TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

UNIVERSIDAD DE ALMERIA

ESCUELA SUPERIOR DE INGENIERÍA

"Acelerando la configuración óptima de sistemas automáticos de trading"

Curso 2016/2017

Alumno/a:

Manuel Segura Sagredo

Director/es:

Gracia Ester Martín Garzón

Gloria Ortega López

A mis padres

Agradecimientos

Dar las gracias a Olga por aguantar que le hable de trading y de paralelismo y por estar siempre a mi lado.

Gracias a Ester y a Gloria por sus sugerencias y valiosísimas aportaciones y por su confianza. Les estoy muy agradecido por todo el tiempo que me han dedicado. Ha sido un placer trabajar con ellas.

Contenido

1 Introducción 7

1.1 Concepto de Trading y Trading automático 7

1.2 Motivación 11

1.3 Objetivos del TFG y cronograma 14

2 Plataformas de trading 17

2.1. Plataformas de Trading 17

3 Metatrader 4 21

3.1 Descripción 21

3.2 Proceso de Creación de un robot en metatrader 24

3.2.1 Fase de Desarrollo de robot 24

3.2.2 Fase de optimización del robot utilizando datos históricos 27

3.2.3 Fase de Lanzamiento en modo Prueba (Test Mode) 32

3.2.4 Fase de prueba en tiempo real (Forward Test) 35

3.2 Descripción de los posibles algoritmos de la fase de optimización 39

3.2.1 Algoritmo de búsqueda exhaustiva 39

3.2.2 Algoritmo genético 39

4. Ejemplo de Sistema automático en metatrader: “Orden perfecto” 45

4.1. Descripción del sistema “Orden perfecto” 45

4.2. Fase de Optimización del sistema “Orden perfecto” 49

5. Estrategias para incrementar el rendimiento en la Fase de optimización 55

5.1. Optimizando el Secuencial 57

5.2. Extracción de paralelismo 60

5.3. Utilizando estructuras de datos para evitar cálculos redundantes 65

6. Resultados obtenidos y trabajos futuros 69

6.1 Evaluación de la optimización del código secuencial 70

6.2 Evaluación del paralelismo 71

6.3 Evaluación de uso estructuras de datos auxiliares 71

6.4 Trabajo futuros 73

7. Conclusiones 75

BIBLIOGRAFIA 77

RECURSOS ONLINE 78

Figuras

Figura 1.1. Bróker al teléfono. 9
Figura 1.2. Planificación Temporal 16
Figura 3.1. Pantalla de trabajo de Metratrader 22
Figura 3.2. Pantalla de datos históricos. 23
Figura 3.3. Entorno de desarrollo de Metatrader 25
Figura 3.4. Esqueleto de robot o Expert Advisor 27
Figura 3.5. Diagrama de funcionamiento en la fase de Optimización 28
Figura 3.6. Ventana de propiedades del experto 29
Figura 3.7. Ventana de Optimización 30
Figura 3.9. Selección de la estrategia de optimización 32
Figura 3.10. Tabla de resultados del modo Tester. 33
Figura 3.11. Gráfico con el Balance en el modo Tester 34
Figura 3.12. Informe del modo Tester 34
Figura 3.13. Diagrama de funcionamiento de Metatrader en modo Tester 35
Figura 3.14. Diagrama de funcionamiento de Metatrader en modo Tiempo real 38
Figura 3.15. Cadena de valores o Cromosoma. 41
Figura 4.1. Cadena de valores o Cromosoma. 46
Figura 4.2. Indicador Adx sobre un gráfico del EURUSD 46
Figura 4.3. Indicador Bandas de Bollinger sobre un gráfico del EURUSD 47
Figura 4.4. Función Ontick() 48
Figura 4.5. Funciones auxiliares de Ontick() 49
Figura 4.6. Parámetros de entrada a optimizar. 50
Figura 4.7. Optimización usando búsqueda exhaustiva 51
Figura 4.8. Optimización usando algoritmo genético 51
Figura 4.9. Optimización finalizada usando algoritmo genético 52
Figura 4.10. Resultado de la optimización 53
Figura 5.1. Distribución de tiempo en el modo tester de Metatrader 56
Figura 5.2. Código fuente original del sistema “orden perfecto” implementado en Metatrader. 57
Figura 5.3. Extracto de la implementación de la media simple 59
Figura 5.4 Enlace de las funciones de la librería en un sistema en Metatrader. 59
Figura 5.5. Extracto de la implementación de la clase que gestiona las hebras. 61
Figura 5.6. Extracto de la implementación del método Execute de una hebra de la librería. 62
Figura 5.7. Fragmento de código de inicialización de la librería 63
Figura 5.8 Ticks de Cpu y número de funciones ejecutadas por hebra 64
Figura 5.9 Código que hace la llamada al cálculo de las funciones. 64
Figura 5.10. Detalle del código original y las llamadas en secuencial a las funciones. 65
Figura 6.1 Tiempo de ejecución del sistema “Orden Perfecto” en modo Tester 70
Figura 6.2 Tiempo de ejecución del sistema “Orden Perfecto” en modo Tester 70
Figura 6.3 Tiempo de ejecución del sistema “Orden Perfecto” usando paralelismo 71
Figura 6.4 Tiempo de ejecución del sistema “Orden Perfecto” almacenando resultados 72
Figura 6.5 Optimización del sistema incluyendo las 3 mejoras. 73

Tablas

Tabla 1.1. Fases del proyecto 15
Tabla 6.1 Tiempo de ejecución del sistema “Orden Perfecto” con distintas hebras. 71

1 Introducción

En este capítulo vamos a explicar en que consiste la inversión en mercados financieros o "trading" de forma automatizada mediante el uso de ordenadores y software, a éste software lo llamaremos robot de trading aunque también nos podemos referir a él como sistema automático de trading o "expert advisor". Como veremos cada robot es capaz de implementar una estrategia de trading particular que consta de una serie de reglas objetivas definidas a priori por el usuario. Además se justificará que determinar la configuración óptima de estos robots requiere de muchos recursos computacionales. Por tanto, se motivará la necesidad de aplicar técnicas de computación de alto rendimiento en este contexto.

1.1 Concepto de Trading y Trading automático

Trading viene de la palabra inglesa "Trade" que significa intercambio de bienes en una operación, cuando los bienes que se intercambian son activos financieros se le llama Trading financiero, es decir inversión financiera.

El trading financiero consiste en comprar o vender un valor subyacente en un mercado financiero con la intención de obtener un beneficio especulativo.

En resumen: compramos algo con la intención de poder venderlo más caro, o bien pactamos una venta primero con la intención de poder comprarlo más tarde más barato.

Se puede hacer con acciones, con futuros referenciados a índices bursátiles, con materias primas y también con divisas. Cuando lo hacemos con divisas lo hacemos en el mercado Forex, o mercado de divisas, y siempre se hace en una divisa respecto de otra. Por ejemplo el euro frente al dólar (EURUSD), o la libra frente al dólar (GBPUSD). Desde la desaparición del Patrón Oro para conocer el valor de una divisa lo debemos comparar con otra divisa. De ahí que siempre se hable de Pares de divisas.

Podemos realizar un trading en el día, en el cual nuestras posiciones se abren o cierran típicamente dentro del mismo día, o de más largo plazo, con posiciones que pueden durar días semanas o incluso meses.

Desde hace algunos años, y gracias al desarrollo de nuevas tecnologías basadas en Internet, y a través de un servicio de Bróker, el cual nos proporciona los datos del mercado y a su vez recibe nuestras órdenes de compra y venta, cualquiera que disponga de unos ahorros, un ordenador y una conexión a Internet puede realizar trading.

Sin embargo, esta facilidad de acceso al trading no implica que el trading sea una actividad sencilla. Conseguir regularidad en nuestros resultados de una forma sostenida en el tiempo es mucho más difícil de lo que en un principio podría parecer.

Para la realización de esta actividad de inversión o como lo llamaremos a partir de ahora: "Trading" se requiere un sistema de trading. Esto es un conjunto de reglas precisas que nos digan cuando entrar en una operación y cuando salir de la misma. A esa notificación de cuando entrar y salir de la operación se le suele llamar señal de entrada o salida del mercado. Estas señales son en su mayoría generadas por indicadores técnicos o una combinación de indicadores técnicos (basados en cálculos estadísticos en su mayoría).

Normalmente todo sistema de trading tiene como objetivo principal maximizar beneficios en relación al dinero invertido, pero no suele ser el único objetivo ya que también nos interesa que el riesgo soportado sea el mínimo posible, también puede ser de interés la

regularidad del sistema, es decir nos interesa que el sistema nos de beneficios pero de la forma lo más lineal posible.

Los sistemas de trading pueden aplicarse básicamente de 2 posibles formas:

Manual: Donde una persona o más aplica unas reglas decidiendo cuando entrar al mercado en algún activo (acciones de bolsa, índices, divisas, materias primas etcétera) y cuando salir del mismo. Actualmente la mayoría de los "traders" (operador de bolsa) operan de forma manual sus sistemas, es decir observan los indicadores económicos o señales de trading en forma de gráfico (ej. cuando la media de 100 días cruza la media de 200 días) y si identifican algún patrón de compra o venta, de forma manual pulsan la opción de compra o venta en su plataforma de Trading y se envían las señales correspondientes al agente de bolsa o "bróker" (antes de la llegada de Internet esto se hacía de forma telefónica, de ahí la típica imagen del "bróker" en las películas en el edificio de la bolsa con uno o varios teléfonos en la mano, recibiendo órdenes de los clientes. Ver Figura 1.1.

