT CHARSET=utf8$$  

175  

176 • CREATE TABLE `valor` (  

177     `ticker` varchar(45) NOT NULL,  

178     `indice` varchar(45) NOT NULL DEFAULT '',  

179     `nombre` varchar(45) DEFAULT NULL,  

180     PRIMARY KEY (`ticker`,`indice`),  

181     KEY `fk_valor_indice`(`indice`),  

182     CONSTRAINT `fk_valor_indice` FOREIGN KEY (`indice`)  

183 ) ENGINE=InnoDB DEFAULT CHARSET=utf8$$  

184  

185 • SET FOREIGN_KEY_CHECKS=1$$

```

The 'Object Info' tab is selected in the bottom-left panel. A 'Snippets' panel on the right shows a 'My Snippets' section with several icons. The status bar at the bottom left says 'SQL Editor Opened.'

Referencias utilizadas en el capítulo: [19], [20], [21], [22], [23], [24], [25], [27].

PARTE 3: EXPERIMENTOS

8. EVALUACIÓN DE LAS REGLAS DE INVERSIÓN

En este capítulo se llevarán a cabo dos tipos de experimentos para comprobar el comportamiento de las reglas de inversión generadas por el sistema software construido en este proyecto.

En la literatura pueden encontrarse gran cantidad de experimentos centrados en los beneficios obtenidos por conjuntos de reglas de inversión generadas y testeadas con el mismo valor de coste de transacción. La razón de ello es que se asume que el algoritmo de programación genética optimiza las reglas para que obtengan los máximos beneficios si se aplican con esos mismos costes en cualquier otro periodo distinto del de generación. Por contra, la mayoría de artículos se centran en optimizar el resto de parámetros del algoritmo e incluso buena parte de estos trabajos no considera los costes de transacción.

Un grupo de experimentos realizados en este capítulo están relacionados con la influencia que tienen los costes de transacción aplicados tanto en la generación como en el periodo de test en los beneficios dados y en el número de transacciones realizadas en un periodo de tiempo dado. Como se ha comentado en el anterior párrafo, esta relación no ha sido estudiada detenidamente en la literatura, y por ello es desconocido el valor óptimo que debe tener el parámetro “coste de transacción” en la generación de un conjunto de reglas para obtener el máximo beneficio en el caso de que a un inversor se le aplique un determinado coste en la realidad. También se desconoce de qué manera afectan los costes aplicados en la generación de reglas con el número de operaciones realizadas por éstas en el mercado, ni tampoco si estos dos conceptos tienen relación entre ellos (aunque en un primer momento pueda parecer obvio que la tengan).

El otro conjunto de experimentos tratarán de relacionar la efectividad de las reglas generadas con la tendencia del mercado en los periodos de generación y de test. Algunos autores han hecho referencia a esta relación en sus artículos [1], aunque no la han estudiado de forma detenida, sino que la han señalado como una razón que pudiera explicar el comportamiento de las reglas durante algún periodo determinado.

La mayoría de artículos de la literatura usan costes de hasta 0,25% por cada transacción de compra o venta. Utilizando estos costes (varían dependiendo del autor), la literatura indica que se puede superar a la estrategia de inversión *buy and hold*. No obstante, los parámetros y los indicadores de análisis técnico utilizados por el algoritmo de generación de reglas difieren entre los artículos. En este proyecto se utilizan un subconjunto reducido y común en todos los artículos, con lo que posiblemente se podrían obtener mejores resultados ampliando el número de indicadores.

Debe tenerse en cuenta que todos los beneficios mostrados son respecto a *buy and hold*, a menos que se indique lo contrario.

8.1 Entorno de la experimentación

La experimentación se ha realizado con dos índices: el IBEX 35 (IBEX) y el Dow Jones Industrial (DJI). El primero es el principal índice español, mientras que el segundo es uno de los mayores y más antiguos índices estadounidenses. Como se ha comentado en el capítulo 3, las estrategias de inversión pasivas, las cuales utilizan la metodología *buy and hold*, usualmente invierten en una cesta de acciones que replica el índice al que pertenecen. Por este motivo, hacer la experimentación tomando como referencia el valor de cotización de un índice en lugar del precio de una acción concreta, permite comprobar si las reglas obtenidas, que gestionan la cartera activamente, mejoran a estrategias pasivas y a *buy and hold*.

Se ha escogido el método llamado *rolling forward*, utilizado ampliamente en la literatura, para llevar a cabo los experimentos con ambos índices. Este método consiste en disponer de más de una secuencia de entrenamiento – validación – test y organizar cada una de forma que se superponga parcialmente en el tiempo con otras. En la figura siguiente se muestran las secuencias que han sido utilizadas para realizar los experimentos en este capítulo:

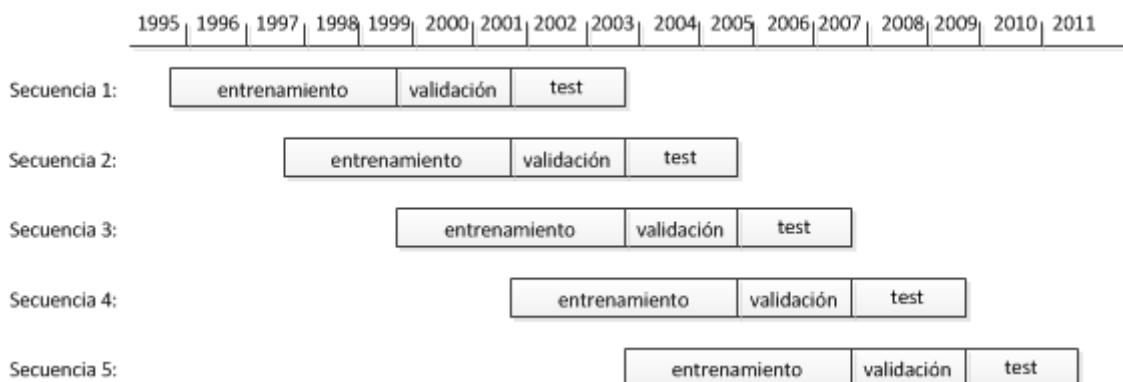


Figura 30: Representación del método "rolling forward" usado en la experimentación.

Como se puede comprobar, en estos experimentos se han escogido cinco secuencias cuyos períodos de entrenamiento, validación y test tienen una duración de 4, 2 y 2 años respectivamente, tal y como se hace en la mayoría de artículos relacionados con este tema (ver capítulo 4.3). Cada secuencia comienza el 1 de agosto del año indicado y dos años más tarde que la secuencia anterior, con lo cual ambas se superponen 6 años. Utilizando este método es posible comprobar si los resultados son consistentes en el tiempo, es decir, si a partir de unos datos iniciales parecidos, los resultados obtenidos con ellos son también parecidos. Si esto es cierto, entonces se podría llegar a la conclusión de que existen patrones detectables en los precios mediante indicadores de análisis técnico y que estos patrones se repiten con cierta frecuencia en el tiempo y tienen una duración elevada. En caso de que no se produzca este caso, no se puede determinar si existen patrones detectables o si existen pero cambian con una elevada frecuencia (en un tiempo menor a la duración de los períodos de entrenamiento y validación). En todo caso, verificar esta hipótesis no es un objetivo de los experimentos llevados a cabo en este proyecto.

El método *rolling forward* también tiene la ventaja de permitir experimentar con históricos de precios con una duración relativamente corta, ya que los periodos de experimentación se superponen y se reutilizan sus datos. Encontrar históricos con la suficiente cantidad de datos y disponibles públicamente sin coste resulta complicado, y los índices escogidos en estos experimentos son de los pocos que disponen de ellos.

Durante los períodos de test, se tomarán las decisiones de inversión según el voto mayoritario del conjunto de reglas con el que se realiza la experimentación. Esto quiere decir que se estará dentro del mercado siempre que, como mínimo, el 50% de las reglas de la población indiquen estar dentro; se estará fuera del mercado en caso contrario. Cada transacción de compra o venta en el periodo de test (y también en los períodos utilizados en la generación de las reglas) se realiza con el 100% del capital disponible para invertir.

Los parámetros con los que se ha configurado el algoritmo son similares a los indicados en el artículo de Allen y Karjalainen, aunque con algunas variaciones. Fundamentalmente estas variaciones están relacionadas con la reducción del tamaño máximo que puede llegar a tener un individuo (número de nodos) y el tamaño de las poblaciones. Estos dos parámetros son los principales responsables del tiempo de ejecución del algoritmo, y, a causa de que con los valores indicados en el artículo la ejecución resulta excesivamente lenta, se han reducido de 500 a 200 individuos cada población y de 100 a 80 el número máximo de nodos por individuo. A cambio se ha aumentado el número de máximo de generaciones durante las que se ejecuta el algoritmo de 50 a 100 y el de generaciones sin mejora de 25 a 50, ya que se ha observado que en algunos casos el número de generaciones ejecutadas no es suficiente y sería posible obtener mejores individuos ejecutando algunas más. Por otra parte, se ha reducido el número de iteraciones ejecutadas (reglas de élite obtenidas en cada población) de 100 a 50, ya que esta es una muestra suficientemente representativa y con ello el tiempo de generación de las reglas se reduce a la mitad. En cuanto a las probabilidades de los operadores, se ha reducido la probabilidad de cruce del 100% al 90% y se ha introducido el operador de mutación con una probabilidad del 5%. Estas probabilidades cambian considerablemente dependiendo del artículo, con lo cual no parecen determinantes. En el cuadro siguiente se pueden ver los parámetros utilizados:

Parámetros del algoritmo	
Probabilidad de cruce	90%
Probabilidad de mutación	5%
Iteraciones	50
Tamaño de la población de individuos	200
Máximo número de generaciones	100
Número de generaciones sin obtener mejora	50
Número máximo de nodos por individuo	80

Los parámetros son similares en la mayor parte de artículos que tratan este tema así como en el libro de Koza. El no haber un consenso sobre el valor exacto de los parámetros a utilizar, sino aproximaciones, parece indicar que pequeñas variaciones en éstos no son importantes para que el algoritmo de programación genética produzca mejores resultados.

El espacio de búsqueda del problema es el número de soluciones que pueden ser formadas, o, en términos del problema, el número de individuos (árboles) representables. Con el número máximo de nodos por individuo indicado en la anterior tabla y teniendo en cuenta que hay 16 tipos de nodos (el tipo de nodo *constante* se cuenta como 3 distintos, ya que hay que escogerlo entre booleano, entero o decimal) es posible aproximarse al valor que tendría el espacio de búsqueda.

Estimar una cota superior para el espacio de búsqueda del algoritmo no resulta complicado, y mediante el siguiente razonamiento se puede conseguir. Cada individuo dispone de un vector donde guardar todos sus nodos, cada uno ocupando una posición. Este vector siempre contiene los nodos en sus primeras posiciones, de forma que si en una posición no hay ningún nodo, en las siguientes tampoco habrá ninguno. Por tanto, una cota superior del número de árboles que pueden ser formados es:

$$\text{espacio de búsqueda} < \sum_{i=1}^p n^i = \frac{n^{p+1} - n}{n - 1}$$

Donde p es el número de posiciones del vector (máximo número de nodos por individuo) y n es el número de tipos de nodo disponibles. La fórmula indica que la complejidad del problema es exponencial al número máximo de nodos. Para los experimentos llevados a cabo, la cota superior tiene un valor de $2,85 \cdot 10^{30}$. No obstante, en el cálculo no se tiene en cuenta el número de descendientes que debe tener cada tipo de nodo (ver tabla siguiente), lo cual limita los tipos de nodos permitidos en las últimas posiciones del árbol.

Número de descendientes	Número de tipos de nodo disponibles
0	2
1	4
2	7
3	1

Tampoco se tiene en cuenta la limitación que impide combinar nodos cuyos descendientes no sean del tipo adecuado, ni la restricción que indica que el primer nodo del vector siempre ha de ser de tipo booleano. Es por esto que en realidad el espacio de búsqueda es algunos órdenes de magnitud inferior al resultado mostrado.

8.2 Análisis de beneficios

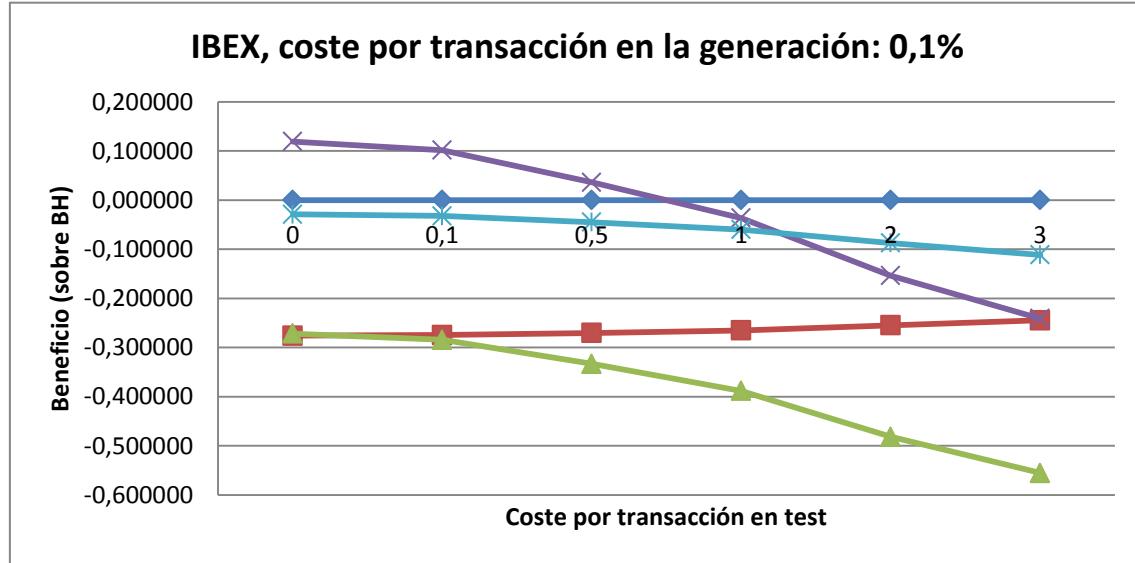
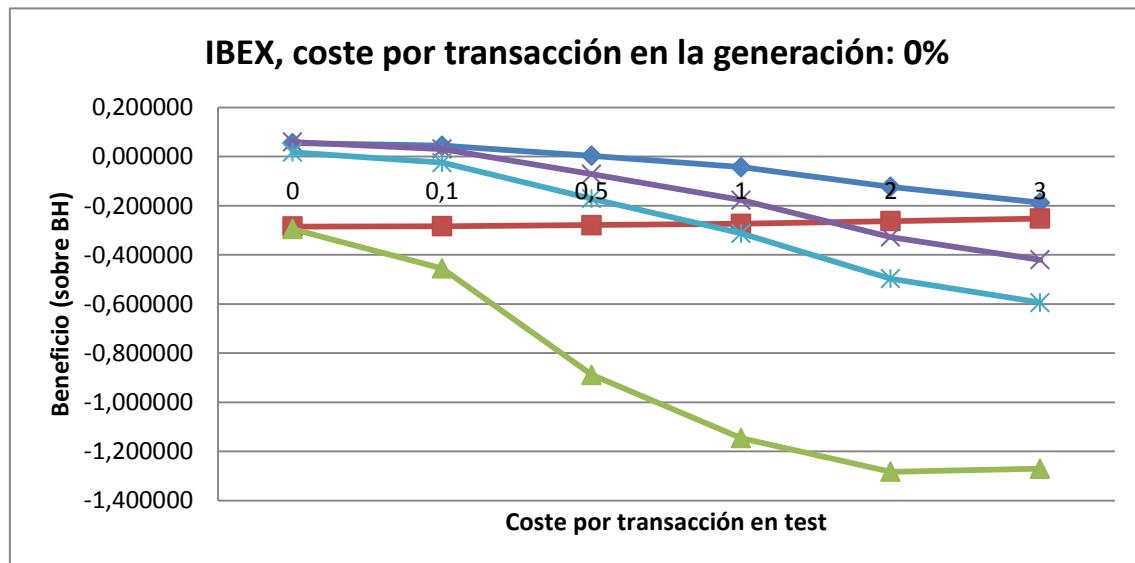
El objetivo de este análisis es determinar cómo se comportan las reglas con diferentes costes por transacción, tanto en los períodos de generación como en los de test, así como el número de transacciones realizadas por éstas.