Figura 1.1. Bróker al teléfono.

Automática: Ocurre cuando las reglas de trading están monitorizadas por un software, el cual es el encargado de enviar las ordenes al bróker tanto de entrada como de salida del mismo (compra/venta) de una forma automática, o con muy poca intervención del usuario. Es decir las decisiones se basan en algoritmos matemáticos. Este tipo de trading está en gran crecimiento en todo el mundo y ya ocupa un porcentaje importante del volumen de trading mundial.

Podríamos considerar sistemas de trading de tipo **semiautomático**, donde la entrada podría ser decidida por un sistema automático y la salida de la operación por un humano o al revés, el humano decide el momento de la entrada al mercado y es el sistema

automático, también llamado robot, el que decide cuando y como salir (salida de todo el montante de la operación o varias salidas parciales en distintos niveles etcétera).

El trading automático hace unos años estaba reservado a bancos y grandes fondos de inversión, pero hoy día gracias a la evolución de la tecnología, cualquier persona con un ordenador personal con acceso a internet y una cuenta en un bróker (desde 100 euros ya se puede abrir) puede operar con un sistema automático de trading.

Para poder hacer trading, tanto manual como automático, una persona necesita tener en su ordenador un software que es el que hará la comunicación con el Bróker, le enviará las órdenes de compra, venta etc. Este software lo conocemos como plataforma de trading. Estas plataformas a veces en lugar de tenerlas en el ordenador particular del cliente son alquiladas en un servidor remoto (VPS :"Virtual Private Server"). Esto suele ser lo más recomendable sobre todo cuando operamos con robots (trading automático) ya que normalmente estarán encendidos 24 horas x 5 días a la semana y en un centro de datos hay conexión a internet de calidad: rápida, estable y tienen sistemas de copia de seguridad y están preparados para posibles caídas de alimentación eléctrica etc.

Recientemente, también las plataformas de trading ofrecen entornos de programación que permiten definir nuestras propias reglas o indicadores de trading mediante lenguajes visuales para usuarios poco expertos en programación o también con lenguajes de programación tradicionales como C, Java, Pascal etc. Este hecho ha potenciado el trading automático, haciendo que haya multitud de robots e indicadores tanto de pago como de código abierto [1,10]

Un sistema de trading automático funciona en tiempo real y realmente no necesita muchos recursos computacionales, ya que suelen ser programas que evalúan parámetros estadísticos relacionados con muestras de varios cientos de datos. Por ejemplo, comparan 2 medias de 200 datos. Este tipo de cálculos se pueden completar en un ordenador personal actual sin problemas en un tiempo relativamente pequeño, de manera que es posible construir un sistema trading automático capaz de actuar en tiempo real.

Ventajas de un sistema de trading automático frente a manual:

- Una de las ventajas del trading automático sobre el manual es que podemos probar nuestro sistema de trading sobre datos históricos (por ejemplo los últimos 10 años) y la plataforma nos dará una estadística sobre el porcentaje de decisiones acertadas y fallidas, el beneficio que hubiéramos acumulado etc. Este tipo de prueba (backtesting) de forma manual nos llevaría un considerable tiempo.

- Otra de las ventajas que le hacen muy popular es que el trading automático está exento de emociones que suele ser uno de los grandes problemas en el trading manual sobre todo cuando se gestionan grandes patrimonios ocurriendo a veces que el operador se siente fuertemente presionado por las emociones (miedo, codicia, avaricia) y no aplica las reglas del sistema de forma metódica aún habiéndolas probado de antemano.

- Por último, otra ventaja del trading algorítmico o automático es que dado su velocidad de proceso, podemos abarcar muchos más activos de los que podríamos abarcar de forma manual.

1.2 Motivación

El sistema de trading automático que hemos comentado anteriormente, se basa en una serie de parámetros de configuración (o variables) que deben ser optimizados para garantizar que se obtenga el máximo beneficio con la mínima pérdida. La búsqueda de dichos óptimos podría consumir un tiempo de cómputo considerable.

Por norma general, las variables o parámetros de configuración [2] que se tienen en cuenta a la hora de diseñar un sistema automático de trading serán:

-Las variables de tiempo (time - frame, sesión europea, sesión americana, sesión asiática, hora de operativa …)
-Variables de precio y su comportamiento (acción del precio: máximos, mínimos etcétera)
-Variables de volatilidad (la cantidad de movimiento que realiza el precio en un momento dado)
-El volumen de negociación en un momento dado.

Para ilustrar el uso de estas variables, se presenta un ejemplo de un sistema de trading automático definido por una serie de reglas:

Regla1 : Regla de entrada:"voy a comprar 1 contrato del Indice español "Ibex35" cuando al cierre del día la media del precio de 10 días sea mayor que la media de 50 días"..

Regla2 : Ahora definiremos otra regla para cerrar esta operación que hemos abierto y podría ser: "voy a cerrar la operación (venderiamos el contrato del Ibex35 que tenemos) cuando consiga 10 puntos de beneficio (1 punto en el Ibex35 vale 10 euros) ". Esta regla se aplicaría claramente si el precio va a favor nuestro.

Regla3: Puede ocurrir que entremos en el mercado y el precio vaya en contra nuestra, en este caso nunca sabemos hasta cuando el precio puede ir en contra nuestra así que necesitamos un límite de pérdidas, esto se llama "Stop Loss", por ejemplo puede ser de otros 10 puntos, es decir cuando el precio caiga 10 puntos por debajo de nuestro nivel de entrada, el sistema venderá nuestro contrato y perderemos 10 puntos(x10 euros el punto, serán 100 euros de pérdida).

Observando ese sistema de ejemplo, ¿Por qué hemos elegido esos números tanto en las reglas de entrada como de salida? ¿Porque las medias ideales son 10 días y 50 días? ¿y si fueran 20 y 60 días? ¿Cuál es el beneficio óptimo antes de vender los activos, porqué he elegido 10 puntos en lugar de 15? Si el sistema está en pérdidas ¿Sería más beneficioso esperar a que pierda 20 puntos en lugar de 10 por si a los 15 se diera la vuelta y el precio fuera a nuestro favor?

En un sistema con sólo 3 reglas aparecen gran cantidad de variables con muchas combinaciones posibles. La identificación de los valores específicos de éstas variables que maximicen el beneficio y minimicen el riesgo simultáneamente se define como la optimización del sistema. Estos valores óptimos son los que definen la actuación del robot para las reglas definidas.

Una estrategia para ajustar los parámetros de configuración, consiste en evaluar las posibles configuraciones con los datos históricos y determinar los parámetros que habrían optimizado la actuación de la plataforma durante el tiempo simulado. Estos valores se pueden tomar como parámetros de configuración actuales, ya que han sido valores óptimos en el pasado es razonable pensar que pueden darnos buenos resultados en el futuro, sobre todo si se utilizan los suficientes datos estadísticos, es decir a más años estudiados, mas fiables serán los resultados.

La configuración u optimización del sistema trading es un proceso con gran demanda computacional y debe completarse con cierta frecuencia para garantizar que el sistema trading se pueda adaptar a las variaciones del mercado. Por ejemplo, para completar la simulación del sistema estudiando 5 años de datos históricos con una configuración particular, se han de procesar 70 millones de cambios de precio aproximadamente (o sea el precio cambia de valor esas veces), el robot en función de éste precio variable y otras variables como volatilidad, hora, volumen decide si entra en el mercado y en caso afirmativo ejecuta la orden y decide cuando cerrarla. Esta simulación deberá repetirse para cada configuración de variables que se evalúe, así que si teníamos una función con una variable (el precio en cada momento) que ya sabemos que cambia unos 70 millones de veces en 5 años y queremos optimizar más variables debemos multiplicar el tiempo de cálculo por las combinaciones de variables que tengamos.

Por tanto, la configuración del sistema se define como un problema de optimización biobjetivo. Es decir, dado el conjunto de variables de configuración $\mathbf{x}=(x_1, x_2,...,x_n)$ se pretende identificar qué combinación de variables **x** obtienen un valor máximo del beneficio G(x) y un riesgo mínimo R(x).

El tiempo para resolver este problema se dispara a medida que aumenta el número de variables de configuración, n. Todas las plataformas dan la posibilidad de usar algoritmos

genéticos para buscar estos óptimos sin tener que hacer todas las combinaciones, sacrificando casos de prueba pero dan un resultado razonablemente bueno en un plazo de tiempo admisible.

La reducción del tiempo para configurar un sistema trading es de gran interés y se puede basar en las siguientes grandes líneas:

1.-Aplicación de métodos heurísticos de optimización biobjetivo que reduzcan el número de simulaciones para determinar los parámetros óptimos de configuración.

2.-Aplicación de técnicas computacionales de alto rendimiento, computando en paralelo varias simulaciones y/o cada simulación.

En este trabajo nos vamos a centrar en la segunda estrategia.

1.3 Objetivos del TFG y cronograma

El objetivo principal de este trabajo fin de grado será la optimización computacional de sistemas de trading mejorando el tiempo de respuesta de la plataforma trading MetaTrader4 aplicando técnicas de computación de alto rendimiento.

Hay que destacar que Metatrader4 es un software diseñado para sistemas operativos de 32 bits y con no muchos recursos de memoria 1 ó 2 Gigabytes de RAM, y pensado para procesadores con un solo núcleo.

Para mejorar del rendimiento de la plataforma primeramente se identificarán los distintos niveles de paralelismo que ofrece la aplicación y posteriormente se seleccionarán aquellas arquitecturas más adecuadas para la explotación de los mismos. Cabe destacar se

desea que la nueva versión de la plataforma sea portable y accesible para usuarios particulares que disponen de equipos personales de sobremesa. Por tanto, en principio se plantea que el paralelismo de los códigos sea explotado por los procesadores multi-core.

Este objetivo general se estructura en las siguientes metas específicas:

a) Revisar los modelos principales en los que se basan los sistemas automáticos de trading, centrándonos especialmente en el modelo utilizado por Metatrader 4 , debido a su enorme difusión a nivel internacional.[3]
b) Analizar el código de un sistema de trading en la plataforma Metatrader 4 identificando los procedimientos que deterioran el rendimiento de la aplicación, así como los distintos niveles de paralelismo de la aplicación.
c) Optimizar la versión secuencial de las rutinas más costosas
d) Desarrollar la versión multicore de las rutinas más costosas
e) Evaluar las mejoras del rendimiento de la plataforma basada en las nuevas rutinas.

Fases	Fecha Inicio	Fecha Fin
1. Revisión de los modelos implementados por las plataformas trading.	03/09/2016	12/09/2016
2. Análisis del sistema de trading en la plataforma trading Metatrader 4.	12/09/2016	21/09/2016
3. Identificación de las rutinas con más carga computacional.	24/09/2016	28/09/2016
4. Optimización de estas rutinas.	29/09/2016	13/10/2016
5. Validar el funcionamiento del sistema de trading con la nuevas rutinas.	14/10/2016	20/10/2016
6. Evaluar el rendimiento del sistema basándose en las rutinas optimizadas.	21/10/2016	27/10/2016
7. Redacción de la memoria.	28/10/2016	25/11/2016

Tabla 1.1. Fases del proyecto

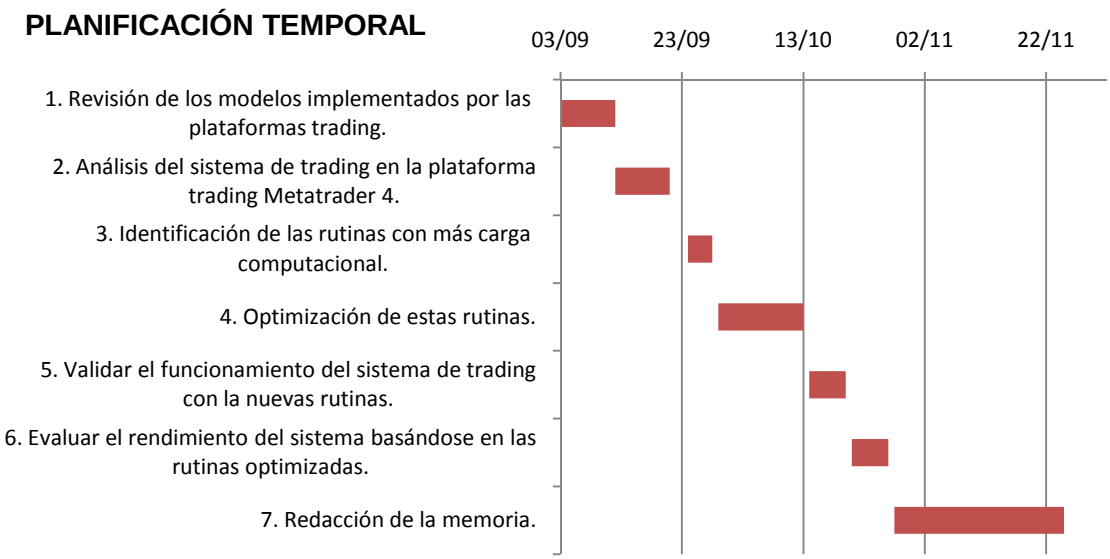


Figura 1.2. Planificación Temporal

2 Plataformas de trading

En este capítulo vamos a analizar las plataformas de trading más extendidas del mercado que nos permiten el desarrollo de robots. Nuestra atención se centra en Metatrader 4 dado que es la plataforma con más difusión a nivel mundial debido a su potencia y flexibilidad en el desarrollo de robots de trading.

2.1. Plataformas de Trading

Como hemos comentado en el Capítulo 1, una plataforma de trading, es un software que nos permite operar en los mercados (hacer trading), para ello envía las órdenes de compra, venta, de stop de perdidas, de toma de beneficios al bróker, siendo finalmente el bróker el que lanza las ordenes al mercado, para ello el bróker nos cobra una comisión.

Con el avance de la tecnología, este software se hace cada vez más sofisticado, incorporando cada vez más nuevas características para hacernos cada vez más fácil el proceso del trading, no siendo ya solo un mero intercomunicador con el bróker, sino que incorpora características de Análisis de los activos, Plataforma de desarrollo de indicadores, sistemas automáticos (robots), incluso características de red social para ponernos en contacto con otros traders e intercambiar ideas, robots, etc.

Las características comunes que podemos encontrar en las modernas plataformas de trading suelen ser:

-Gráficos del activo (barras, velas, líneas u otros tipos) donde se muestra el precio del activo (una acción o una divisa, por ejemplo EURUSD, o un índice, por ejemplo el DAX de Frankfurt) en función del tiempo.

-Herramientas de análisis: posibilidad de marcar sobre el gráfico distintas líneas y marcas (líneas de tendencia, soportes y resistencias, retrocesos de Fibonacci etc), ésto es utilizado por los inversores para poder detectar puntos clave en el precio, donde éste se puede girar etc.

-Posibilidad de añadir indicadores, que no son más que algoritmos mátematicos, los cuales nos pueden dar alguna señal que puede ser tan directa como Compra o Vende o una función en el tiempo y que podemos superponer en el gráfico del activo, como pueden ser medias móviles (ponderadas, exponenciales etc), medidores de volatilidad, medidores de tendencia etc. Estos son los instrumentos que ayudan al operador a saber qué está pasando en el mercado y le ayudan en la toma de decisiones, un ejemplo típico es el cruce de 2 medias móviles, muchos operadores entran al mercado si la media móvil de 100 periodos se cruza con la media móvil de 200 periodos entendiendo que una tendencia alcista se está formando etc.

-Centro de noticias: en tiempo real la plataforma va recibiendo noticias económicas que pueden influir en el mercado como puede ser: publicaciones de Datos de desempleo en los distintos paises, Cambios en los tipos de interés las distintas divisas etc. ya que los mercados tienen una influencia directa con las decisiones económicas de los gobiernos, bancos centrales etc.

-Plataforma de desarrollo: casi todas las plataformas modernas también tienen algún tipo de lenguaje de programación que le permite hacer desde pequeños programas o scripts hasta completos sistemas de trading automático. Como hemos comentado en el apartado 1, las hay que utilizan lenguajes estándar como por ejemplo VisualChart el cual utiliza VBA (Visual Basic para aplicaciones) o Metatrader que aunque su lenguaje es llamado

MQL (Metaquotes Language) es lenguaje C con algunos complementos para poder hacer trading, otras han desarrollado su propio lenguaje de scripts y otras han desarrollado incluso lenguajes visuales con diagramas de flujo para programadores no expertos.

Las plataformas de trading más conocidas del mercado son:

- Pro Real Time: Es un software que no se instala en nuestro ordenador sino que está en la nube y se descarga en el momento de ejecución, con lo cual siempre está actualizado, tiene una excelente interfaz gráfica con muchas funciones. Al ser una herramienta Online no permite hacer análisis en modo desconectado sino que requiere estar siempre conectado, pero hoy día no es un gran inconveniente. Respecto al interfaz de programación, tiene un lenguaje propio para desarrollo de indicadores y sistemas llamado Probuilder parecido al Basic que es bastante visual e intuitivo. El software en sí es gratuito, se suele pagar mensualmente por los datos que o bien nos suscribimos al servicio de datos del fabricante del software o por el hecho de ser cliente de algunos brokers nos lo incluyen como parte de sus servicios o nos cobran muy poco. [4]

- VisualChart: Está desarrollada en España, es una muy buena herramienta de análisis, es una aplicación Windows que se instala en nuestro ordenador, manteniendo los históricos de datos etc. La herramienta es gratuita e incluso si no queremos datos en tiempo real ya que hacemos una operativa a largo plazo (ejemplo diaria) nos dan los datos al finalizar el día de forma gratuita, es decir los movimientos de los mercados de cada día, nos dan acceso al finalizar el día. En caso de querer datos en tiempo real, entonces sí tendremos que pagar una suscripción. También disponen de una herramienta en Java que no requiere instalación para poder acceder desde cualquier ordenador. Tiene 2 sistemas de programación, un lenguaje visual para los no iniciados en la programación y también tiene la posibilidad de desarrollar en VBA (Visual Basic para Aplicaciones de Microsoft) con una herramienta de depuración etc. El problema de Visual Chart es que es un devorador de memoria RAM y suele ser bastante lento al arrancar incluso con equipos potentes.[5]

- Metatrader: Desarrollado por la empresa Metaquotes en el año 2002. Es un software exclusivamente para Windows y como herramienta de análisis no es especialmente buena ya que tiene algunas limitaciones. Una ventaja muy importante es su gran velocidad, tanto de arranque como de funcionamiento y es bastante liviano en gasto de memoria. Además desde que salió al mercado contaba con una gran facilidad para desarrollar scripts y robots que automatizaban el trading, ya que su lenguaje de programación es una simplificación del lenguaje C, donde se minimiza el uso de punteros. Rápidamente se hizo muy popular entre los programadores de sistemas de trading y empresas de terceros que desarrollaban scripts y robots, tanto libres como comerciales. Los bróker se dieron cuenta de esta gran difusión que tuvo y no tardaron en ofrecerla como herramienta para sus clientes de forma gratuita. Hoy día es muy raro encontrar un bróker que no ofrezca Metatrader 4 entre sus plataformas de trading. De aquí en adelante al referirnos a Metatrader nos estaremos refiriendo siempre a la versión 4. [3]

En el siguiente capítulo, la herramienta Metatrader será descrita en detalle.