El comportamiento de las reglas según diferentes costes por transacción se puede observar representando los resultados en un gráfico, donde los beneficios en el periodo de test (en tanto por uno) se representan en el eje Y y los costes en el mismo periodo en el eje X. Para probar

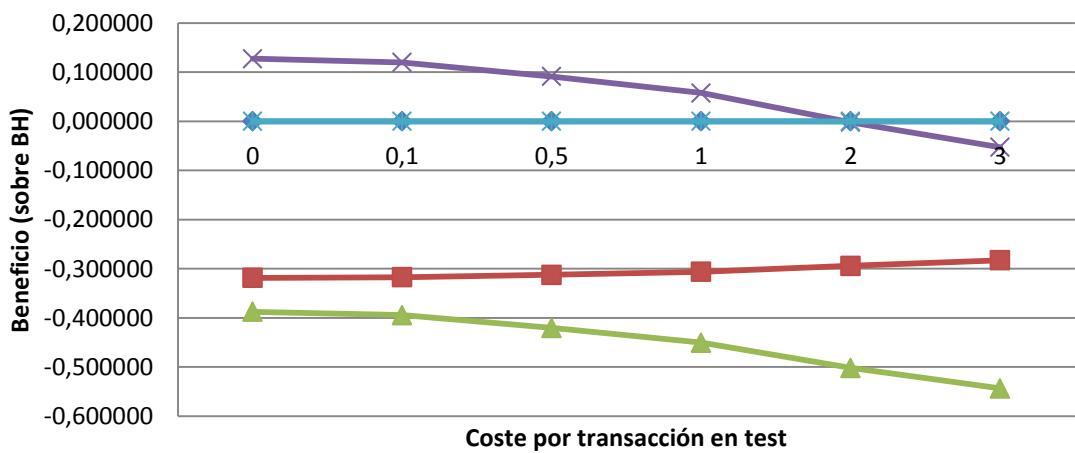
todas las combinaciones entre costes aplicados a la generación de las reglas y al test hay que generar tantos gráficos como costes utilizados en la generación de las reglas. En cada gráfico se representan los resultados obtenidos para cada secuencia indicada en la figura 30:

LEYENDA

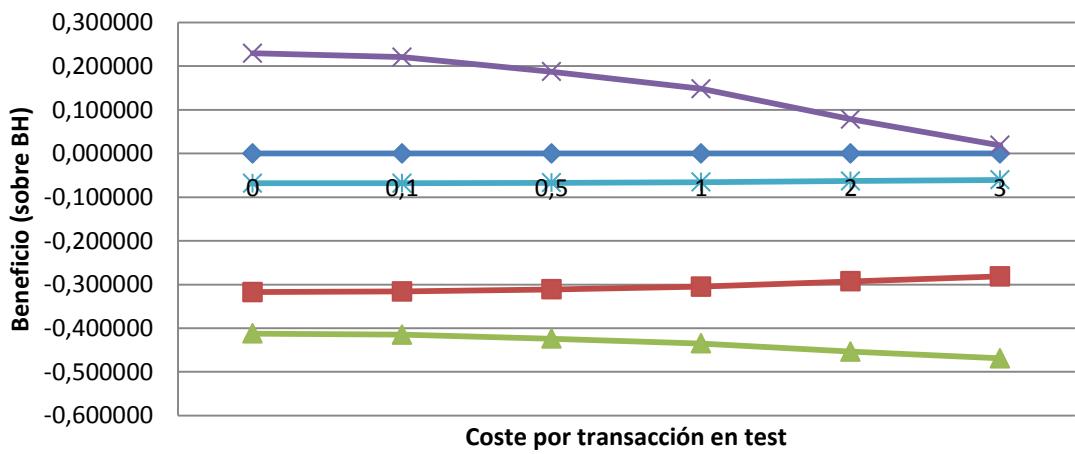
- Secuencia 1
- Secuencia 2
- ▲— Secuencia 3
- ×— Secuencia 4
- *— Secuencia 5



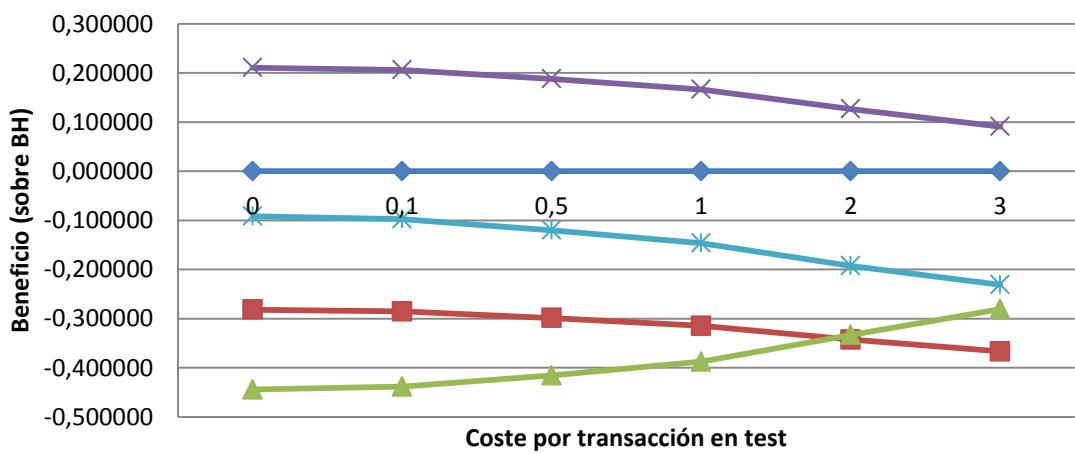
IBEX, coste por transacción en la generación: 0,5%



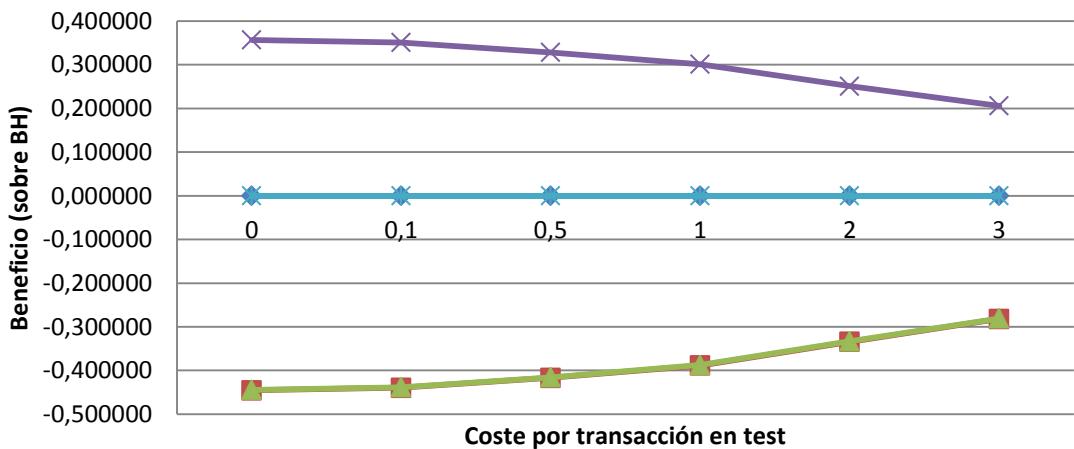
IBEX, coste por transacción en la generación: 1%



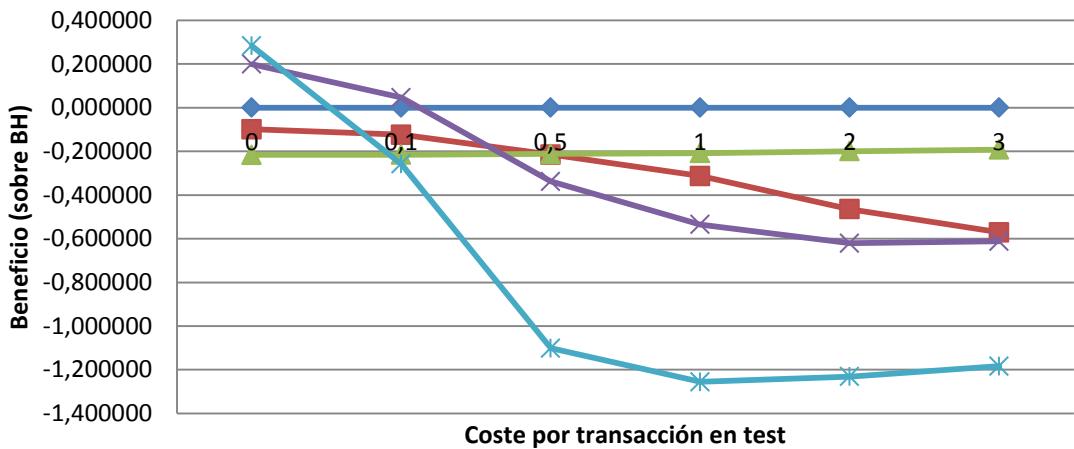
IBEX, coste por transacción en la generación: 2%



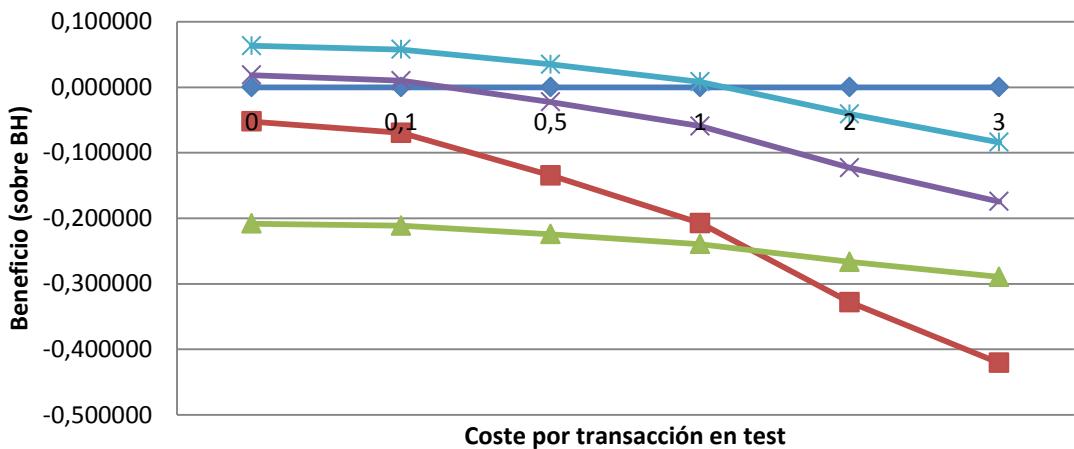
IBEX, coste por transacción en la generación: 3%



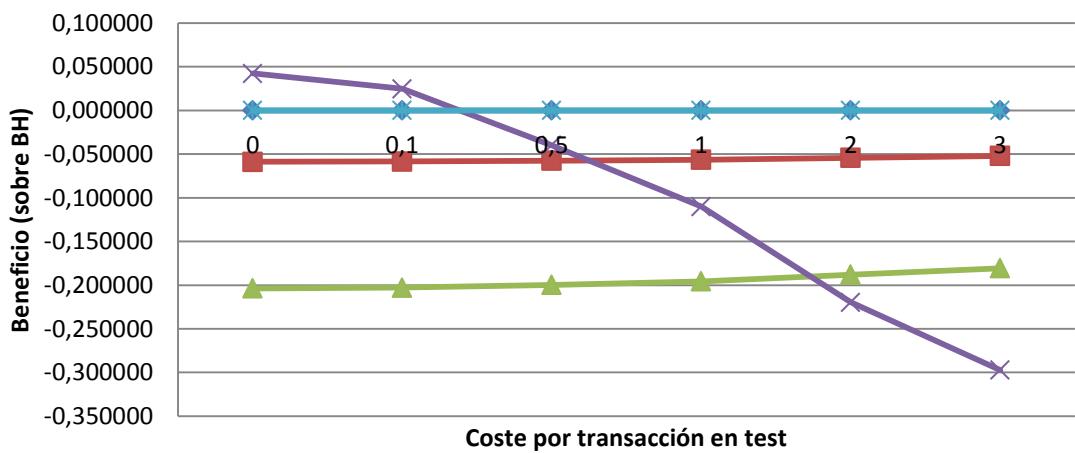
DJI, coste por transacción en la generación: 0%



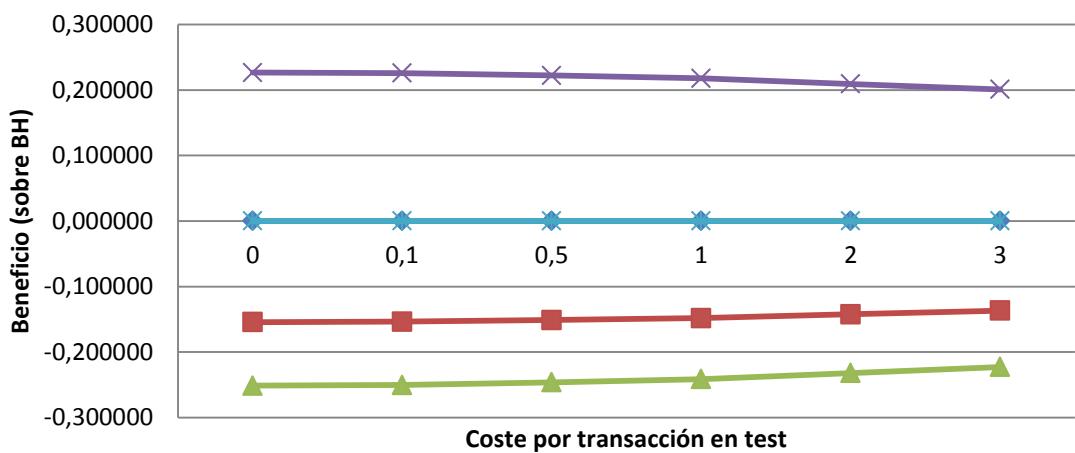
DJI, coste por transacción en la generación: 0,1%



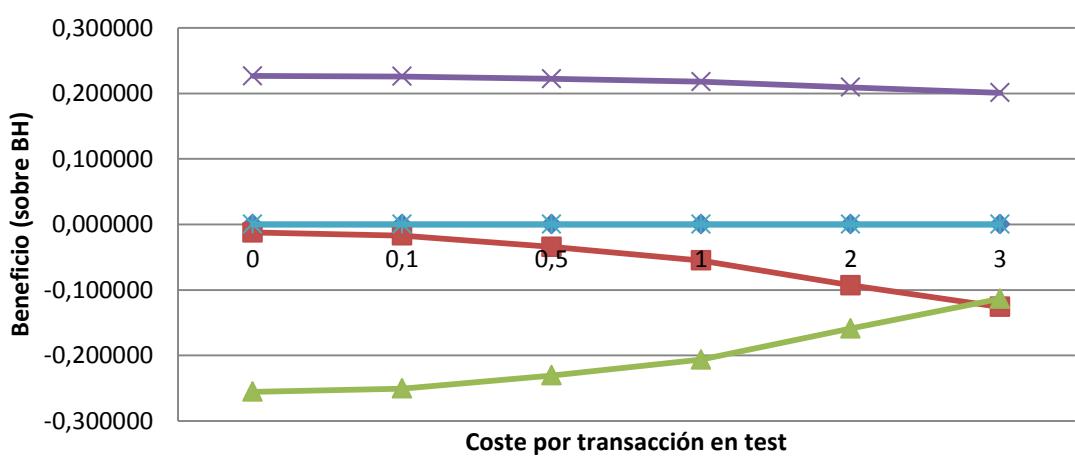
DJI, coste por transacción en la generación: 0,5%

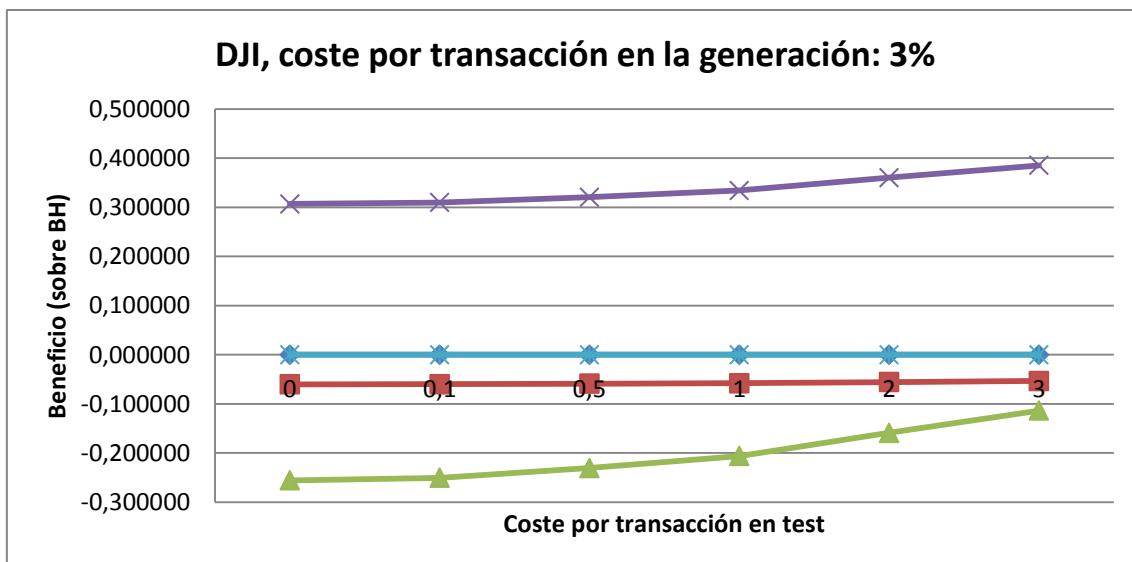


DJI, coste por transacción en la generación: 1%



DJI, coste por transacción en la generación: 2%





En los gráficos se puede observar cómo la pendiente descendente de los beneficios se suaviza a medida que los costes utilizados en la generación aumentan. Esto indica que, como cabía esperar, las reglas se adaptan a los costes con las que son generadas, es decir, intentar obtener los mayores beneficios teniendo en cuenta los costes por transacción indicados. También se puede observar que, el uso de reglas generadas con unos costes de transacción mayores a los usados en el test no obtienen beneficios considerablemente superiores a los obtenidos en el periodo de test con los mismos costes que en la generación. Dicho de otra forma, las reglas se adaptan para obtener el máximo beneficio para el nivel de costes usados en la generación e inferior, siendo similar el beneficio con los costes de generación y con costes inferiores.

En todos los costes de generación las reglas se comportan de forma consistente, es decir, las que superan a la estrategia *buy and hold* lo hacen para todos los costes de generación en una magnitud similar y lo mismo sucede para las que no superan a dicha estrategia. Esto es importante para comprobar que el algoritmo siempre llega a unos resultados similares con una configuración de parámetros dada, ya que cada línea de beneficios es el resultado de generar una población de reglas escogiendo las 50 mejores de sendas ejecuciones del algoritmo, con cada ejecución inicializada con una población aleatoria.

En algunos casos, la pendiente de beneficios es ascendente en lugar de ser descendente. Esto ocurre porque las reglas indican estar fuera del mercado durante todo el periodo de test, con lo cual no realizan ninguna transacción. Al no aplicar ningún coste por transacción sobre dichas reglas, su beneficio sobre *buy and hold* aumenta a medida que los costes por transacción en el periodo de test también lo hacen. Esto es porque la estrategia *buy and hold* siempre hace dos transacciones, una de compra al inicio del periodo y otra de venta al final, a las que son aplicados los costes por transacción indicados, lo que repercute en su beneficio absoluto. Este caso se da con más frecuencia a medida que los costes en la generación de las reglas aumentan a causa de que es más complicado sacar beneficio haciendo operaciones en el mercado.