3 Metatrader 4

En este capítulo daremos una breve explicación de cómo funciona la herramienta Metatrader 4 y los pasos que hay que seguir en el desarrollo de un sistema automático de trading, o Expert Advisor como se le conoce en Metatrader.

3.1 Descripción

Popularmente conocido como MT4, Metatrader 4, como se ha comentado en el punto 2.1, ha sido desarrollado por la empresa Metaquotes lanzando su primera versión en el año 2002. Siendo su versión 4 lanzada en 2005 la más popular. A pesar de que en 2010 se liberó la versión 5, en 2013 casi ningún bróker la ofrecía debido a cambios importantes en el lenguaje de programación que hacían todo el desarrollo existente para las versiones previas incompatible, con lo cual los usuarios se ven obligados a reescribir el código de robots e indicadores, en la figura 3.1 podemos ver la pantalla de trabajo de Metatrader 4.

Figura 3.1. Pantalla de trabajo de Metratrader

La pantalla principal de Metatrader se compone de la parte central o principal que son los gráficos(charts) que estamos visualizando sobre algún o algunos activos: EURUSD etcétera y que podemos tenerlos en pestañas o las ventanas en modo mosaico. A la izquierda tenemos los indicadores y Expert Advisor(robots) que simplemente arrastramos al gráfico que estamos visualizando para hacerlos funcionar. Aunque todos estos bloques se pueden configurar, normalmente se suele ver abajo, la información sobre nuestra cuenta, cuánto dinero dispone etcétera y las operaciones de trading que tenemos abiertas en el momento. En la parte superior, como suele ser típico en las aplicaciones tenemos los botones y menús con las acciones: Zoom + y Zoom – en el gráfico, cambiar tipo de gráfico, abrir una orden etc.

Una característica a destacar de Metratader 4 que para muchos ha sido un paso atrás es que en la versión 5 es la existencia de un centro de datos históricos donde el usuario puede exportar los datos a Excel, importarlos de distintos bróker para probar los robots etc. Ver figura 3.2.

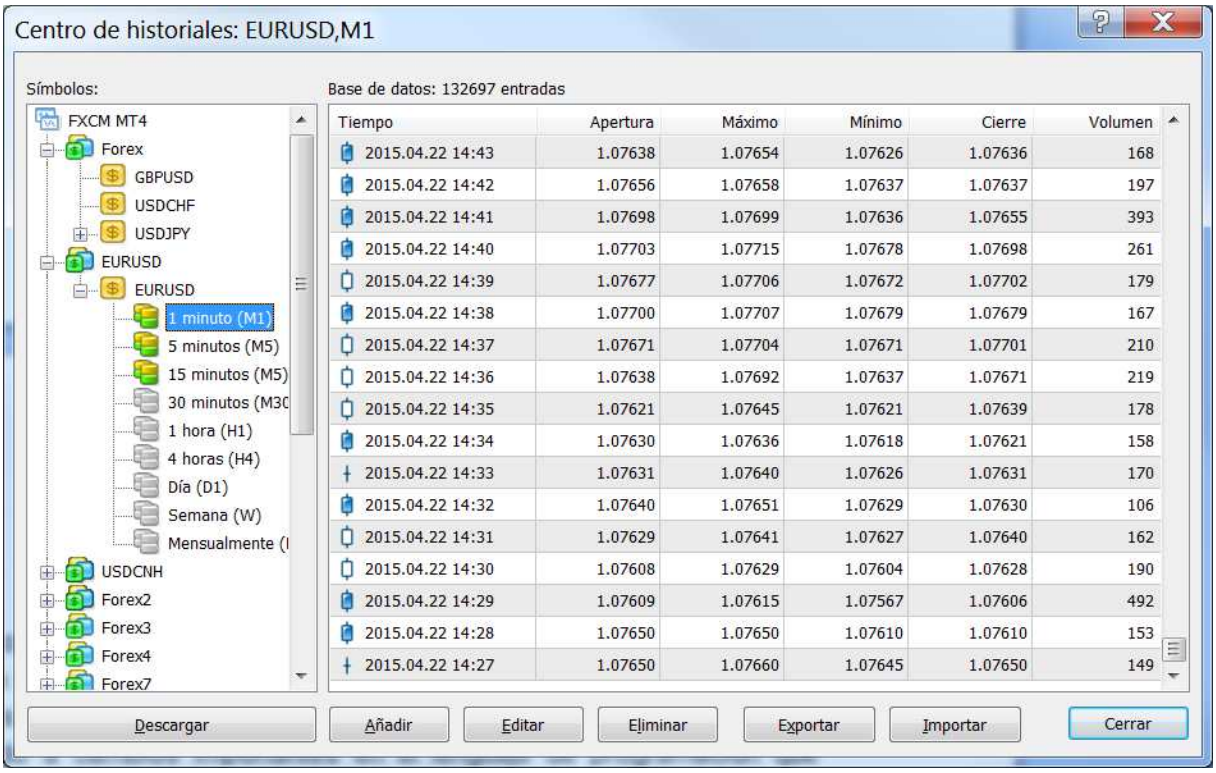


Figura 3.2. Pantalla de datos históricos.

En la nueva versión Metatrader 5, la ventana de datos históricos ya no está disponible al usuario, es decir no hay sitio donde poder verlos ni exportarlos, ni importarlos, simplemente se descargan de forma automática desde el servidor de metaquotes pero hay mercados donde no todos los brokers tienen exactamente el mismo precio ni la misma precisión con lo cual es importante probar el robots con distintos datos históricos. No todo es malo en la última versión ya que es una aplicación de 64 bits con posibilidad de usar todos los cores de la CPU en la optimización.

Lo ideal sería tener las posibilidades de programación y de uso de datos históricos de la plataforma Metatrader 4 con la velocidad de la versión 5, pero por ahora no es posible.

De ahí viene la motivación de este proyecto intentar dar a la versión 4 las posibilidades de velocidad de la versión 5 pero manteniendo la compatibilidad con el código existente tanto de robots como de indicadores, y si no es una compatibilidad de 100% pero que requiera los mínimos cambios en el código.

Metatrader 4 es un software de 32 bits y que sólo utiliza una hebra de ejecución, con lo cual aunque tengamos un procesador multicore, Metatrader 4 no utilizará el resto de cores del mismo y se limitará a usar uno sólo.

Luego si no es un programa multicore ¿Cómo haremos para mejorar su rendimiento en el cálculo de los parámetros óptimos del robot ?. Para este propósito vamos a utilizar otra de las facilidades de programación que da Metatrader 4, que es la posibilidad de llamar a librerías externas (DLL o Dynamic Link Libray en Windows), la idea es si en Metatrader no es posible la mejora de rendimiento, le pasaremos los datos a una librería externa que debe hacer los cálculos con más rendimiento, mejorando así el tiempo de CPU requerido para la optimización de un robots. [6]

3.2 Proceso de Creación de un robot en metatrader

En esta sección vamos a ver los pasos necesarios para la creación de un robot desde cero en Metatrader, lo que se conoce como un Expert Advisor o EA.

3.2.1 Fase de Desarrollo de robot

Esta parte es el desarrollo software en sí, simplemente nos vamos a Herramientas y pulsamos en Editor de Metaquotes Lenguaje (F4), y se nos abrirá el entorno de desarrollo, que no difiere mucho del entorno de desarrollo de cualquier lenguaje de programación: Java, Visual C etc. Entorno que podemos ver en la figura 3.3.

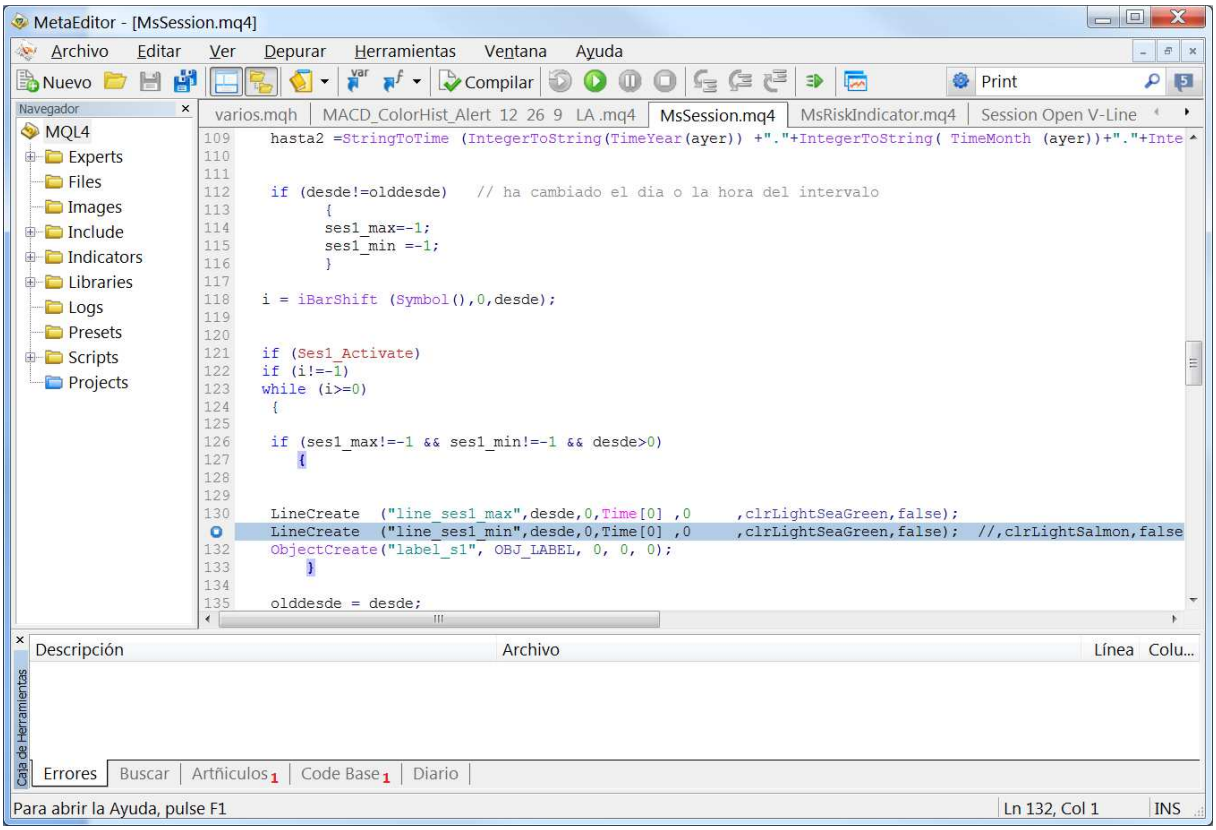


Figura 3.3. Entorno de desarrollo de Metatrader

En esta fase se procede como en cualquier desarrollo de software: procedemos a la codificación del robot, como hemos comentado antes es un lenguaje C ligeramente simplificado. El entorno contiene un depurador paso a paso, con puntos de ruptura, visor de variables etc.

Una vez terminada la codificación y corregidos los errores de sintaxis. Compilamos el programa generándonos una especie de ejecutable pero que tiene extensión .EX4, realmente no es un ejecutable en sí ya que siempre necesita estar dentro del entorno de Metatrader para ser ejecutado. Con esto ya tendríamos un robot que podría funcionar, pero antes de ponerlo en producción debemos probarlo en la fase de optimización.

En la figura 3.4 podemos ver lo que podríamos llamar el Esqueleto de un robot (Expert Advisor). Estos programas están orientados a eventos, es decir no tienen un main() como cualquier programa en C.

Hay al menos 3 eventos que son llamados por Metatrader y son:

- Una función o evento de inicialización: OnInit(), para inicializar variables, contadores etc, es llamado por el sistema al inicializar el programa por primera vez.

- Una función o evento donde se desarrolla el algoritmo del robot: OnTick(), este evento es llamado cada vez que el bróker registra un nuevo precio para el activo donde está operando el robot. Dentro de esta función estará el algoritmo que decide la entrada en el mercado y el que decide la salida del mismo. Normalmente llevará un código que hará de disparador para entrar al mercado "Trigger" y otro código para salir del mismo.

- Una función o evento de finalización: OnDeinit(), útil para liberar memoria, cerrar ficheros logs si los hubiera etcétera y cerrar la aplicación de forma ordenada. Este evento es llamado cuando termina la ejecución del programa.

```
//+------------------------------------------------------------------+
//|                                                      prueba.mq4  |
//|                                          Manuel Segura Sagredo   |
//|                                          https://www.mql4.com    |
//+------------------------------------------------------------------+
#property copyright "Manuel Segura Sagredo"
#property version   "1.00"
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//---

//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---

  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//---

  }
//+------------------------------------------------------------------+
```

Figura 3.4. Esqueleto de robot o Expert Advisor.

3.2.2 Fase de optimización del robot utilizando datos históricos

Esta fase se utiliza para buscar una configuración óptima de parámetros para el robot. Un ejemplos de parámetros podrían ser: la hora óptima de entrar al mercado, el nivel máximo de pérdida que debo aguantar si la posición va en mi contra, el nivel máximo que debo aguantar la posición en caso de ir en beneficio antes de cerrarla. Lo que vamos

buscando es maximizar una función objetivo dado un periodo de tiempo a estudiar, por ejemplo desde Junio de 2010 a Julio de 2015.

El diagrama de funcionamiento de Metatrader 4 para ésta fase de Optimización seria el correspondiente a la figura 3.5.

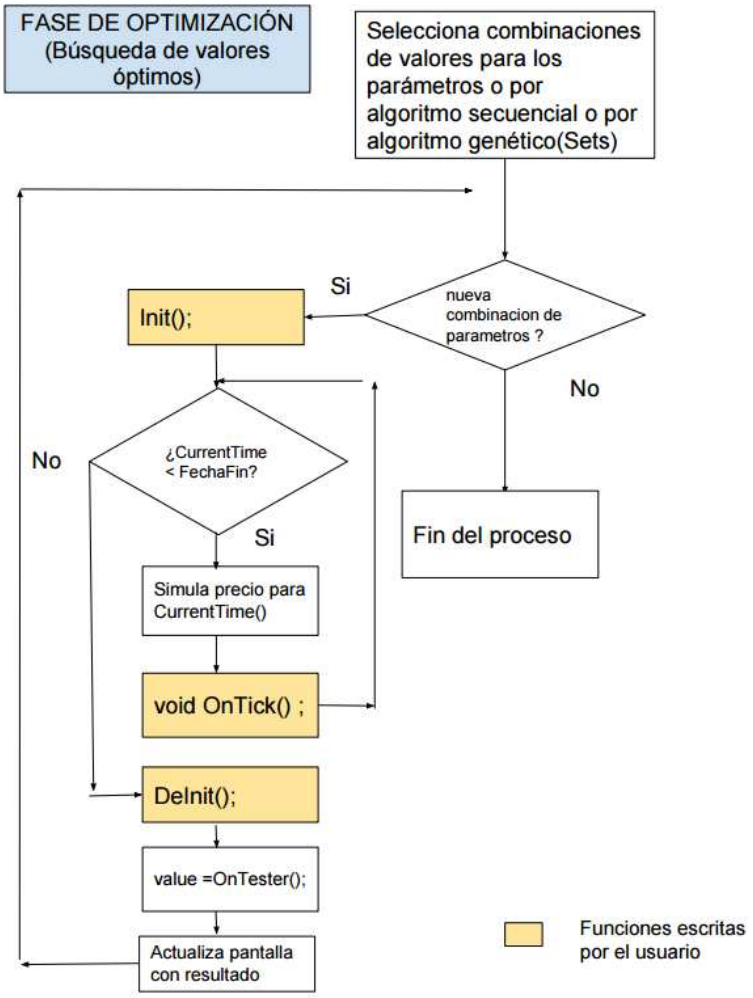


Figura 3.5. Diagrama de funcionamiento en la fase de Optimización

Lo primero que tenemos que hacer es seleccionar que parámetros de los que tiene el sistema queremos optimizar, los parámetros son variables normales del programa que se han declarado con la directiva "extern", gracias a esto nos aparecerán en la ventana donde pueden ser seleccionados para ser optimizados, en el botón de propiedades del experto, ver figura 3.6.

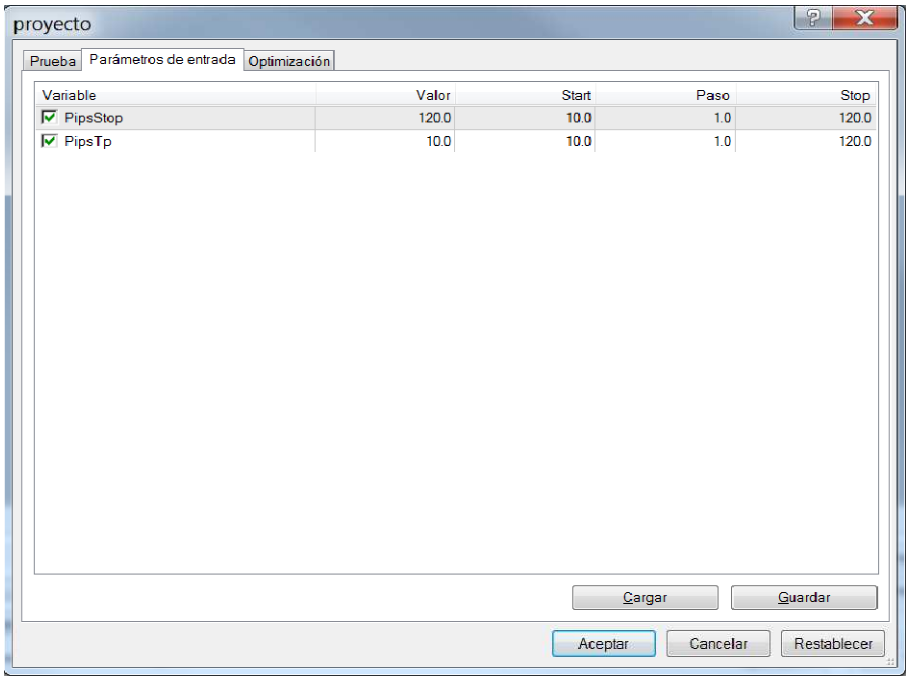


Figura 3.6. Ventana de propiedades del experto

Marcamos con la caja de chequeo las variables del sistema que queremos optimizar, en este caso "PipsStop" y "PipsTp" (Un Pip es la unidad mínima de ganancia en el mercado de divisas, en otros mercados sería un punto), en la figura 3.6 e indicamos que queremos probar desde un valor de inicio hasta un valor fin y en pasos de n elementos. A más rango de de valores a probar, mayores combinaciones tendrá que probar el optimizador y más tardará. En la figura 3.7 podemos ver la ventana con el proceso de optimización arrancado y el tiempo que tardará el proceso.