También es posible observar que las pendientes de los beneficios de todas las secuencias aparecen en el mismo orden tanto en DJI como en IBEX, y además tienen una progresión similar en ambos índices; no obstante difieren en la magnitud de los beneficios. Teniendo en cuenta que los precios de ambos índices están fuertemente correlacionados [8], resulta lógico que los resultados dados por las reglas también sean similares. En las siguientes figuras se puede observar a simple vista la correlación que tienen ambos índices:



Figura 31: Histórico de precios del índice IBEX 35.

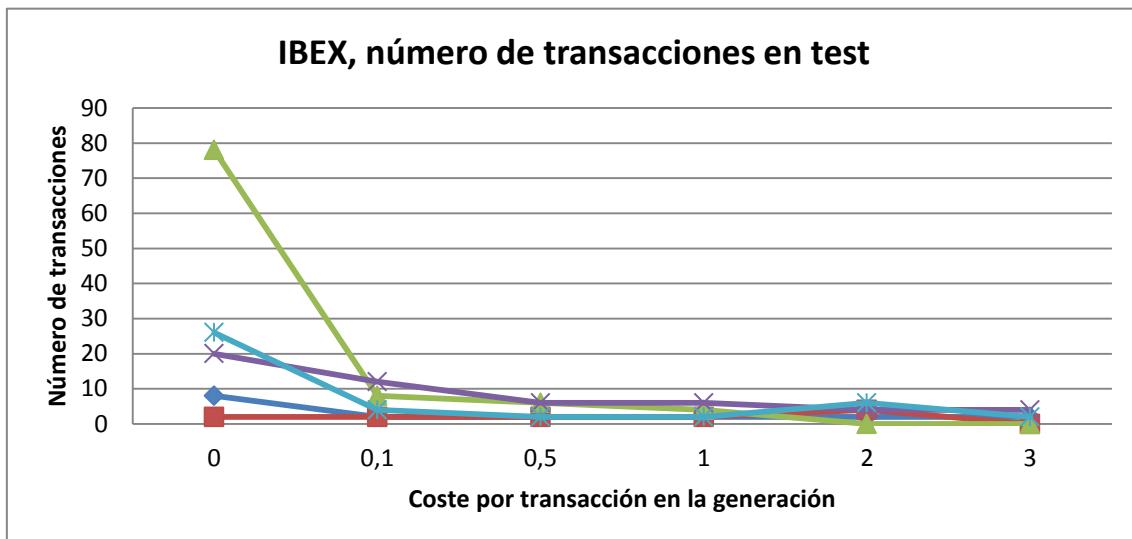


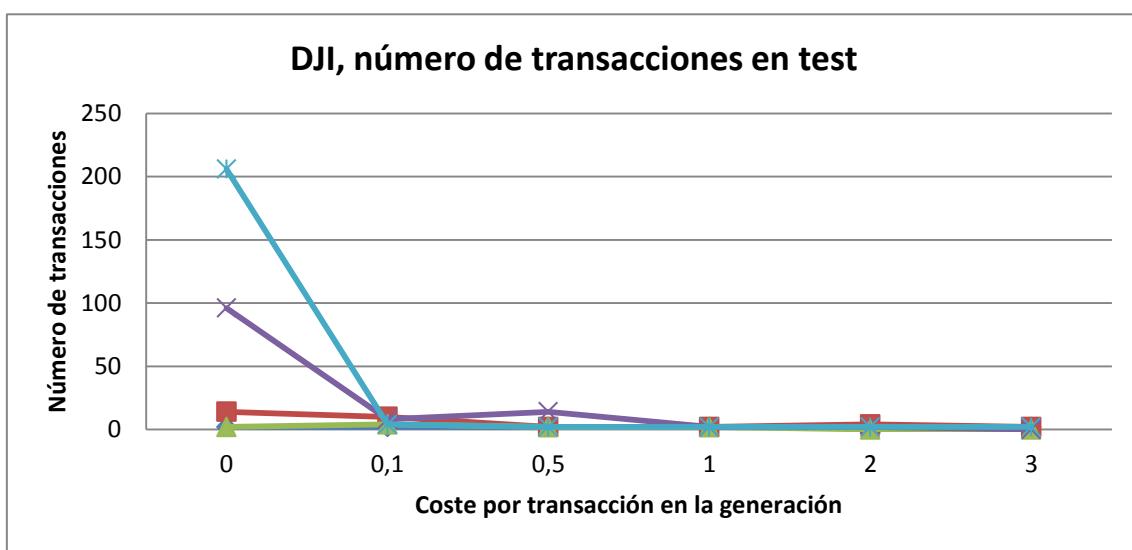
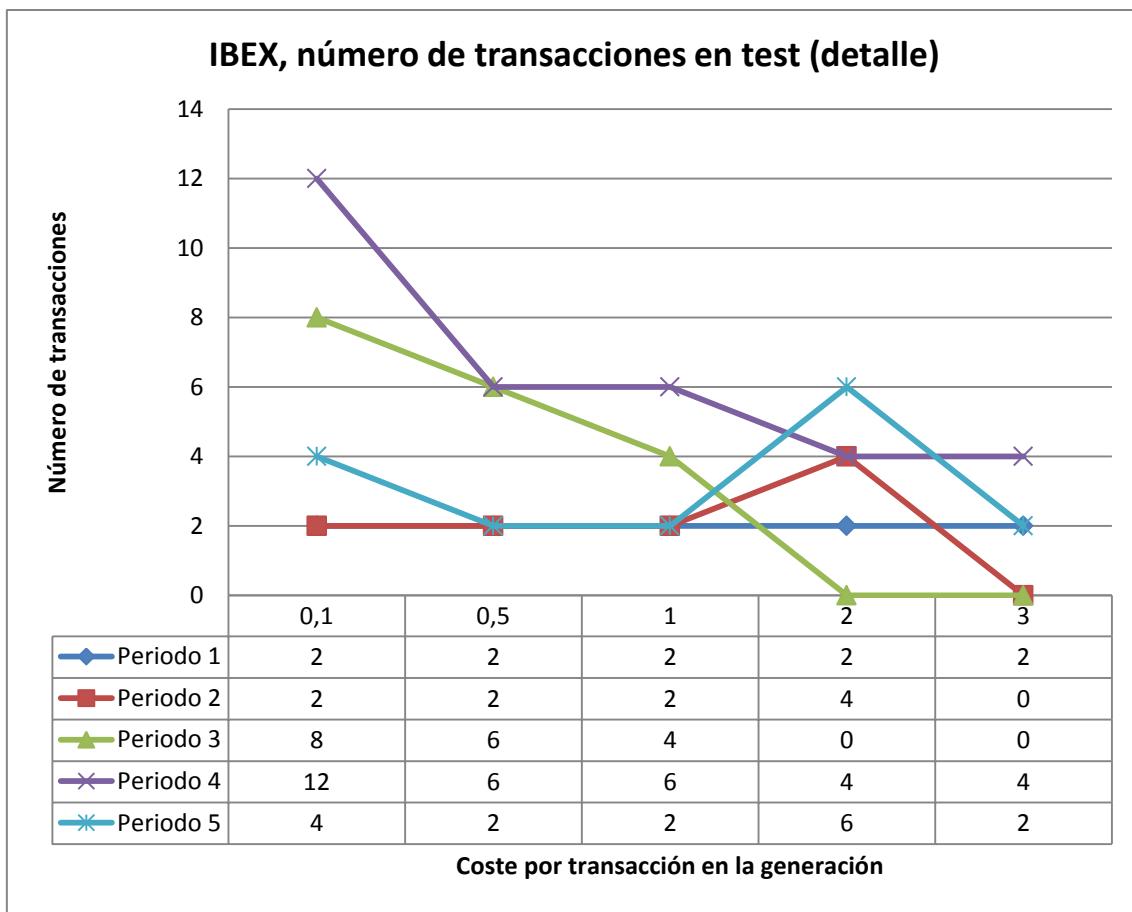
Figura 32: Histórico de precios del índice Dow Jones Industrial.

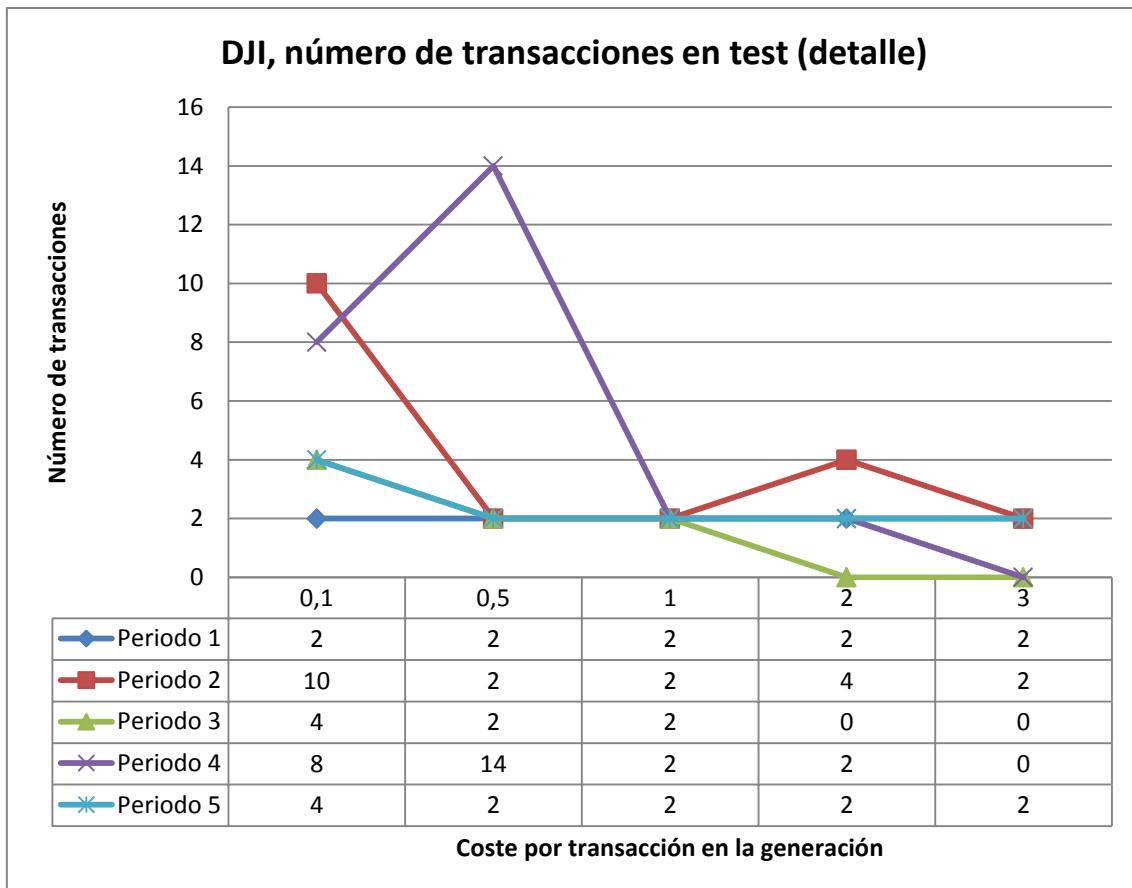
Por último, parece que los costes por transacción más adecuados durante la generación de las reglas son altos, entorno al 3%, aunque en estos experimentos éste es el valor máximo con el que se han realizado y el valor correcto puede resultar más elevado. Esto sucede para casi todas las secuencias, excepto para la número 2 y 3 en IBEX y la número 3 en DJI. De todas formas, las secuencias 2 y 3 resultan en beneficios muy negativos para todos los costes de generación y de test en ambos índices. Por lo general las reglas se suelen comportar ligeramente peor en IBEX que en DJI.

En cuanto al riesgo asumido por las reglas, se puede estimar contando el número de transacciones efectuadas, ya que, a mayor riesgo, mayor será el número de veces que una regla intente entrar en el mercado buscado sacar beneficio de la operación. Por el contrario, a menor riesgo, más certeza ha de tener la regla de que la operación a realizar será provechosa y menores serán las entradas en falso (operaciones con pérdidas), puesto que los costes de transacción harían disminuir los beneficios en el periodo de estudio en caso de realizar una operación con pérdidas.

En el conjunto de gráficos anteriores se puede observar que la pendiente de los beneficios se suaviza a medida que los costes por transacción en la generación aumentan. Esto implica que el número de transacciones efectuadas por las reglas también baja, ya que, al aumentar el coste aplicado en el periodo de test, los beneficios no se reducen de forma tan acusada como lo hacen con costes de generación inferiores. Esto indica que las reglas tienen en cuenta los costes que se indican en los periodos de generación y son generadas para obtener el máximo beneficio con ellos (el riesgo de entrar en el mercado y realizar una operación que resulte en pérdidas es más elevado a medida que los costes por transacción aumentan). Para comprobar que esto es así, a continuación se muestran el número de transacciones efectuadas en cada periodo de test y el coste por transacción utilizado en la generación.







8.3 Análisis de la efectividad según la tendencia

Este análisis tiene como objetivo relacionar la tendencia en el mercado durante los períodos de entrenamiento, validación y test con el beneficio obtenido por las reglas. La tendencia en el mercado puede resultar un factor importante para explicar por qué algunas secuencias del experimento consiguen buenos resultados y otras no, ya que dependiendo de la tendencia el mercado se comporta de forma diferente [9]. Las tendencias alcistas se producen en fases de bonanza en la economía, en las cuales el optimismo de los inversores sobre el estado económico es alto, hay una baja volatilidad en los precios del mercado y éstos suben lenta pero constantemente. Las tendencias bajistas se producen cuando las expectativas sobre la economía son malas, lo que hace aumentar la preocupación y el nerviosismo entre los inversores, que, a su vez, provoca una alta volatilidad en los precios y una bajada de éstos, en ocasiones de forma súbita.

Las tendencias del mercado se pueden identificar observando a las figuras 31 y 32. Hasta principios del año 2000 se puede observar una tendencia alcista muy fuerte en ambos índices. A continuación se inicia una tendencia bajista que dura hasta principios de 2003, en donde se vuelve a iniciar una tendencia alcista, muy fuerte en el caso del IBEX, hasta mediados del año 2007. En ese momento, de nuevo comienza una tendencia bajista muy acusada que dura hasta principios del año 2009. A partir de aquí, por primera vez, los índices difieren: DJI inicia una nueva tendencia alcista, mientras que IBEX no tiene una tendencia clara (también denominada

tendencia plana). A continuación se muestra una tabla con un resumen sobre la tendencia existente en cada secuencia de los experimentos y el resultado dado en el periodo de test (el símbolo $\times 2$ se refiere a que la duración de la tendencia es como mínimo el doble en comparación con la otra tendencia indicada; cuando hay una tendencia mixta, es decir, más de una en el periodo, siempre se detallan en el orden temporal en que ocurren):

Secuencia	Entrenamiento	Validación	Test	Resultado
1	Alcista muy fuerte	Mayormente bajista	Mayormente bajista	Siempre igual que BH
2	Mixta (alcista – bajista)	Mixta (bajista – alcista)	Alcista	Siempre peor que BH
3	Mayormente bajista	Alcista	Alcista (muy fuerte en IBEX)	Siempre peor que BH (el peor)
4	Mixta (bajista – alcista $\times 2$)	Alcista (muy fuerte en IBEX)	Mixta (bajista muy fuerte – alcista)	Siempre mejor que BH
5	Alcista	Mixta (bajista muy fuerte $\times 2$ – alcista)	Alcista en DJI Plana en IBEX	Mejor o igual que BH en DJI Ligeramente peor que BH en IBEX

Puede comprobarse como en la primera secuencia, las reglas obtenidas siempre indican estar dentro del mercado. Esto es lógico porque en el periodo de entrenamiento hay una tendencia alcista acusada, en la que resulta muy complicado obtener más beneficio entrando y saliendo del mercado que estando dentro de él en todo momento. A pesar de que el periodo de validación sea bajista, no es posible encontrar una regla que sea la mejor de alguna generación en el periodo de entrenamiento que se comporte mejor que *buy and hold* en el de validación, ya que siempre la mejor regla indicará estar dentro del mercado en todo momento.

Los mejores resultados se producen en la secuencia 4 y, en el caso de DJI, también en 5. Una característica que tienen en común es que los periodos de entrenamiento y validación tienen una tendencia alcista uno de ellos y una tendencia mixta el otro, concretamente bajista en primer lugar y con una considerablemente más duradera que la otra. Una razón que explique que esta observación pueda ser un factor importante es que es necesario contar con una tendencia clara y con un periodo en el que se produzca un cambio de tendencia para poder aprender patrones que den beneficio durante la tendencia y al mismo tiempo anticipen un cambio en ella. En contraste, la secuencia 2, que siempre se comporta peor que *buy and hold*, tiene tendencias mixtas y opuestas en entrenamiento y validación, con lo cual no acertaría a descubrir patrones que den beneficios en tendencias duraderas, como ocurre en su periodo de test.

Otra coincidencia en las secuencias 4 y 5 es que los periodos de entrenamiento y de test tienen las mismas tendencias. Una explicación para esta coincidencia puede ser que el periodo de entrenamiento es el utilizado para encontrar los patrones en los precios, mientras que el de validación es para comprobar que esos patrones se mantienen en el tiempo. Por ello, si el

mercado se comporta de forma similar cuando la tendencia es la misma, resulta más efectiva la búsqueda de patrones de precios comunes cuando las tendencias son similares. Este motivo es reforzado por la secuencia 3, que siempre da los peores resultados, en la que los períodos de entrenamiento y test tienen tendencias completamente opuestas.

Es necesario tener en cuenta que las relaciones explicadas entre la tendencia del mercado en los diferentes períodos de las secuencias y los resultados dados por las reglas no son concluyentes. Es fácil observar que tanto el número de secuencias como las tendencias en cada período no son suficientes para aportar resultados sólidos. Por lo tanto, este apartado es una guía de cómo realizar experimentos que relacionen tendencia y resultados de las reglas, y qué razones pueden estar detrás de los resultados.

Por último, la tendencia del mercado podría no ser un factor importante en los resultados producidos por las reglas, ya que, como se ha comentado, la muestra no es lo suficientemente grande como para poder afirmar este hecho.

8.4 Resumen de conclusiones

Mediante la realización de los experimentos de este capítulo, se ha llegado a las siguientes conclusiones:

- Las reglas generadas tienen en cuenta los costes por transacción, el algoritmo las genera de forma consistente, y el comportamiento es parecido entre índices correlacionados.