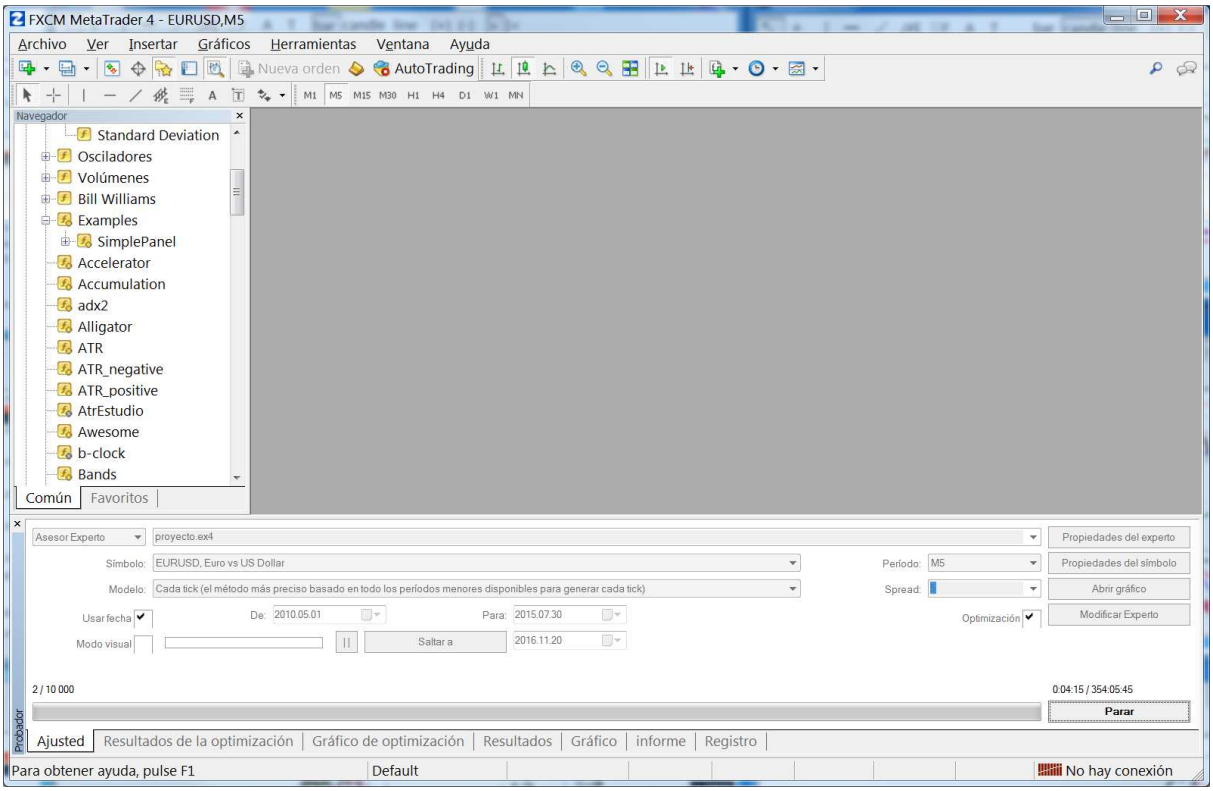
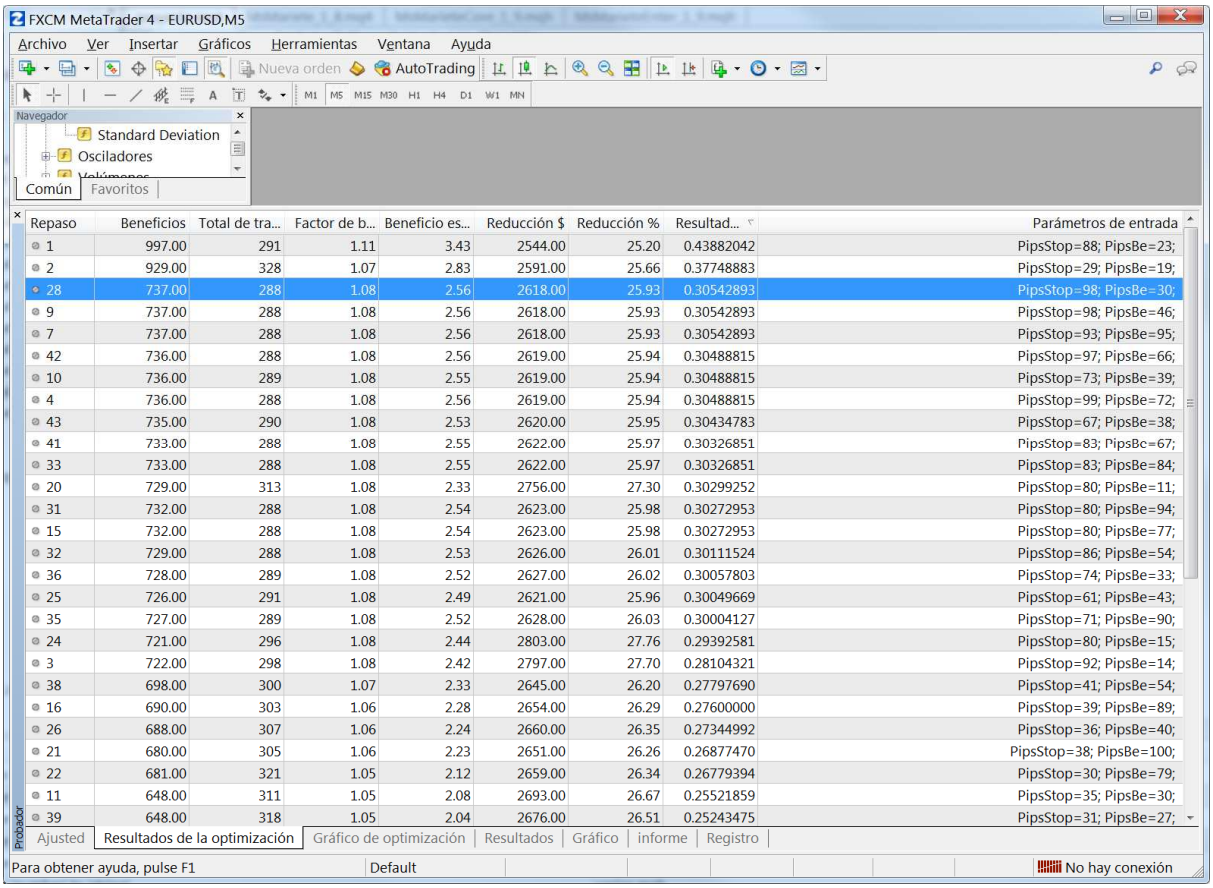


Figura 3.7. Ventana de Optimización

El optimizador le va dando valores a los parámetros de configuración del robot y éste va ejecutándose con los datos del histórico simulando las operaciones, para cada combinación de parámetros que el optimizador está probando (también llamado un "set" de parámetros) el robot simula todas las operaciones como si hubieran sido reales y al final de la simulación calcula el valor de la función de optimización para dicho set de parámetros, esto se repite para todas las combinaciones de parámetros, generándose así una tabla con los distintos valores de la función de optimización, además de otros valores que siempre nos interesan como son el beneficio neto conseguido, la pérdida máxima, el factor de beneficio, el número de operaciones realizadas y la función de optimización personalizada, donde el programador puede crear una propia función de optimización que puede combinar más de un objetivo, normalmente será bi-objetivo: maximizar beneficio y minimizar riesgo. Al estudiar dicha tabla podríamos ordenarla por la columna que nos interese, lo típico es ordenar por beneficio neto conseguido que usualmente es el objetivo buscado. Al ordenarla nos pondrá en la primera fila la combinación de parámetros que maximiza el beneficio o la función objetivo que hayamos creado.