- Por lo general, el mejor coste por transacción para generar reglas y que éstas den los mayores beneficios es del 3%. No obstante, al no experimentar con mayores costes, puede ser superior.
- Solo se deben aplicar al período de test reglas generadas con un coste por transacción igual o superior al finalmente aplicado en el de test.
- A mayor coste por transacción aplicado en la generación, menores son las transacciones realizadas en el período de test por las reglas.
- Las reglas han de ser generadas con igual tendencia de mercado en el período de entrenamiento y el de test, y con los períodos de entrenamiento y validación uno con tendencia alcista claramente definida y el otro con tendencia mixta (una de las dos tendencias dentro de la mixta ha de durar el doble o más que la otra).
- En los períodos de tiempo utilizados en los experimentos, solo uno de cinco consigue superar a la estrategia *buy and hold*; dos se comportan igual que ella; y dos se comportan considerablemente peor. El período en el que se consigue superar a *buy and hold*, así como en uno de los que se comporta igual, se cumple lo apuntado en el anterior punto.

Como último apunte, es necesario tener presente que los experimentos en este capítulo no pretenden demostrar ningún hecho, únicamente han de servir como indicaciones a considerar

en futuros experimentos y como una guía a seguir para utilizar el algoritmo de generación de reglas de forma adecuada.

En cuanto a futuros experimentos, probar todas las combinaciones posibles de tendencias en las secuencias de experimentación para determinar si hay alguna otra combinación no probada en este proyecto que dé mejores resultados. Un experimento interesante a considerar consiste en generar reglas en periodos de generación y de test cuya tendencia sea la misma, y comprobar si en este caso los resultados superan a los mostrados en estos experimentos, ya que debería ser posible encontrar patrones comunes y efectivos cuando la tendencia es la misma. También se puede considerar probar con períodos no consecutivos en tiempo, ya que es difícil encontrar tendencias lo suficientemente duraderas como para cubrir los tres períodos necesarios para realizar experimentos. Si es cierto que la tendencia determina los resultados de las reglas, entonces comprobar con las mejores combinaciones de tendencias si en la mayor parte de las ocasiones las reglas batén a *buy and hold*. También con estas combinaciones de tendencias, listar el mejor coste por transacción en la generación de reglas por cada coste de test, de modo que se adapte al coste que le es aplicado a cada inversor particularmente. Las muestras debieran de ser de un tamaño adecuado, así como también deberían considerarse otros índices o valores. De todas formas, estos son solo una pequeña muestra de la infinidad de experimentos que pueden ser llevados a cabo relacionados con costes de transacción y tendencia en el mercado.

Referencias utilizadas en el capítulo: [1], [2], [3], [4], [8], [9].

CONCLUSIÓN

9. CONCLUSIONES

9.1 Objetivos conseguidos

Este proyecto ha contado con 10 objetivos, mencionados en el apartado 1.3. En el presente apartado se explicará en qué grado se ha logrado cumplir cada uno de ellos.

Profundizar los conocimientos sobre algoritmos genéticos y en particular de programación genética.

Los conocimientos previos al comienzo del proyecto sobre algoritmos genéticos fueron adquiridos en la asignatura Inteligencia Artificial (IA). No obstante estos conocimientos eran muy básicos e insuficientes para desarrollar el proyecto. De estos algoritmos, la técnica de programación genética no era conocida, y por ello fue necesario aprenderla. Después de realizar el proyecto, se ha adquirido un nivel medio de conocimiento en estos dos conceptos, suficiente para realizar el proyecto.

Aprender las características de los mercados de valores y los diferentes criterios de inversión, en particular el análisis técnico.

Los conocimientos previos al comienzo del proyecto sobre todo lo relacionado con mercados financieros eran bajos – medios. Estos conocimientos provenían de un interés personal en las finanzas y de un seminario sobre finanzas computacionales organizado por el director de este proyecto. Después de haber finalizado el proyecto, se ha aumentado ligeramente el nivel de conocimientos en este aspecto.

Diseñar e implementar, a partir de otros trabajos de investigación sobre el tema, un algoritmo de programación genética que sirva para encontrar reglas de inversión basadas en el análisis técnico.

Se ha conseguido diseñar e implementar el algoritmo señalado funcionando correctamente. Los artículos en los que ha sido basado su diseño no ofrecen detalles sobre la implementación del algoritmo realizada por sus autores, y por ello ha sido necesario hacer la implementación con ideas propias y apoyándose en el trabajo de Keith y Martin [5].

Adicionalmente, se ha incluido una mejora no observada en los artículos en los que ha sido basado el diseño. Esta mejora consiste en reducir el espacio de búsqueda introduciendo un tipo de datos para cada nodo de los árboles y limitando los árboles válidos a aquellos que tienen nodos cuyos descendientes son de un tipo adecuado.

Desarrollar una aplicación de tamaño medio siguiendo todas las etapas de la ingeniería del software: análisis, especificación, diseño e implementación. Esta aplicación ha de permitir utilizar el algoritmo de programación genética desde el punto de vista de un inversor o de un investigador.

El objetivo ha sido cumplido satisfactoriamente. Previamente al inicio del proyecto ya se disponía de experiencia en desarrollar una aplicación de forma similar, adquirida en las asignaturas Proyecto de Ingeniería del Software y Bases de Datos (PESBD), Ingeniería del Software I (ES1) e Ingeniería del Software 2 (ES2). Consecuentemente, al acabar el proyecto, dicha experiencia ha sido reforzada. La aplicación ha permitido generar los datos necesarios para realizar la experimentación del capítulo 8.

Mejorar la habilidad de programación en el lenguaje C++ y aprender a usar funciones contenidas en librerías dinámicas de terceras partes.

La habilidad y los conocimientos relativos a C++ antes de iniciar el proyecto eran bajos, ya que no se había desarrollado ninguna aplicación utilizando las características propias de dicho lenguaje (aunque sí utilizando sintaxis de C). Sin embargo, si que se disponía de experiencia en C, antecesor de C++, y en Java, que también es un lenguaje orientado a objetos. Por esto, la adaptación a C++ fue posible sin dedicarle un tiempo excesivo. Después de realizar el proyecto, el nivel de C++ adquirido puede calificarse como medio – alto.

De manera similar a C++, no se disponía de experiencia previa en la utilización de librerías dinámicas externas. Una vez finalizado el proyecto, se han adquirido los conocimientos necesarios para utilizar estas librerías de la forma correcta en un proyecto de C++.

Utilizar el framework Qt para construir una interfaz de usuario con apariencia profesional.

El objetivo se ha cumplido satisfactoriamente. La experiencia inicial con Qt era baja – media, adquirida en la asignatura Visualización e Interacción Gráficas (VIG), pero suficiente para desarrollar la interfaz gráfica del proyecto.

Realizar un conjunto de experimentos que permitan determinar la relación existente entre los costes por transacción aplicados en la generación de las reglas de inversión con el algoritmo de programación genética y el beneficio obtenido por éstas junto al número de transacciones realizadas en un periodo de test.

Los experimentos han sido realizados, mostrando que las reglas de inversión generadas se comportan de forma lógica y como cabría esperar. Concretamente, a medida que los costes por transacción aumentan en los períodos utilizados para generar las reglas, éstas realizan

menos transacciones en el periodo de test. Además, las reglas se adaptan a los costes por transacción con las que fueron generadas y obtienen los mejores beneficios si las reglas son aplicadas en el periodo de test con esos mismos costes o inferiores. En general, los costes por transacción que han de ser aplicados para generar reglas y que éstas se comporten de la mejor forma posible en el periodo de test son altos, 3% o superiores.

Realizar un conjunto de experimentos que permitan relacionar el beneficio dado por las reglas de inversión obtenidas con el algoritmo de programación genética con la tendencia del mercado en los periodos utilizados para generarlas y testearlas, teniendo en cuenta los costes por transacción.

Los experimentos han sido realizados, mostrando que las reglas de inversión generadas dan el mejor resultado en el periodo de test cuando se cumplen las siguientes dos condiciones: (1) el periodo de entrenamiento y de test tienen la misma tendencia, y (2) uno de los periodos de entrenamiento o validación tiene una tendencia alcista clara y el otro una tendencia mixta. Sin embargo, las muestras han sido insuficientes para poder sacar conclusiones claras.

Realizar un conjunto de experimentos que permitan confirmar si las reglas de inversión obtenidas con el algoritmo de programación genética generan un beneficio superior a la estrategia *buy and hold*, teniendo en cuenta los costes por transacción.

Los experimentos han sido realizados utilizando los datos generados en los dos experimentos anteriores, mostrando que las reglas de inversión generadas pueden batir a *buy and hold* únicamente bajo ciertas condiciones, especificadas en el experimento explicado en el objetivo anterior. No obstante las muestras no han sido suficientes como para poder afirmarlo rotundamente.

Elaborar una documentación que recoja todo lo realizado y que permita en un futuro entender y ampliar el proyecto por diferentes personas.

Como se puede comprobar, la documentación ha sido realizada. Con la excepción del código fuente, se ha incluido todo lo necesario para poder comprender el algoritmo, la estructura de la aplicación diseñada mediante ingeniería del software y todos los datos y conclusiones sobre los experimentos con las reglas de inversión generadas por el algoritmo. Además, se la ha dotado de un estudio económico del proyecto así como de una estructura ordenada.

9.2 Ampliaciones futuras

El proyecto tiene bastantes puntos por los cuales puede ser ampliado en un futuro, algunos de los cuales ya han sido mencionados a lo largo del proyecto. En este apartado se resumirán los más importantes.

Una ampliación posible es dotar al algoritmo de programación genética con más tipos de nodos, correspondientes a indicadores de análisis técnico. Esto permitiría tener en cuenta más factores que pueden afectar a la detección de patrones en los precios, con lo cual podría incrementar la efectividad de las reglas generadas.

En la parte de los experimentos hay diversas ampliaciones posibles. Éstas han sido comentadas en el apartado 8.4, y se resumen en ampliar el número de muestras en cada experimento, aumentar el número de casos de estudio en la parte de la influencia de la tendencia y en considerar pequeñas variaciones en los experimentos llevados a cabo. Aparte de estas ampliaciones, se deberían realizar test estadísticos para poder demostrar científicamente las conclusiones.

Por último, al ser un proyecto en el que la programación se ha realizado con el lenguaje C++, en el cual es necesario liberar toda la memoria cogida de forma dinámica cuando ya no se utilice, se debería de controlar el uso de la memoria para que no haya memoria sin liberar o *memory leaks*. En caso de haber algún punto en el que no se libere memoria, el programa puede detenerse de forma inesperada al cabo de un cierto tiempo de ejecución por haber agotado toda la memoria disponible en el sistema. En un proyecto de tamaño mediano o grande resulta complicado controlar que toda la memoria sea liberada al final de su uso debido a la gran cantidad de código. Por ello existen programas especiales para tal fin, por ejemplo *Valgrind*, que ayudan a detectar *memory leaks* en tiempo de ejecución. En el programa desarrollado en este proyecto no ha sido controlada esta situación de forma exhaustiva por falta de tiempo, aunque ha sido probado durante 2 horas sin causar problemas en un sistema con 1GB de memoria. No obstante, en el futuro debería considerarse eliminar *memory leaks* que pudieran existir.

9.3 Valoración personal

Este proyecto ha supuesto una dedicación de tiempo y esfuerzo muy importantes, diferente por estos motivos al resto de proyectos desarrollados durante la carrera. El trabajar de principio a fin y de forma individual, ha permitido poner en práctica y clarificar conceptos que durante la realización de otros proyectos en la carrera se habían pasado por alto.

Adicionalmente, la necesidad de aprender algunas partes de forma autónoma y de adaptar ese aprendizaje al proyecto, además de realizar tareas de investigación, ha supuesto reflexionar considerablemente sobre todos los conceptos involucrados. Esto ha supuesto, aparte de la importante adquisición de conocimientos, probar alternativas que finalmente no fueron viables. Por ejemplo, en el diseño del algoritmo, se intentó utilizar una técnica de programación genética prometedora, llamada Gene Expression Programming (GEP), la cual tiende a encontrar buenas soluciones de forma más rápida y reduce el volumen de cálculo, entre otras carac-

terísticas. Sin embargo, esta técnica fue descartada a causa de que no se adapta correctamente a tener nodos con tipos diferentes y a la necesidad de controlar los tipos de nodos descendientes al aplicar los operadores. Otro ejemplo es el intento de introducción del operador de cruce por un punto en el algoritmo genético (escoger una posición del vector de nodos y realizar el cruce de todos los nodos a partir de esa posición), que no fue posible por el mismo motivo. En este sentido, el director del proyecto ha sido de gran ayuda para clarificar dudas y sacar conclusiones.

Durante el desarrollo de la parte final del proyecto han surgido imprevistos que han imposibilitado trabajar a jornada completa en él. Por ello, la planificación inicial se ha visto alterada en cuanto a la fecha prevista de finalización, que ha sido pospuesta algo más de 5 meses. Sin embargo, la cantidad de horas dedicadas no ha sufrido cambios significativos.

10. REFERENCIAS

10.1 Bibliografía

- [1] Allen, Franklin y Karjalainen, Risto (1993). *Using Genetic Algorithms to Find Technical Trading Rules*. Rodney L. White Center for Financial Research, The Wharton School, University of Pennsylvania, PA, EEUU.
- [2] Yu, Tina; Shu-Heng, Chen y Kuo, Tzu-Wen (2005). Discovering Financial Technical Trading Rules Using Genetic Programming with Lambda Abstraction. *Genetic Programming Theory and Practice II*, Chapter 2, 11 – 30.
- [3] Korczak, J. y Roger, P. (2002). *Stock Timing Using Genetic Algorithms*. John Wiley & Sons, Ltd.
- [4] Potvin, Jean-Yves; Soriano, Patrick y Vallée, Maxime (2004). *Generating Trading Rules on the Stock Markets with Genetic Programming*. Computers & Operations Research, number 31, 1033 – 1047.
- [5] Keith, Mike J. y Martin, Martin C. (1994). Genetic programming in C++: implementation issues. *Advances in genetic programming*, capítulo 13, 285 – 310. MIT Press Cambridge, MA, EEUU.
- [6] Koza, John R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, EEUU.
- [7] Arratia, Argimiro A. (2010). *Computational Finance. A Short Introduction*. LSI, Universitat Politècnica de Catalunya, Barcelona, España. Transparencias del curso.
- [8] Maté, Carlos y Arroyo, Javier (2006). Descriptive Statistics for Boxplot Variables. *COMPSTAT 2006. Proceedings in Computational Statistics*. Springer.
- [9] Gonzalez, Liliana; Powell, John G.; Shi, Jing y Wilson, Antony (2005). Two centuries of bull and bear market cycles. *International Review of Economics & Finance*, vol. 14, issue 4, 469 – 486.
- [10] McGuigan, Thomas P. (2006). *The Difficulty of Selecting Superior Mutual Fund Performance*. CFP.
- [11] Costal, Dolors; Sancho, M. Ribera y Teniente, Ernest (2000). *Enginyeria del software: Especificació*. Universitat Politècnica de Catalunya.
- [12] Gómez, Cristina; Mayol, Enric; Olivé, Antoni y Teniente, Ernest (2001). *Enginyeria del software: Disseny 1*. Universitat Politècnica de Catalunya.
- [13] (2010). *Intel·ligència Artificial (IA). Apunts*. LSI, Universitat Politècnica de Catalunya. Apuntes del curso.

10.2 Sitios web

- [14] <http://sabia.tic.udc.es/mgestal/cv/AAGTutorial/node12.html>, accedida en noviembre de 2011
- [15] <http://www.actibva.com/magazine/mercados-financieros/la-gestion-de-carteras-gestion-activa-frente-a-gestion-pasiva>, accedida en noviembre de 2011
- [16] <http://www.investopedia.com>, accedida en noviembre de 2011
- [17] <http://www.encyclopediafinanciera.com/inversion/rentavariante/indicesbursatiles.htm>, accedida en noviembre de 2011
- [18] <http://www.ta-guru.com>, accedida en noviembre de 2011
- [19] <http://blog.ulf-wendel.de/?p=216>, accedida en diciembre de 2011
- [20] <http://www.fileinfo.com/extension/so>, accedida en diciembre de 2011
- [21] <http://dev.mysql.com/doc/refman/5.0/en/c.html>, accedida en diciembre de 2011
- [22] <http://netbeans.org/kb/trails/cnd.html>, accedida en diciembre de 2011
- [23] <http://www.dreamincode.net/forums/topic/27087-advantagesdisadvantages-of-programming-languages>, accedida en diciembre de 2011
- [24] <http://dev.mysql.com/doc/refman/5.1/en/load-data.html>, accedida en diciembre de 2011
- [25] <http://dev.mysql.com/doc/refman/5.1/en/insert-speed.html>, accedida en diciembre de 2011
- [26] <http://www.cplusplus.com>, accedida en diciembre de 2011
- [27] <http://www.wikipedia.org>, accedida en diciembre de 2011
- [28] <http://www.elergonomista.com/if07.html>, accedida en noviembre de 2011
- [29] <http://es.finance.yahoo.com>, accedida en enero de 2012

APÉNDICE A: INSTALACIÓN DE LIBRERÍAS EXTERNAS

En esta sección se detalla cómo instalar las librerías externas necesarias para ejecutar e implementar la aplicación construida en este proyecto. Es necesario tener instaladas las librerías de la base de datos (ver capítulo 7.3.3) antes de proceder a instalar el conector de MySQL – C++.

LIBRERIAS BOOST

1. Bajar la última versión de las librerías desde <http://www.boost.org>.
2. Descomprimir el paquete descargado.
3. Abrir un terminal y situarse en la carpeta descomprimida.
4. Ejecutar los siguientes comandos:

```
$ ./bootstrap.sh  
$ sudo ./bjam install
```

CONECTOR MYSQL – C++

1. Bajar el código fuente del conector de <http://www.mysql.com/downloads/connector/cpp/>. Hay que seleccionar la opción *Source Code* del menú desplegable. Las versiones precompiladas parecen no funcionar.