Repaso	Beneficios	Total de tra...	Factor de b...	Beneficio es...	Reducción $	Reducción %	Resultad...	Parámetros de entrada
1	997.00	291	1.11	3.43	2544.00	25.20	0.43882042	PipsStop=88; PipsBe=23;
2	929.00	328	1.07	2.83	2591.00	25.66	0.37748883	PipsStop=29; PipsBe=19;
28	737.00	288	1.08	2.56	2618.00	25.93	0.30542893	PipsStop=98; PipsBe=30;
9	737.00	288	1.08	2.56	2618.00	25.93	0.30542893	PipsStop=98; PipsBe=46;
7	737.00	288	1.08	2.56	2618.00	25.93	0.30542893	PipsStop=93; PipsBe=95;
42	736.00	288	1.08	2.56	2619.00	25.94	0.30488815	PipsStop=97; PipsBe=66;
10	736.00	289	1.08	2.55	2619.00	25.94	0.30488815	PipsStop=73; PipsBe=39;
4	736.00	288	1.08	2.56	2619.00	25.94	0.30488815	PipsStop=99; PipsBe=72;
43	735.00	290	1.08	2.53	2620.00	25.95	0.30434783	PipsStop=67; PipsBe=38;
41	733.00	288	1.08	2.55	2622.00	25.97	0.30326851	PipsStop=83; PipsBc=67;
33	733.00	288	1.08	2.55	2622.00	25.97	0.30326851	PipsStop=83; PipsBe=84;
20	729.00	313	1.08	2.33	2756.00	27.30	0.30299252	PipsStop=80; PipsBe=11;
31	732.00	288	1.08	2.54	2623.00	25.98	0.30272953	PipsStop=80; PipsBe=94;
15	732.00	288	1.08	2.54	2623.00	25.98	0.30272953	PipsStop=80; PipsBe=77;
32	729.00	288	1.08	2.53	2626.00	26.01	0.30111524	PipsStop=86; PipsBe=54;
36	728.00	289	1.08	2.52	2627.00	26.02	0.30057803	PipsStop=74; PipsBe=33;
25	726.00	291	1.08	2.49	2621.00	25.96	0.30049669	PipsStop=61; PipsBe=43;
35	727.00	289	1.08	2.52	2628.00	26.03	0.30004127	PipsStop=71; PipsBe=90;
24	721.00	296	1.08	2.44	2803.00	27.76	0.29392581	PipsStop=80; PipsBe=15;
3	722.00	298	1.08	2.42	2797.00	27.70	0.28104321	PipsStop=92; PipsBe=14;
38	698.00	300	1.07	2.33	2645.00	26.20	0.27797690	PipsStop=41; PipsBe=54;
16	690.00	303	1.06	2.28	2654.00	26.29	0.27600000	PipsStop=39; PipsBe=89;
26	688.00	307	1.06	2.24	2660.00	26.35	0.27344992	PipsStop=36; PipsBe=40;
21	680.00	305	1.06	2.23	2651.00	26.26	0.26877470	PipsStop=38; PipsBe=100;
22	681.00	321	1.05	2.12	2659.00	26.34	0.26779394	PipsStop=30; PipsBe=79;
11	648.00	311	1.05	2.08	2693.00	26.67	0.25521859	PipsStop=35; PipsBe=30;
39	648.00	318	1.05	2.04	2676.00	26.51	0.25243475	PipsStop=31; PipsBe=27;

Figura 3.8. Ventana de resultados de la optimización

En el párrafo anterior hemos comentado que el algoritmo recorre todas las combinaciones de parámetros y simula el sistema, calculando la función objetivo en cada pasada para ver que set de parámetros maximiza la función objetivo. El problema que ocurre es que normalmente el número de combinaciones de parámetros a probar puede ser del orden de miles o millones de combinaciones, para abordar este problema, Metatrader tiene 2 posibles estrategias: usar un algoritmo exhaustivo para la búsqueda de soluciones o usar un algoritmo genético. La estrategia a utilizar en la fase de optimización la seleccionamos en las opciones del sistema o Expert Advisor (Experto en la traducción a español del interfaz), ver figura 3.9.

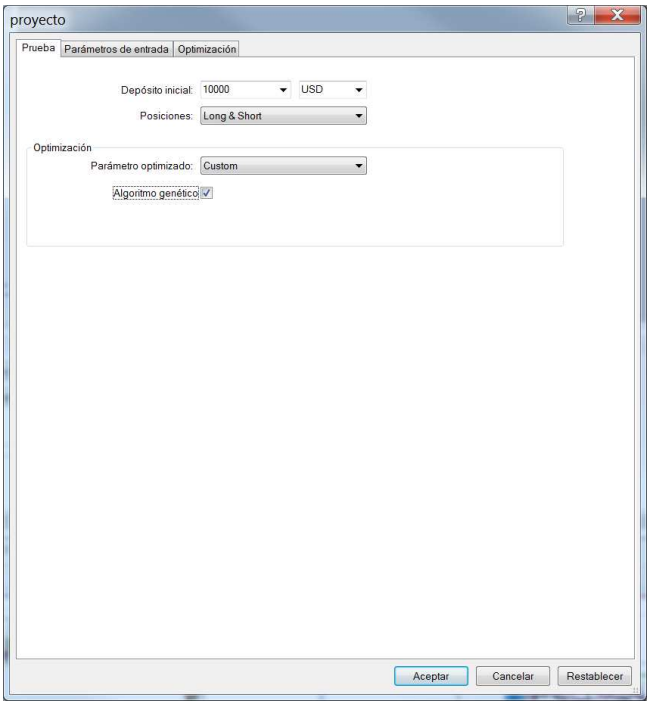


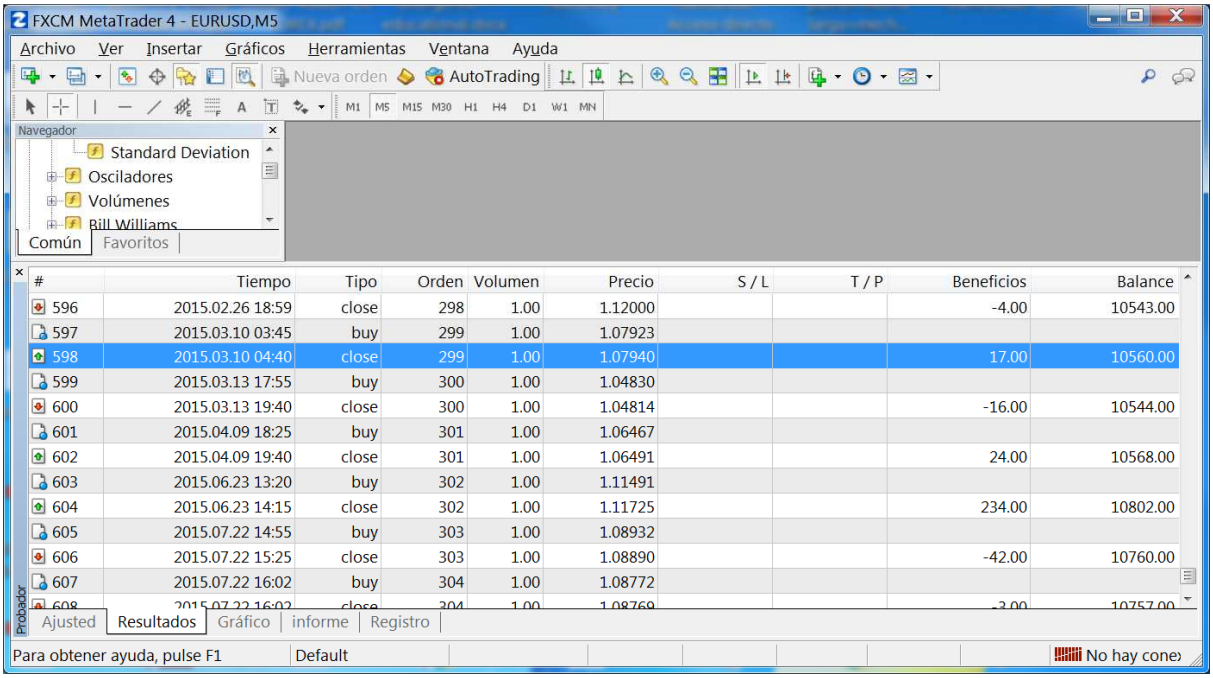
Figura 3.9. Selección de la estrategia de optimización

3.2.3 Fase de Lanzamiento en modo Prueba (Test Mode)

Esta fase de prueba (Tester Mode), también conocida en trading como backtesting, consiste en para una configuración concreta de parámetros, probar el robot con los datos históricos dándonos como resultado un informe completo con numerosa información estadística que nos será muy útil para comparar rendimiento de robots etc. El valor de los parámetros utilizado, normalmente será el que hemos seleccionado en la fase de optimización como el más óptimo para nuestra función objetivo.

Este modo de Tester o prueba tiene una modalidad que puede ser útil que es el "modo visual", esto lo que hace es simular visualmente en el gráfico la operativa del robot con los datos históricos como si fuera en tiempo real, como es natural si hemos establecido un periodo histórico de 5 ó 10 años no vamos a estar ese tiempo contemplando en el gráfico cómo se comporta el robot, para esto tenemos un control que nos permite ajustar la velocidad de esta simulación así podemos multiplicar x4 ó por x16 etcétera la velocidad de la simulación como si fuera la velocidad de una película, con lo que podemos simular todo a gran velocidad y bajar la velocidad en alguna operación en la que queramos concentrarnos para estudiarla a fondo.

Normalmente cuando ya conocemos el robot y sabemos que se comporta bien, es decir cumple perfectamente las reglas del programa y ya está correctamente depurado, no necesitaremos ver visualmente cada operación, el Trader se suelen centrar en el informe que devuelve este modo prueba. El informe proporcionado, consta de 3 partes: Tabla con las operaciones (figura 3.10), gráfico con el balance (figura 3.11) e Informe con los indicadores estadísticos útiles (figura3.12)



#	Tiempo	Tipo	Orden	Volumen	Precio	S / L	T / P	Beneficios	Balance
596	2015.02.26 18:59	close	298	1.00	1.12000			-4.00	10543.00
597	2015.03.10 03:45	buy	299	1.00	1.07923				
598	2015.03.10 04:40	close	299	1.00	1.07940			17.00	10560.00
599	2015.03.13 17:55	buy	300	1.00	1.04830				
600	2015.03.13 19:40	close	300	1.00	1.04814			-16.00	10544.00
601	2015.04.09 18:25	buy	301	1.00	1.06467				
602	2015.04.09 19:40	close	301	1.00	1.06491			24.00	10568.00
603	2015.06.23 13:20	buy	302	1.00	1.11491				
604	2015.06.23 14:15	close	302	1.00	1.11725			234.00	10802.00
605	2015.07.22 14:55	buy	303	1.00	1.08932				
606	2015.07.22 15:25	close	303	1.00	1.08890			-42.00	10760.00
607	2015.07.22 16:02	buy	304	1.00	1.08772				
608	2015.07.22 16:02	close	304	1.00	1.08769			-3.00	10757.00

Figura 3.10. Tabla de resultados del modo Tester.