2. Descomprimir el paquete descargado.
3. Abrir un terminal y situarse en la carpeta descomprimida.
4. Instalar el programa *cmake* mediante el siguiente comando (si no está ya instalado):

```
$ sudo apt-get install cmake
```
5. Ejecutar los siguientes comandos:

```
$ cmake .  
$ make  
$ sudo make install
```

EJECUCIÓN DEL PROGRAMA

1. Abrir un terminal, situarse en la carpeta donde está el ejecutable del programa y ejecutar los siguientes comandos (el primero de ellos solamente en la primera ejecución, ya que cambia las propiedades del archivo para que pueda ser ejecutado):

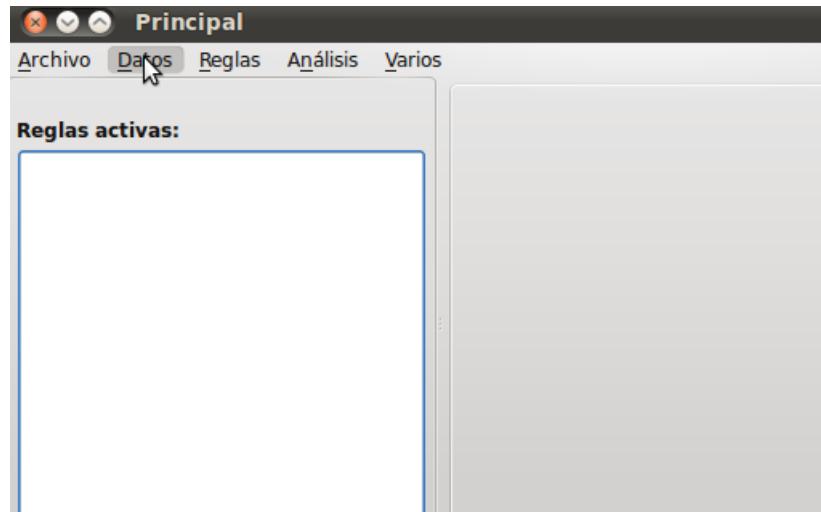
```
$ chmod +x Investigacion_PG  
$ ./Investigacion_PG
```

APÉNDICE B: EJEMPLO DE USO DEL PROGRAMA

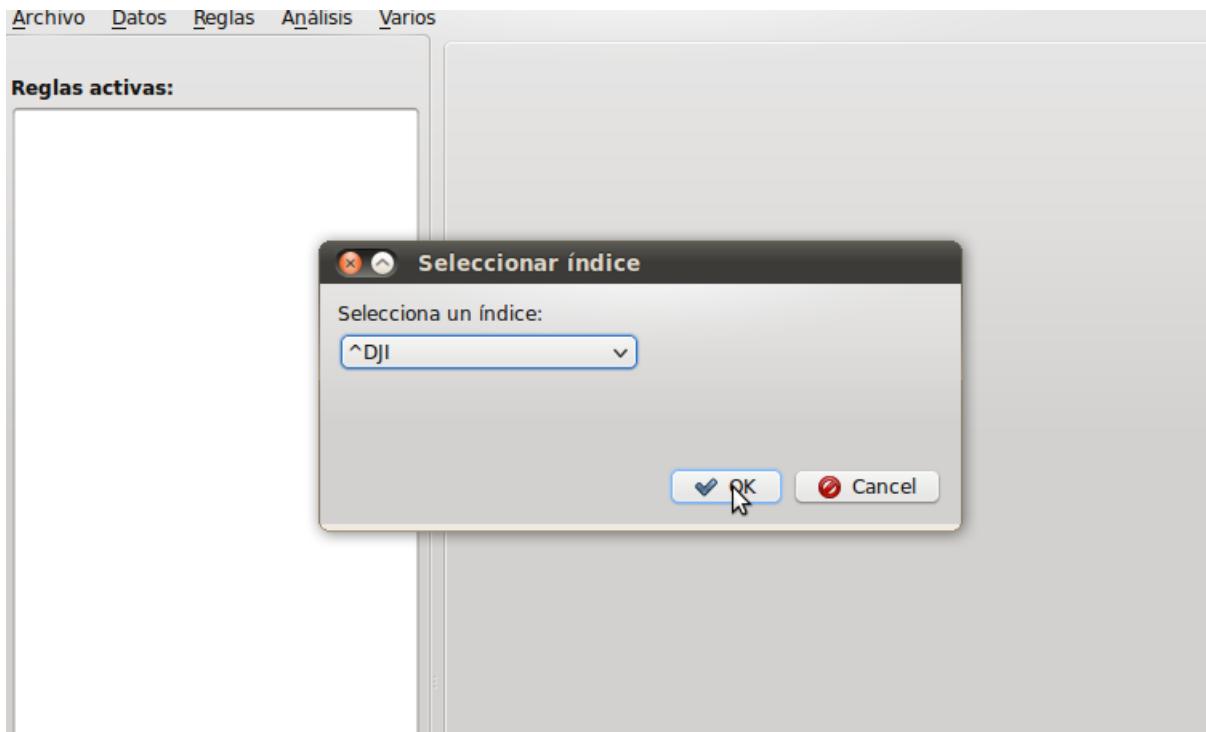
En esta sección se muestran a modo de ejemplo los pasos a seguir para realizar algunos de los casos de uso más importantes especificados en el capítulo 5.3 en el programa implementado en este proyecto. Estos casos de uso se muestran en el orden lógico en el que serían realizados por un usuario. Es importante leer estos pasos, ya que proporcionan indicaciones fundamentales para usar el programa correctamente.

ACTUALIZAR DATOS

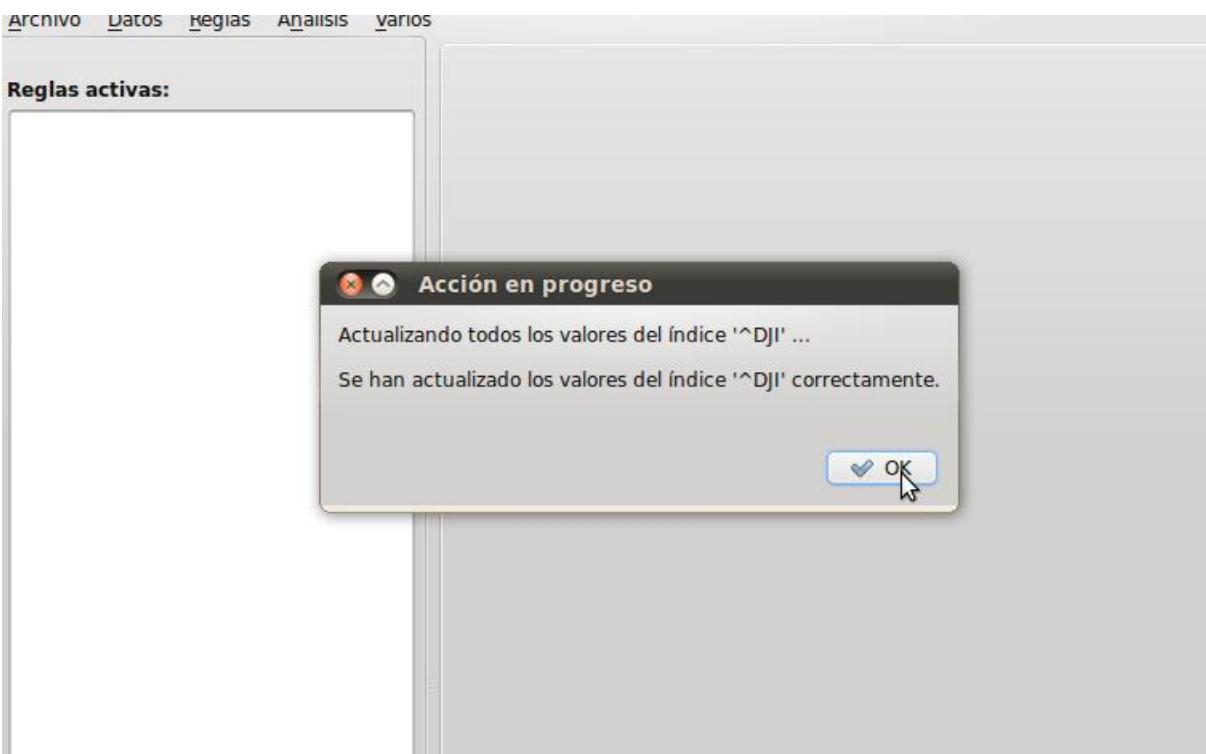
1. En la opción “Datos” del menú principal, seleccionar “Actualizar automáticamente”.



2. A continuación, escoger uno de los índices presentes en la base de datos para actualizar los históricos de precios de todos los valores que lo componen. El índice y los valores que forman parte del índice se han de haber creado previamente. Solo se actualizarán los valores que ya existan en la base de datos; si algún valor forma parte del índice pero no ha sido creado previamente, entonces no será creado ni actualizado.

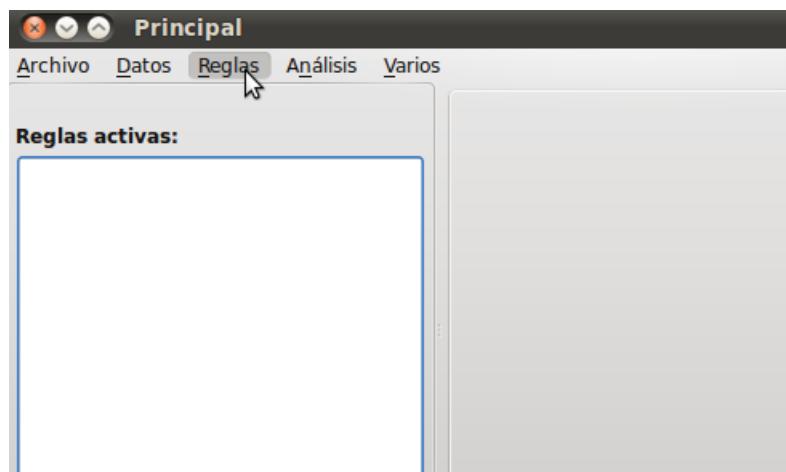


3. En el caso de que la actualización se haya realizado correctamente, se mostrará el diálogo de la siguiente imagen. En caso de error, se mostrará un diálogo con la descripción del error.

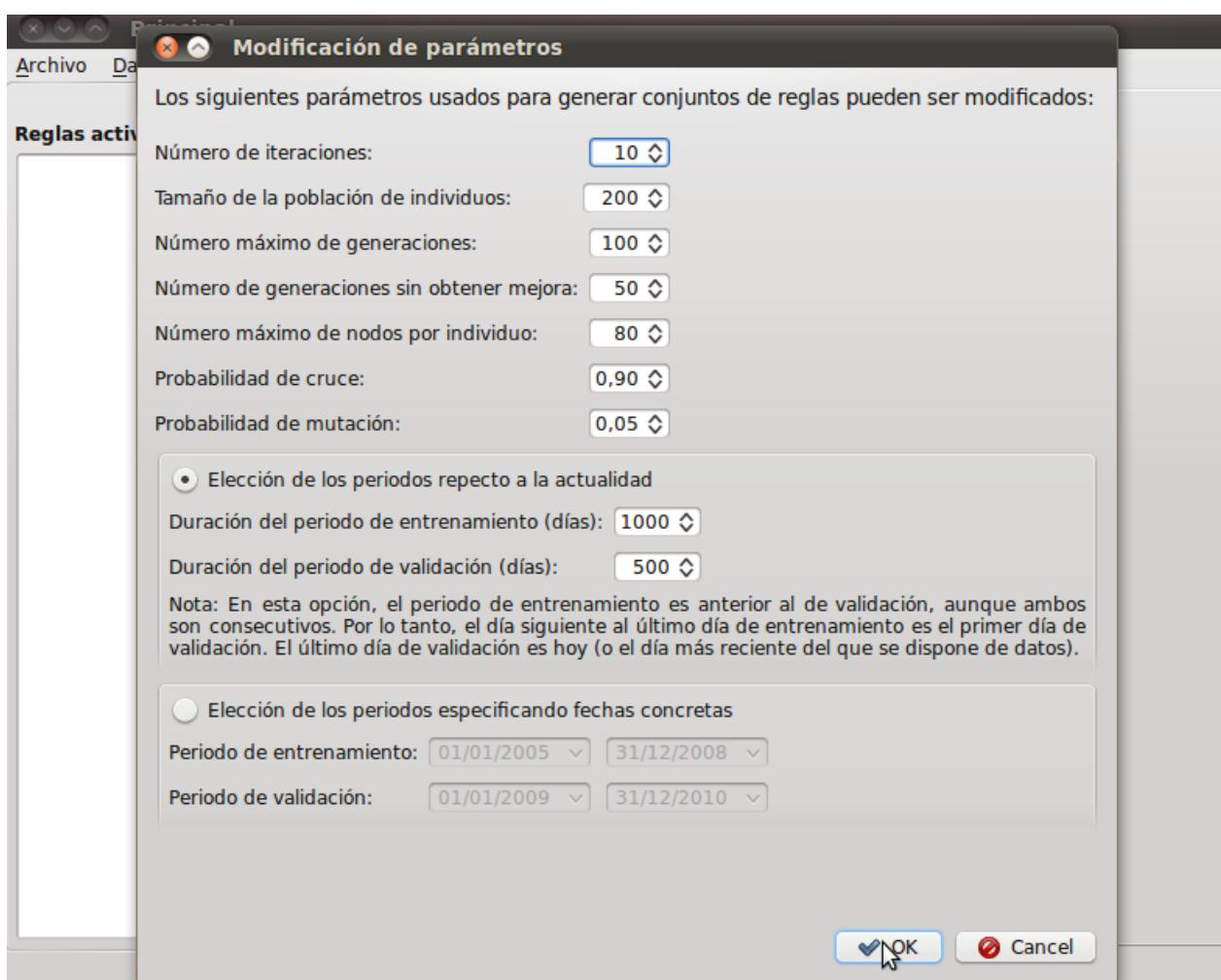


CAMBIAR PARÁMETROS

1. En la opción “Reglas” del menú principal, seleccionar “Ajustar parámetros”.

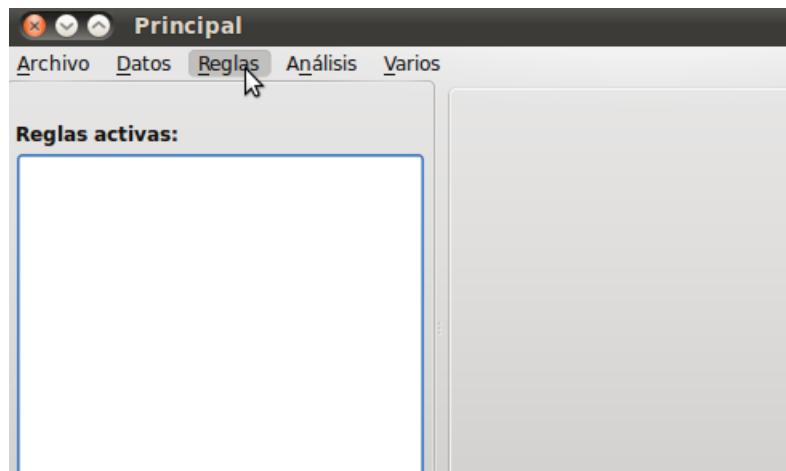


2. En el diálogo mostrado, cambiar los parámetros deseados.

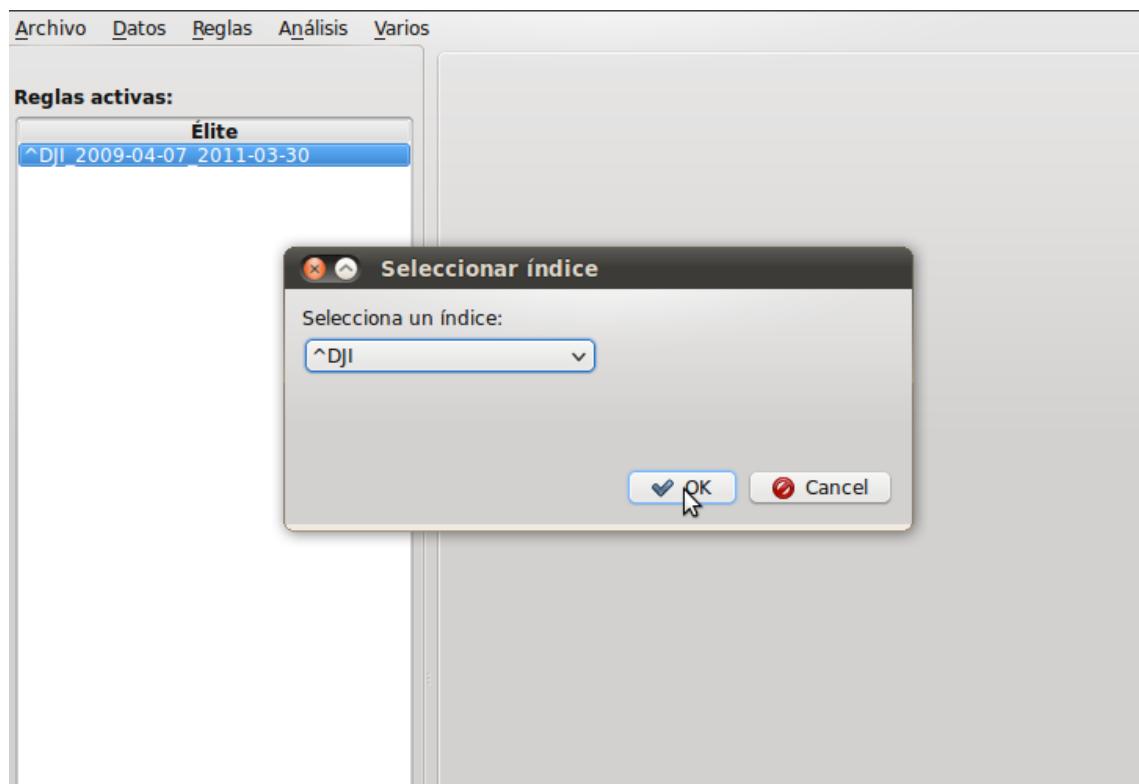


OBTENER CONJUNTO REGLAS

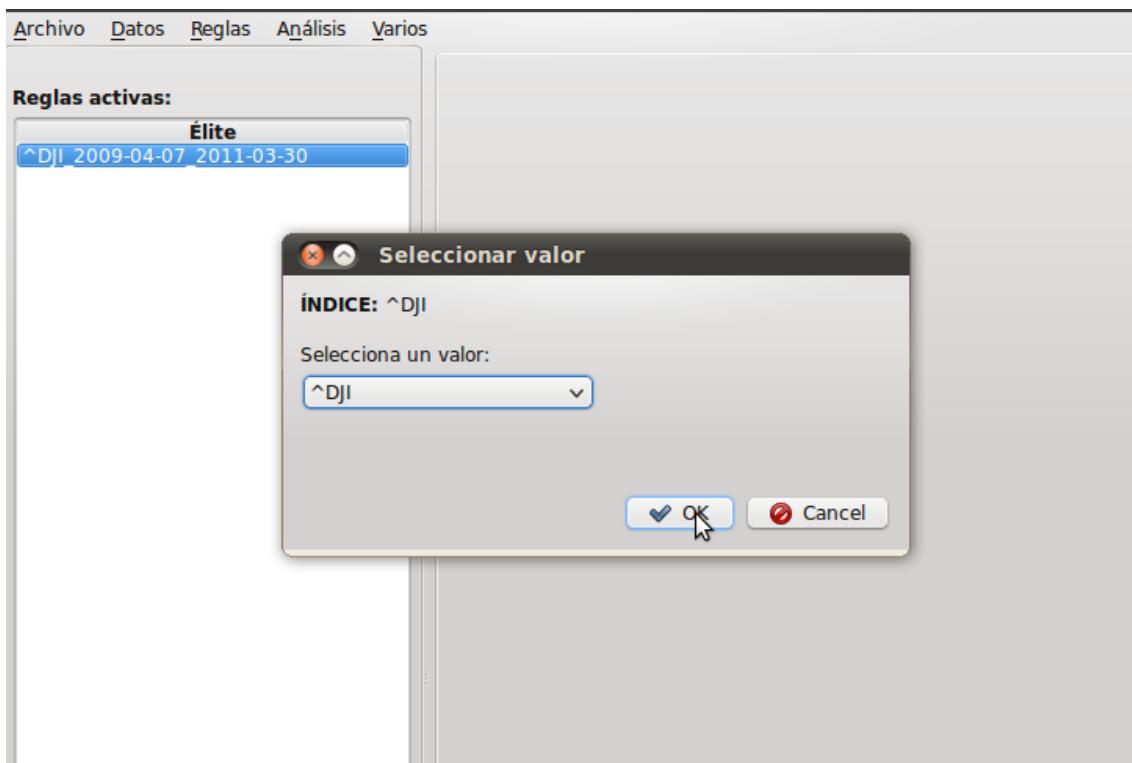
1. En la opción “Reglas” del menú principal, seleccionar “Generar”.



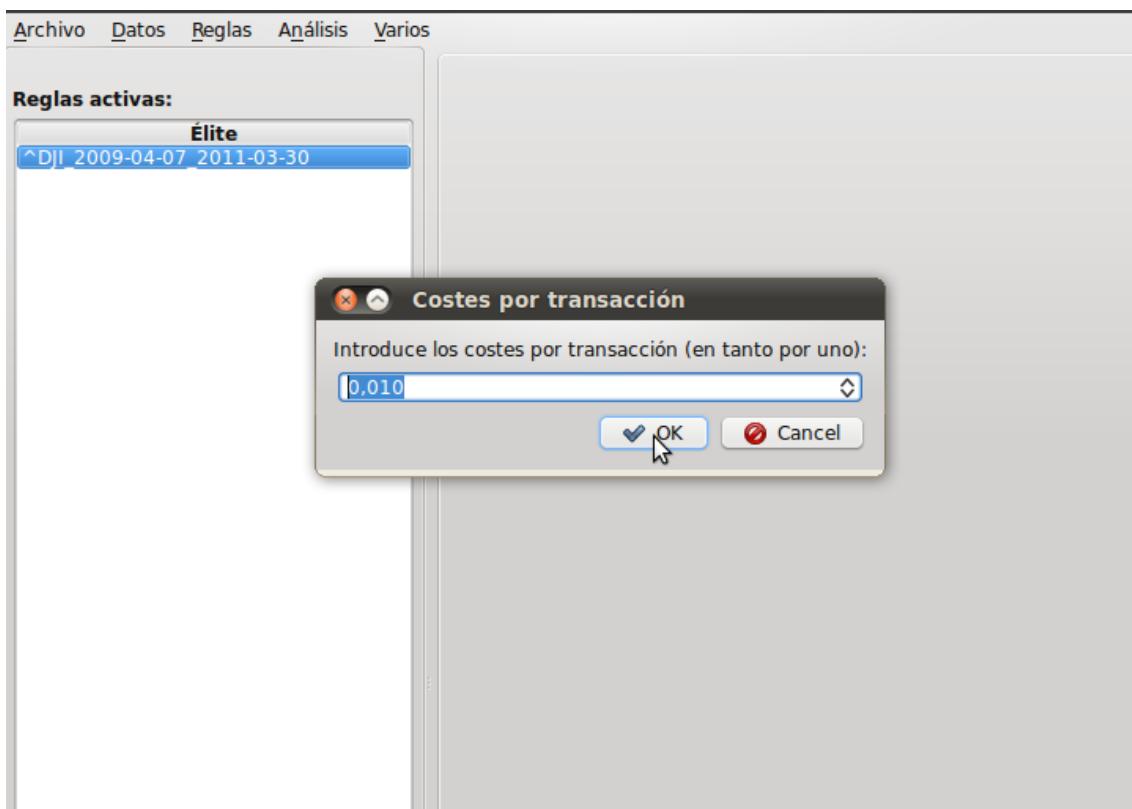
2. Seleccionar uno de los índices existentes en la base de datos. El valor utilizado para generar las reglas ha de pertenecer al índice seleccionado.



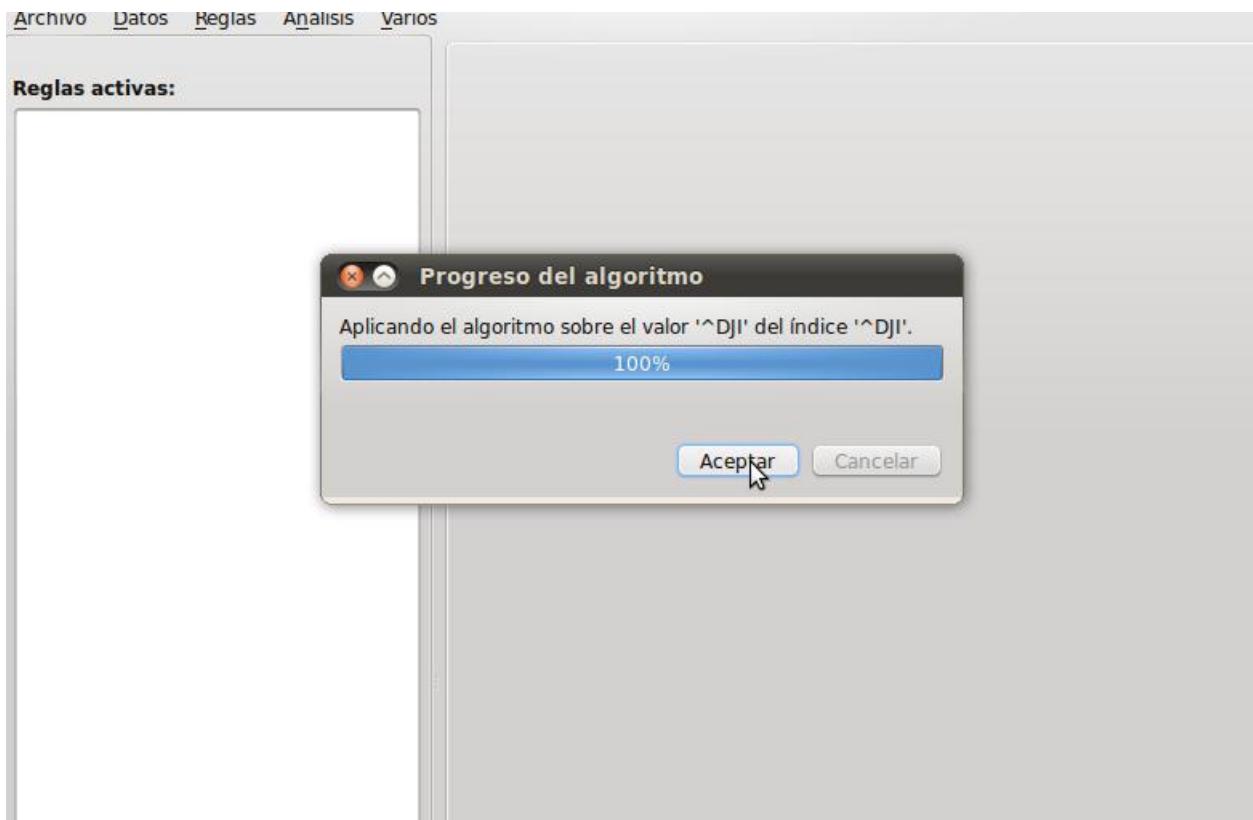
3. Escoger el valor cuyo histórico de precios será utilizado para generar el conjunto de reglas. Si se quiere utilizar el índice en lugar de un valor, éste también debe ser seleccionado en este paso (aparecerá en la lista junto a los valores).



4. Introducir los costes por transacción que serán utilizados en la generación del conjunto de reglas.

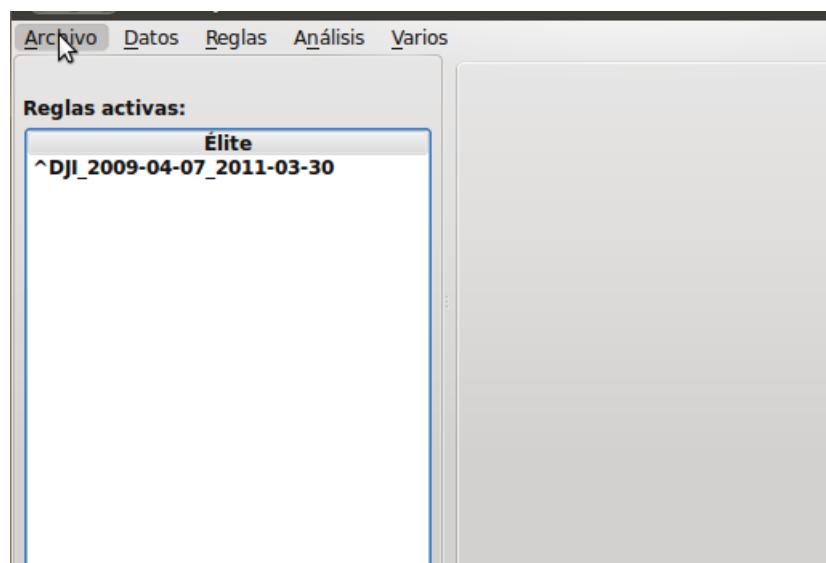


5. Esperar hasta que se complete la generación del conjunto. Durante la generación, se muestra una barra de progreso que aumentará con cada iteración completada (para ver más detalles del progreso, observar el terminal desde el que se ejecuta el programa). Una vez completada la generación del conjunto, aparecerá en la parte “Reglas activas” el conjunto con un nombre por defecto, en el que se incluye el valor utilizado y la fecha inicial y final del periodo de entrenamiento (ver el paso 1 del siguiente caso de uso).

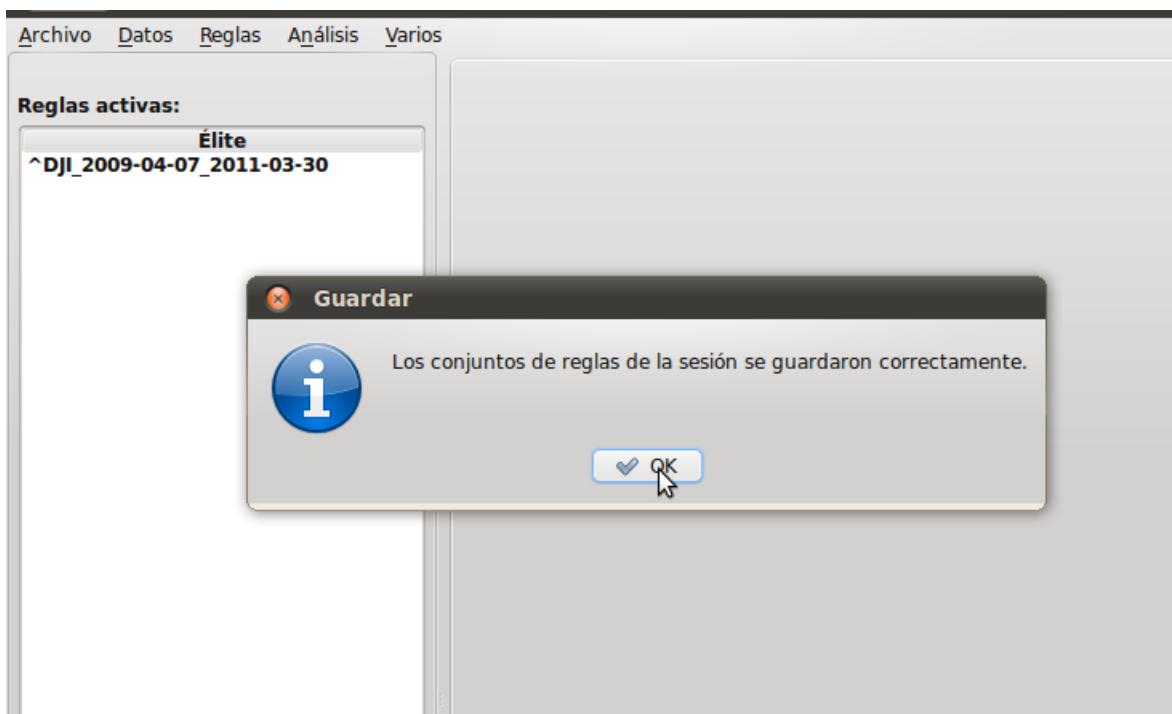


GUARDAR CONJUNTOS REGLAS

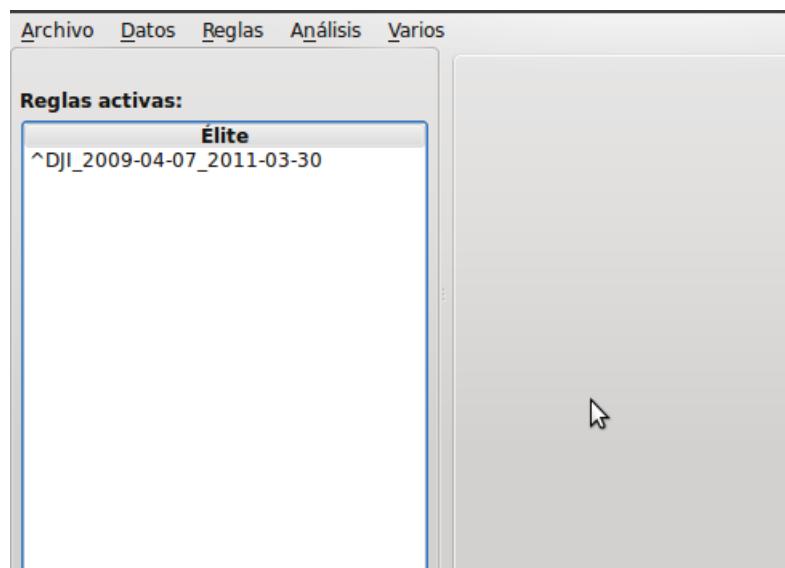
1. En la opción “Archivo” del menú principal, seleccionar “Guardar todo”. Los conjuntos de reglas que no están guardados en la base de datos aparecen en negrita.



2. Si todos los conjuntos de reglas no guardados se han guardado correctamente, aparecerá un diálogo indicándolo. Si ha habido algún error, se detallará en un diálogo.



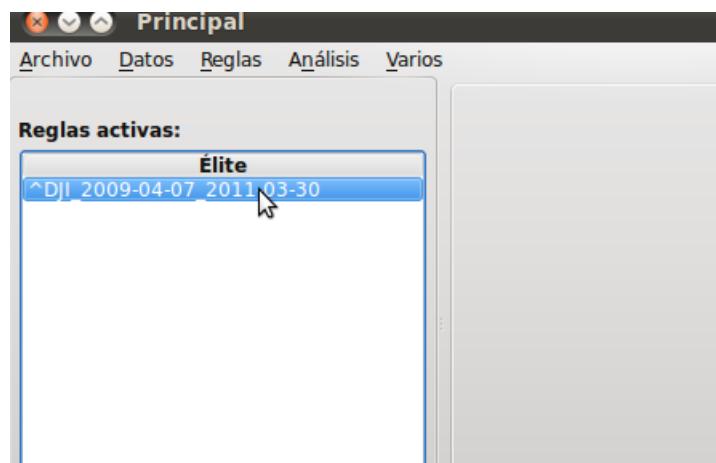
3. Todos los conjuntos guardados correctamente dejarán de mostrarse en negrita.



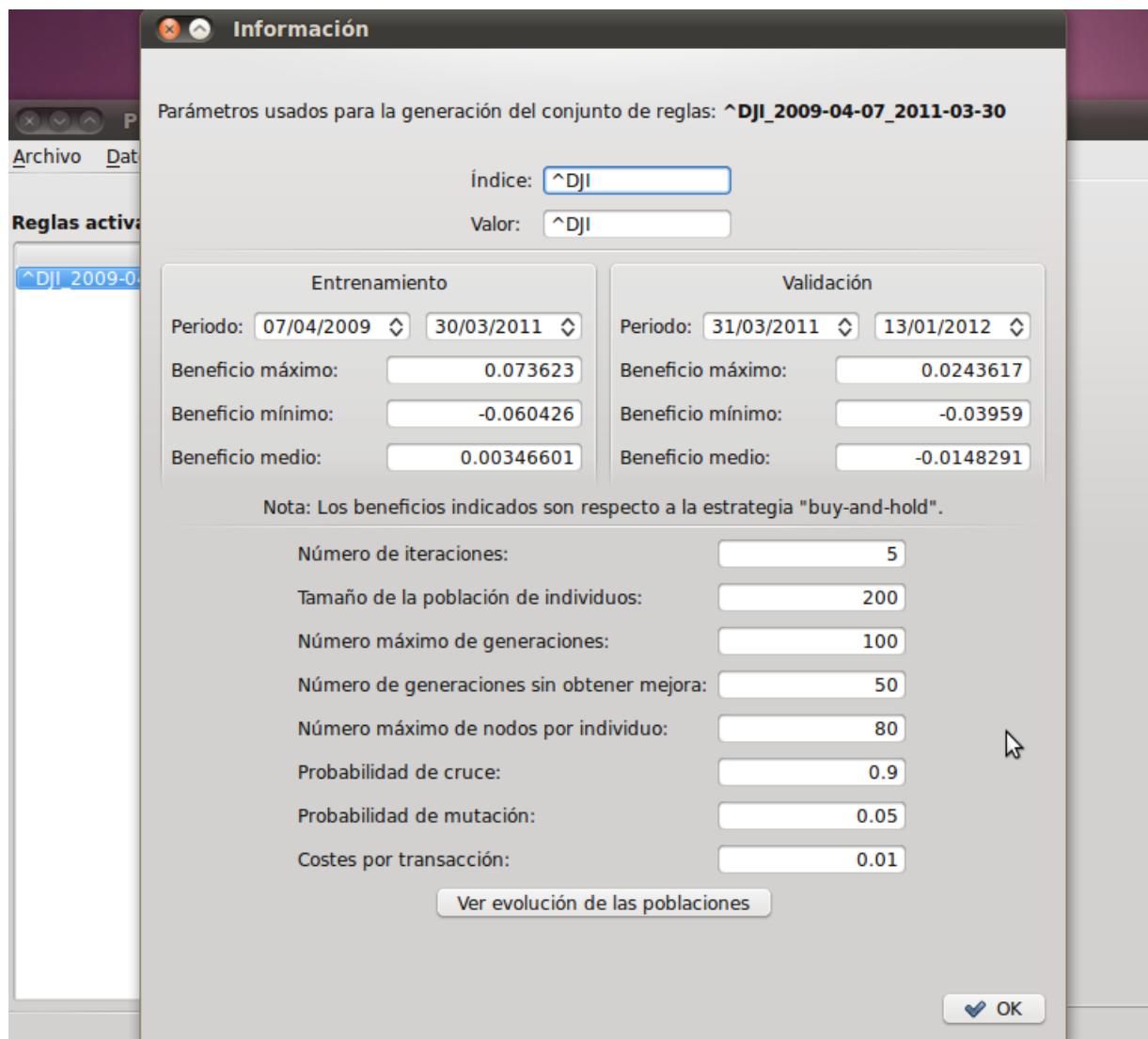
OBTENER INFORMACIÓN CONJUNTO DE REGLAS

Antes de proceder con este caso de uso es necesario haber guardado el conjunto de reglas del que se quiere obtener información. En caso contrario, no se mostrará correctamente la información del conjunto.

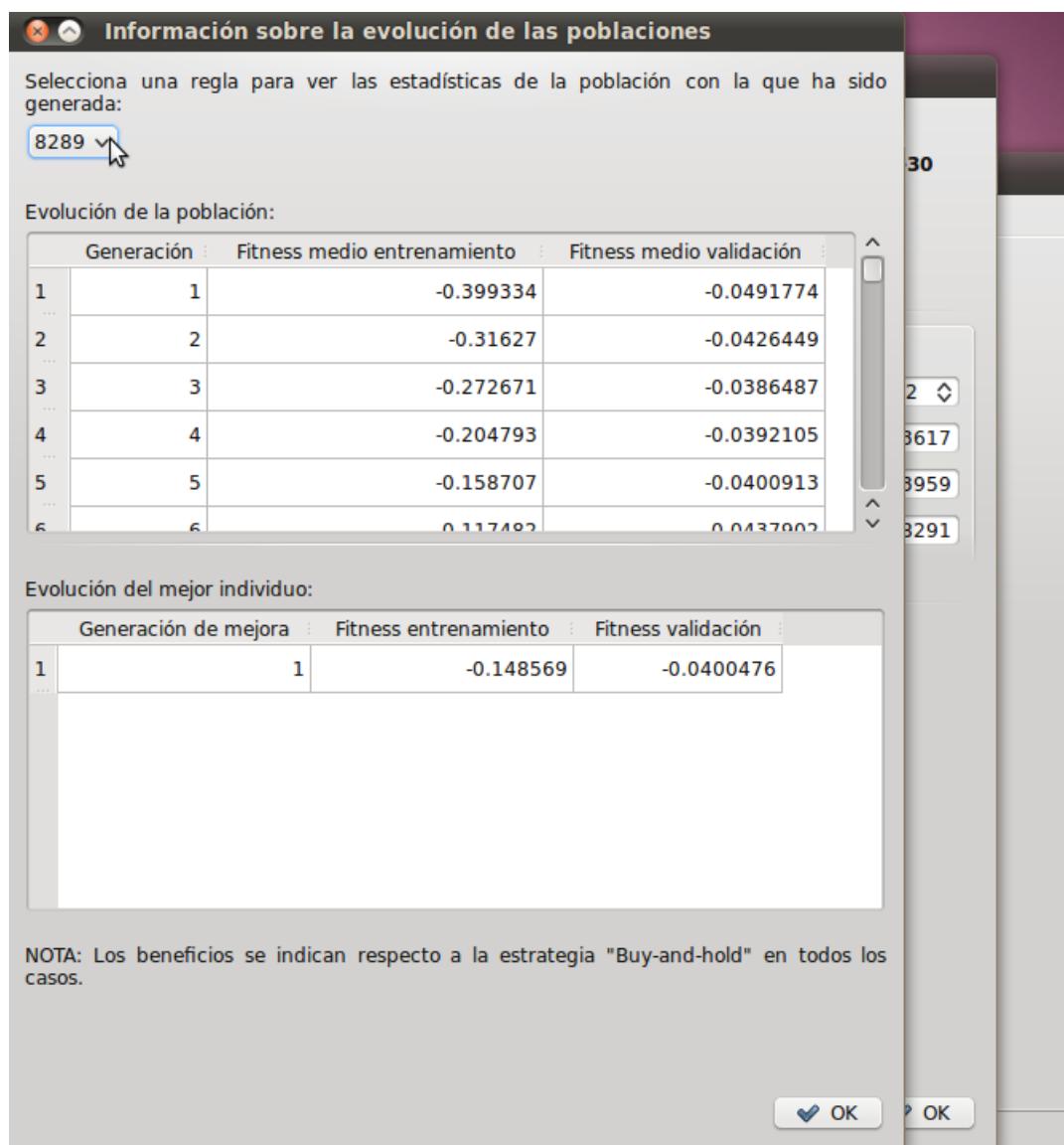
1. Situarse sobre el conjunto del que se quiere obtener información en el panel “Reglas activas” y hacer clic derecho sobre él. A continuación seleccionar la opción “Información” del menú que aparece.



2. En el diálogo mostrado, se pueden observar todos los parámetros usados durante la etapa de generación del conjunto de reglas.



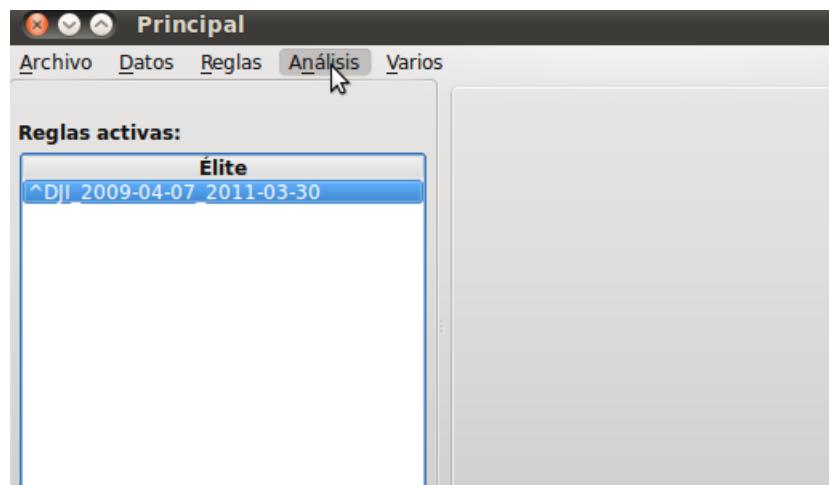
3. También se puede observar la evolución del *fitness* de la población de reglas con la que fue generada cada una de las reglas que componen el conjunto. Para ello hay que seleccionar “Ver evolución de las poblaciones” en el diálogo del paso anterior.



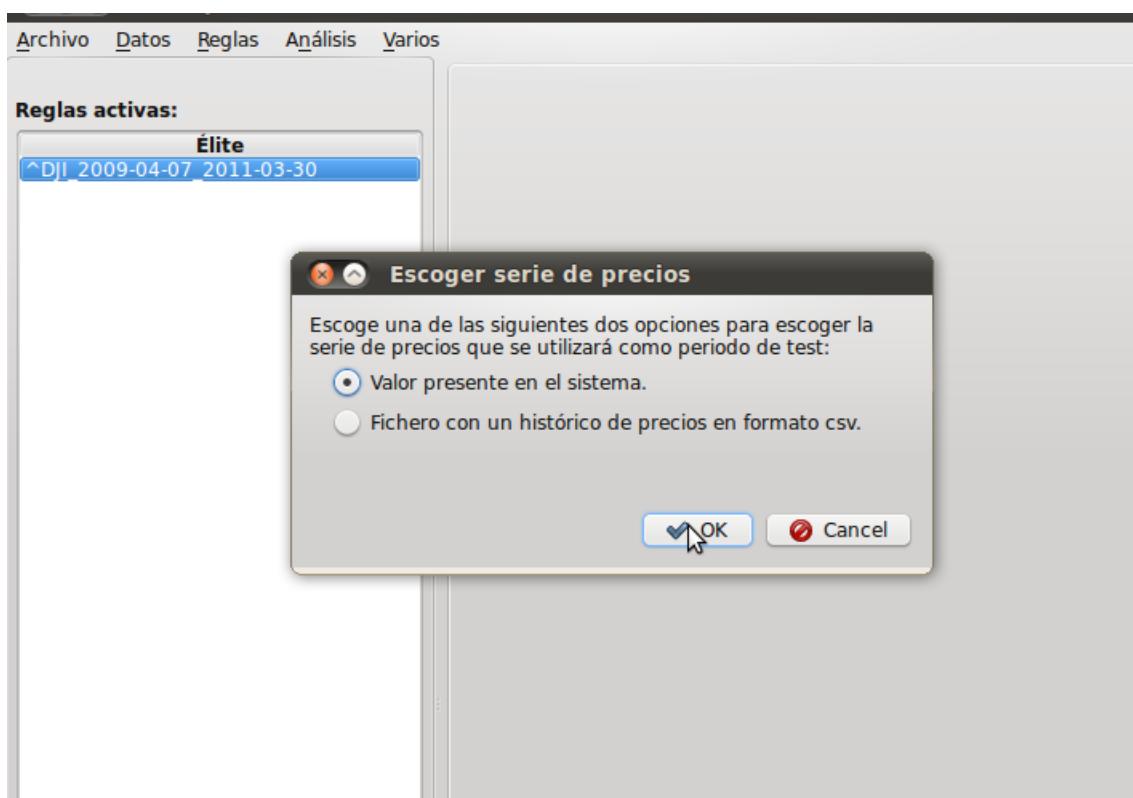
ANALIZAR RESULTADOS

Antes de proceder con este caso de uso, es necesario tener en el panel “Reglas activas” el conjunto de reglas con el cual se quiere efectuar el test.

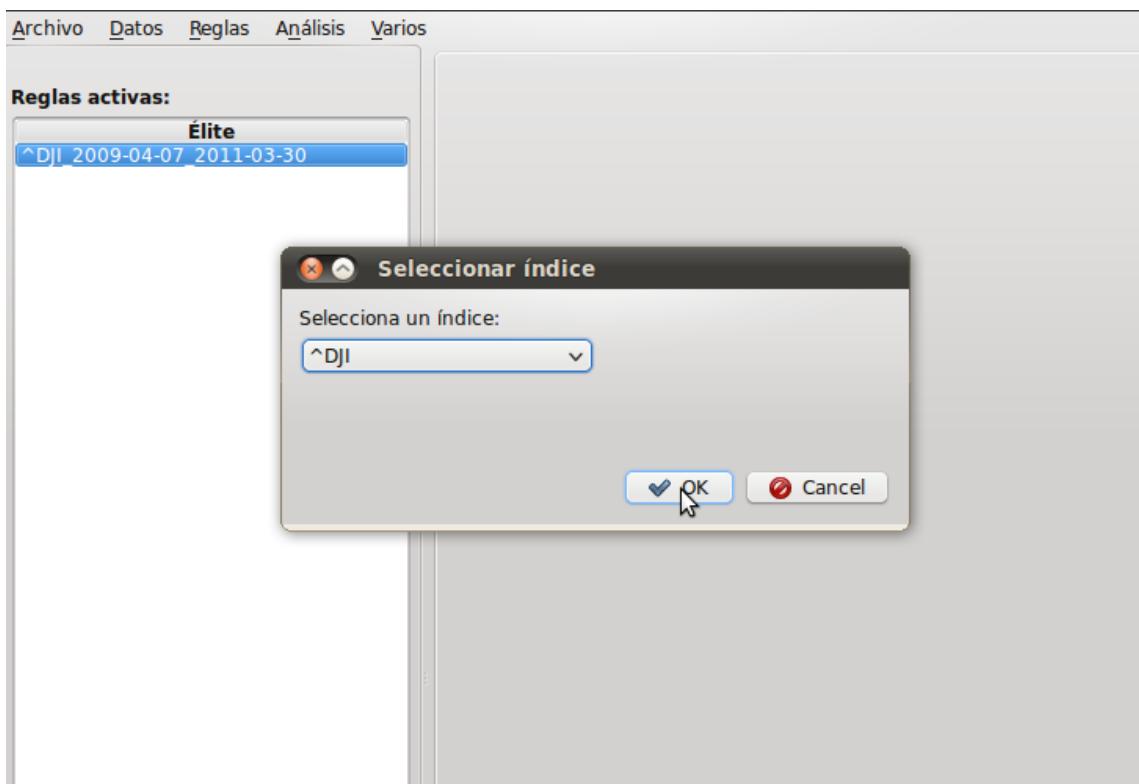
1. En la opción “Análisis” del menú principal, seleccionar “Beneficios”.



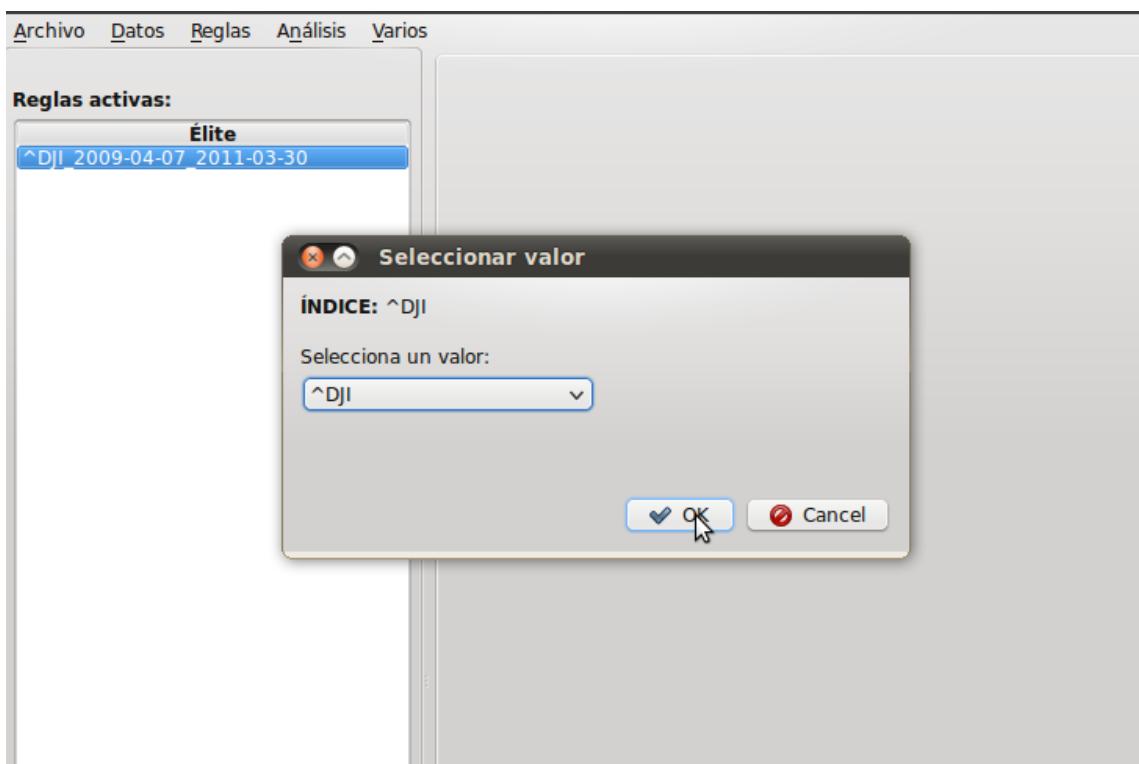
2. Si el valor con el que se quiere hacer el test se encuentra en la base de datos, seleccionar “Valor presente en el sistema.”. Si se dispone de un fichero con los datos del valor, seleccionar “Fichero con un histórico de precios en formato csv.”. En esta explicación se asumirá que se escoge la primera opción.



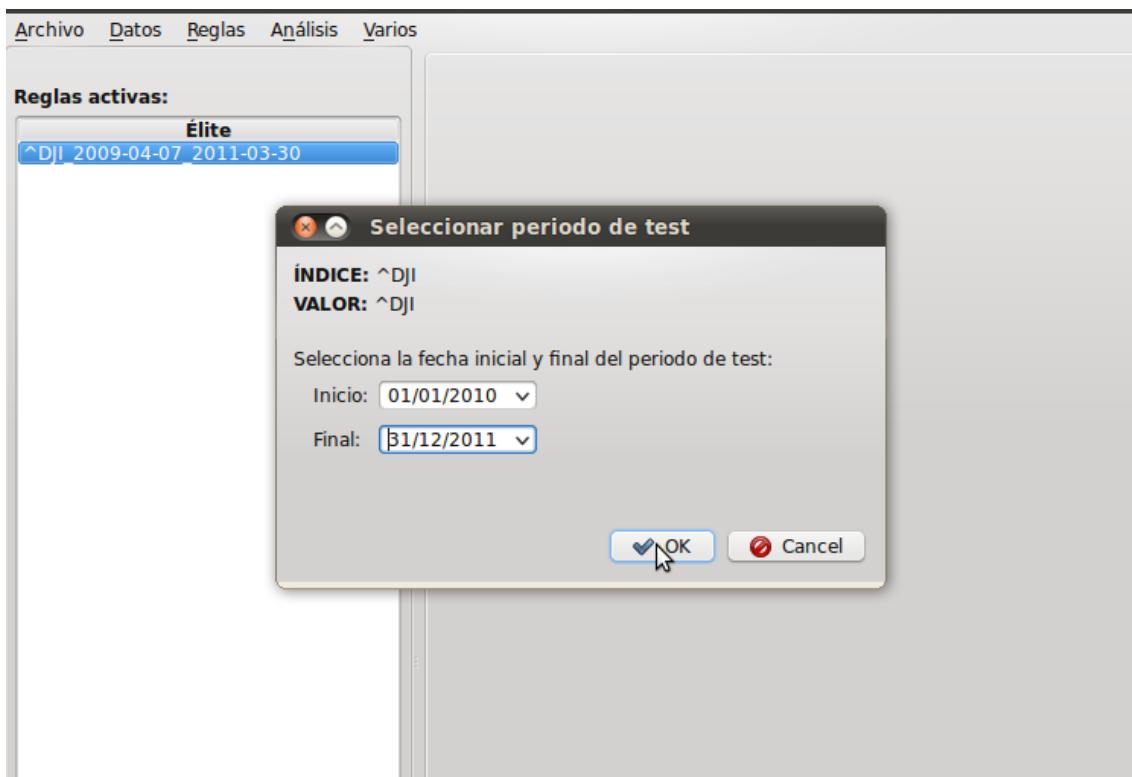
3. Escoger el índice en el que se encuentra el valor cuyo histórico de precios de utilizará en el test.



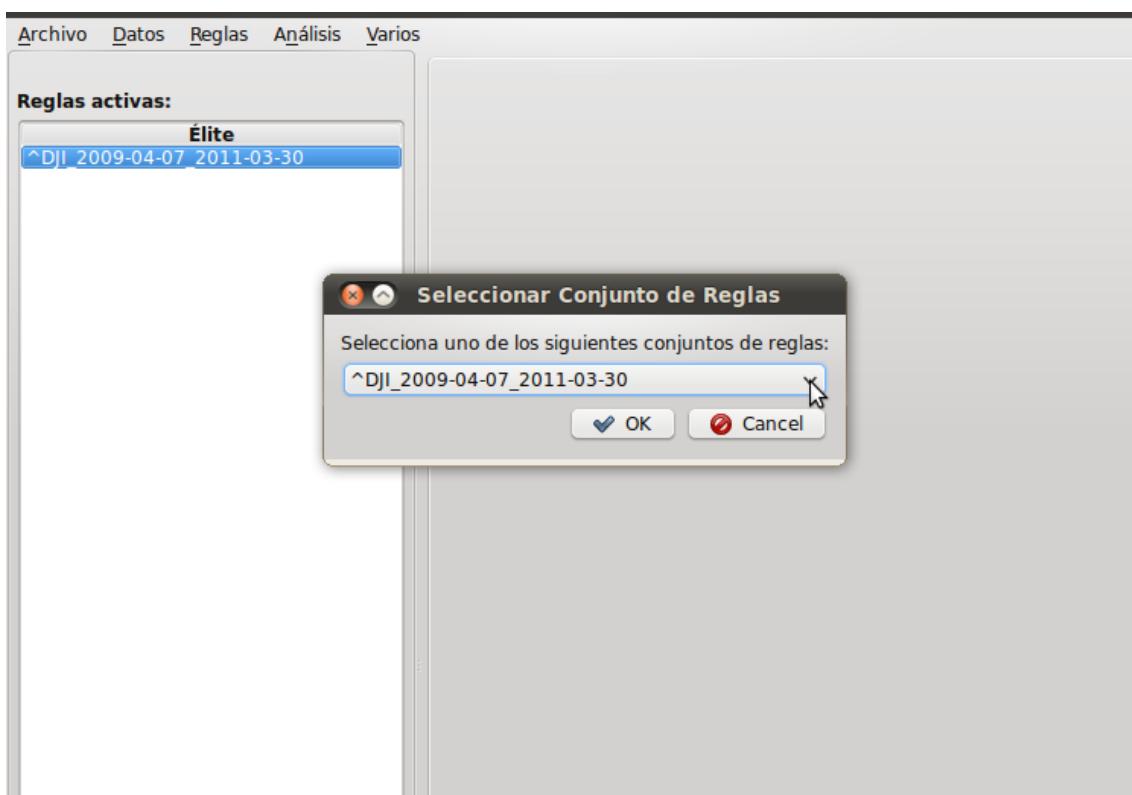
4. Seleccionar el valor que se utilizará en el test. Si en lugar de un valor se desea un índice, también se ha de escoger en este paso (aparecerá en el menú).



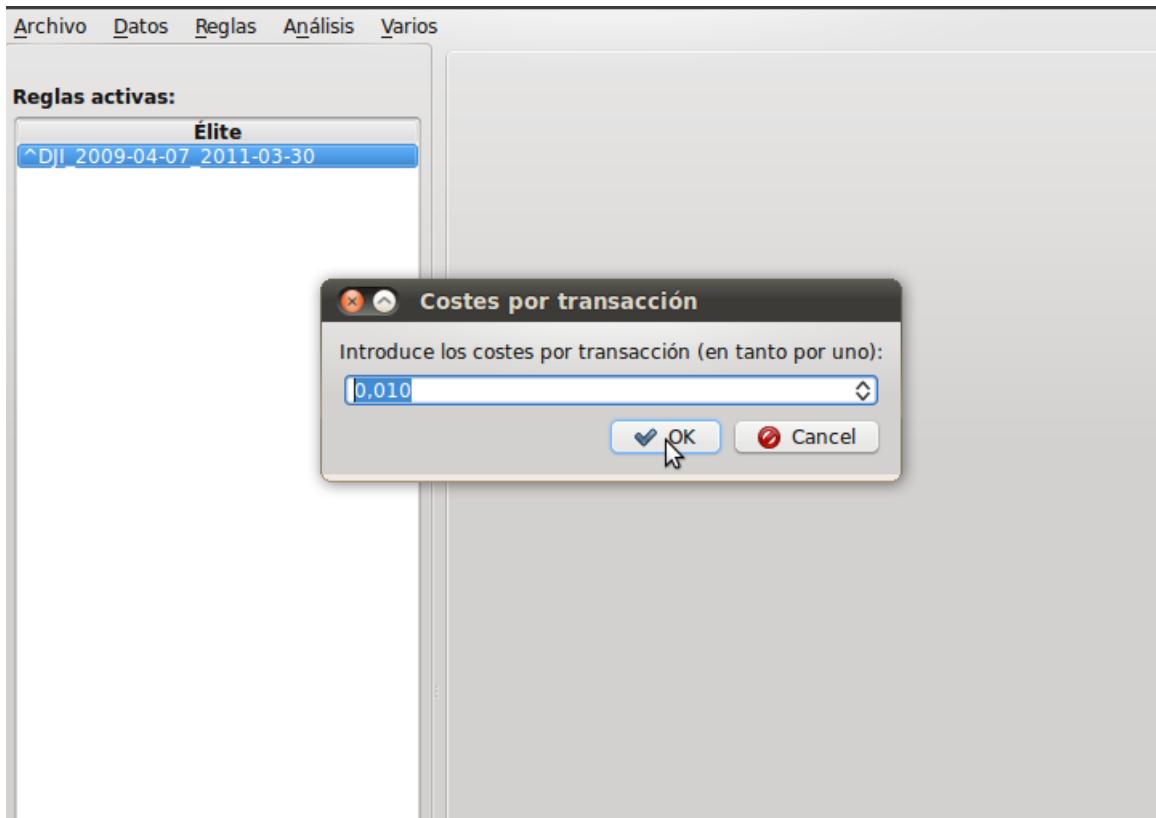
5. Escoger la fecha inicial y final entre las que esta comprendido en periodo de test.



6. Escoger el conjunto de reglas del cual se quiere saber su comportamiento en el periodo de test.



7. Introducir los costes por transacción que se usarán en el test.



8. A continuación se muestran los resultados del test en la pantalla principal. En la parte superior se muestran los beneficios y el número de transacciones de cada regla que compone el conjunto y del voto mayoritario de todas las reglas. En la parte inferior se muestran las estadísticas del conjunto de reglas.

Reglas activas:

Élite

^DJI 2009-04-07 2011-03-30

ÍNDICE: ^DJI

VALOR: ^DJI

PERIODO: 2010-01-01 - 2011-12-31

COSTES POR TRANSACCIÓN: 0.01

CJTO. REGLAS: ^DJI_2009-04-07_2011-03-30

Exportar datos

ID	Beneficio	Número de transacciones
1	8289	0
2	8290	0.166568
3	8291	0
4	8292	0.153516
5	8293	0.0431669
VM*	- 0.0689224	4



*VM: Voto mayoritario.

Una transacción equivale a una compra o una venta, con lo cual siempre habrá un número par.

NOTA: Todos los beneficios están indicados en tanto por uno respecto a la estrategia "buy-and-hold".

	Media	Desviación estándar	Mediana	Máximo	Mínimo
1	0.0726502	0.081831	0.0431669	0.166568	0

APÉNDICE C: DATOS DE LOS EXPERIMENTOS

En este apéndice se muestran los datos numéricos con los que han sido construidos los gráficos mostrados en el capítulo 8.

Cada índice estudiado (DJI y IBEX) cuenta con una tabla, en la que se muestra, para cada coste por transacción utilizado en la generación, los beneficios dados respecto a *buy and hold* en tanto por uno y el número de transacciones efectuadas para cada coste por transacción en el test. En cada coste por transacción en test aparecen 5 resultados, correspondientes a cada una de las secuencias de experimentación, que aparecen ordenadas en las tablas según el número que se les dio en el capítulo 8. Los resultados corresponden a la actuación según el voto mayoritario de las 50 reglas de inversión que componen una población.

La última tabla muestra los beneficios que hubiera obtenido la estrategia *buy and hold* en cada uno de los períodos utilizados en los experimentos del capítulo 8.

IBEX

COSTE EN LA GENERACIÓN	0%	0,1%	0,5%	1%	2%	3%
COSTE EN TEST	0%					
	0,055093	8	0,000000	2	0,000000	2
	-0,284223	2	-0,275733	2	-0,318529	2
	-0,294805	78	-0,271547	8	-0,387507	6
	0,059826	20	0,119308	12	0,127359	6
	0,017668	26	-0,028744	4	0,000000	2
COSTE EN TEST	0,1%					
	0,044393	8	0,000000	2	0,000000	2
	-0,283088	2	-0,274632	2	-0,317258	2
	-0,455041	78	-0,284393	8	-0,394345	6
	0,031164	20	0,101741	12	0,119909	6
	-0,024424	26	-0,032024	4	0,000000	2
COSTE EN TEST	0,5%					
	0,003648	8	0,000000	2	0,000000	2
	-0,278595	2	-0,270273	2	-0,312222	2
	-0,888651	78	-0,333101	8	-0,420442	6
	-0,071674	20	0,036115	12	0,091223	6
	-0,170962	26	-0,044747	4	0,000000	2
COSTE EN TEST	1%					
	-0,042910	8	0,000000	2	0,000000	2
	-0,273078	2	-0,264921	2	-0,306039	2
	-1,145960	78	-0,388293	8	-0,450362	6
	-0,176952	20	-0,036293	12	0,057758	6
	-0,312756	26	-0,059784	4	0,000000	2
COSTE EN TEST	2%					
	-0,122913	8	0,000000	2	0,000000	2
	-0,262368	2	-0,254531	2	-0,294037	2
	-1,282310	78	-0,481641	8	-0,501936	6
	-0,327179	20	-0,153609	12	-0,001819	6
	-0,496547	26	-0,087138	4	0,000000	2
COSTE EN TEST	3%					
	-0,187722	8	0,000000	2	0,000000	2
	-0,252074	2	-0,244544	2	-0,282500	2
	-1,270090	78	-0,555297	8	-0,543671	6
	-0,419717	20	-0,241012	12	-0,052596	6
	-0,593645	26	-0,111145	4	0,000000	2

DJI

COSTE EN LA GENERACIÓN	0%	0,1%		0,5%		1%		2%		3%	
COSTE EN TEST	0%										
	0,000000	2	0,000000	2	0,000000	2	0,000000	2	0,000000	2	0,000000
	-0,099101	14	-0,052337	10	-0,058729	2	-0,154219	2	-0,012446	4	-0,060095
	-0,216216	2	-0,208173	4	-0,203673	2	-0,251234	2	-0,255491	0	-0,255491
	0,199700	96	0,018484	8	0,042361	14	0,226612	2	0,226612	2	0,307051
	0,282838	206	0,063281	4	0,000000	2	0,000000	2	0,000000	2	0,000000
COSTE EN TEST	0,1%										
	0,000000	2	0,000000	2	0,000000	2	0,000000	2	0,000000	2	0,000000
	-0,123751	14	-0,069632	10	-0,058495	2	-0,153603	2	-0,016957	4	-0,059855
	-0,215353	2	-0,211506	4	-0,202860	2	-0,250231	2	-0,250479	0	-0,250479
	0,046527	96	0,009958	8	0,024824	14	0,225707	2	0,225707	2	0,309818
	-0,257863	206	0,057472	4	0,000000	2	0,000000	2	0,000000	2	0,000000
COSTE EN TEST	0,5%										
	0,000000	2	0,000000	2	0,000000	2	0,000000	2	0,000000	2	0,000000
	-0,214673	14	-0,134738	10	-0,057566	2	-0,151165	2	-0,034463	4	-0,058905
	-0,211935	2	-0,224379	4	-0,199640	2	-0,246259	2	-0,230631	0	-0,230631
	-0,337441	96	-0,022492	8	-0,039981	14	0,222124	2	0,222124	2	0,320773
	-1,101670	206	0,034904	4	0,000000	2	0,000000	2	0,000000	2	0,000000
COSTE EN TEST	1%										
	0,000000	2	0,000000	2	0,000000	2	0,000000	2	0,000000	2	0,000000
	-0,312606	14	-0,207570	10	-0,056426	2	-0,148172	2	-0,055172	4	-0,057739
	-0,207738	2	-0,239467	4	-0,195686	2	-0,241383	2	-0,206261	0	-0,206261
	-0,534917	96	-0,059537	8	-0,110047	14	0,217726	2	0,217726	2	0,334223
	-1,255600	206	0,008151	4	0,000000	2	0,000000	2	0,000000	2	0,000000
COSTE EN TEST	2%										
	0,000000	2	0,000000	2	0,000000	2	0,000000	2	0,000000	2	0,000000
	-0,464673	14	-0,328225	10	-0,054213	2	-0,142360	2	-0,092907	4	-0,055474
	-0,199591	2	-0,266506	4	-0,188012	2	-0,231916	2	-0,158952	0	-0,158952
	-0,620489	96	-0,123081	8	-0,219681	14	0,209187	2	0,209187	2	0,360335
	-1,231160	206	-0,040778	4	0,000000	2	0,000000	2	0,000000	2	0,000000
COSTE EN TEST	3%										
	0,000000	2	0,000000	2	0,000000	2	0,000000	2	0,000000	2	0,000000
	-0,570671	14	-0,420828	10	-0,052086	2	-0,136775	2	-0,126106	4	-0,053298
	-0,191760	2	-0,289690	4	-0,180635	2	-0,222817	2	-0,113481	0	-0,113481
	-0,611760	96	-0,174409	8	-0,297207	14	0,200979	2	0,200979	2	0,385432
	-1,183260	206	-0,084066	4	0,000000	2	0,000000	2	0,000000	2	0,000000

IBEX

SECUENCIA	1			2			3			4			5		
	Inicio	Fin	Beneficio (%)												
ENTRENA-MIENTO	01/08/1995	31/07/1999	178,2	01/08/1997	31/07/2001	24,5	01/08/1999	31/07/2003	-25,2	01/08/2001	31/07/2005	19,4	01/08/2003	31/07/2007	106,5
VALIDACIÓN	01/08/1999	31/07/2001	-9,7	01/08/2001	31/07/2003	-17,1	01/08/2003	31/07/2005	44,0	01/08/2005	31/07/2007	43,3	01/08/2007	31/07/2009	-25,5
TEST	01/08/2001	31/07/2003	-17,1	01/08/2003	31/07/2005	44,0	01/08/2005	31/07/2007	43,3	01/08/2007	31/07/2009	-25,5	01/08/2009	31/07/2011	-10,1

DJI

SECUENCIA	1			2			3			4			5		
	Inicio	Fin	Beneficio (%)												
ENTRENA-MIENTO	01/08/1995	31/07/1999	126,3	01/08/1997	31/07/2001	28,0	01/08/1999	31/07/2003	-13,3	01/08/2001	31/07/2005	1,1	01/08/2003	31/07/2007	43,1
VALIDACIÓN	01/08/1999	31/07/2001	-1,2	01/08/2001	31/07/2003	-12,3	01/08/2003	31/07/2005	15,3	01/08/2005	31/07/2007	24,1	01/08/2007	31/07/2009	-30,6
TEST	01/08/2001	31/07/2003	-12,3	01/08/2003	31/07/2005	15,3	01/08/2005	31/07/2007	24,1	01/08/2007	31/07/2009	-30,6	01/08/2009	31/07/2011	32,4