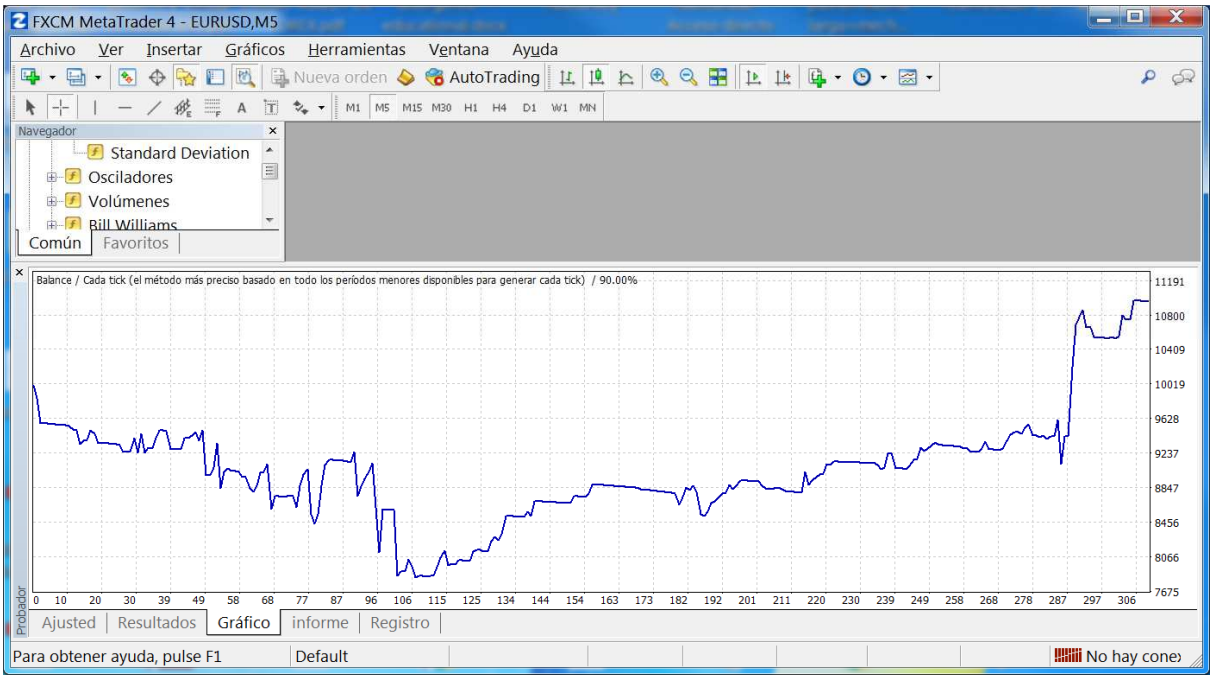


Figura 3.11. Gráfico con el Balance en el modo Tester

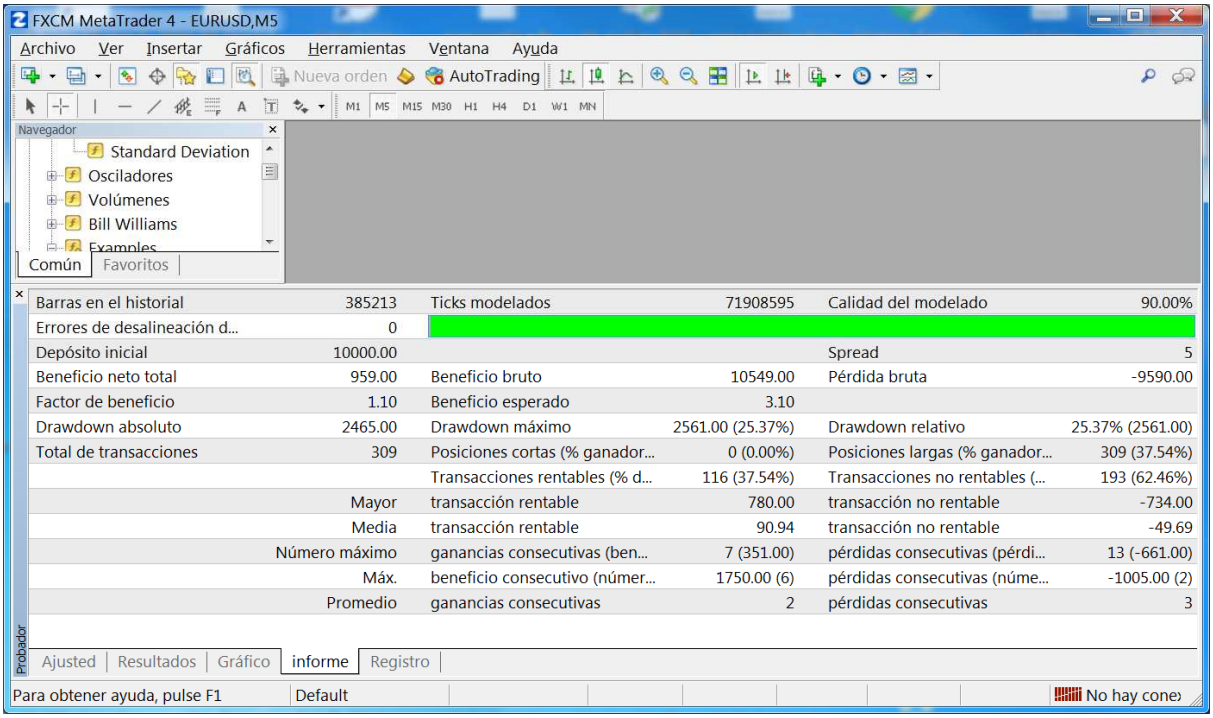


Figura 3.12. Informe del modo Tester

En la figura 3.13 podemos ver el diagrama de funcionamiento de Metatrader ejecutando un robot en modo Prueba

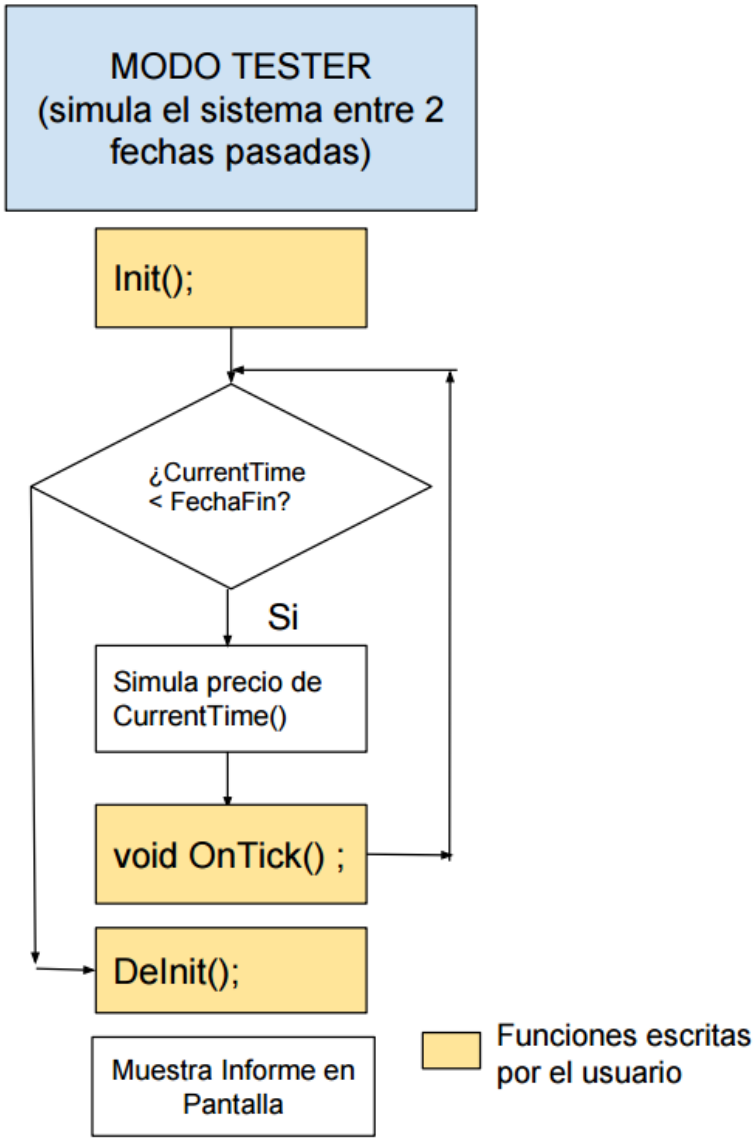


Figura 3.13. Diagrama de funcionamiento de Metatrader en modo Tester

3.2.4 Fase de prueba en tiempo real (Forward Test)

Si la fase de prueba ha sido satisfactoria, o sea el sistema nos da un beneficio neto positivo, tiene un riesgo aceptable (normalmente DrawDown < 15 %) y su gráfica de crecimiento es bastante lineal, es decir sin grande sobresaltos, lo ideal es que forme un

ángulo de 45 grados más o menos, entonces podemos pensar que el sistema podría funcionar bien y tenemos que probarlo con el mercado real, es decir en Forward Test.

Esta fase consiste en probar el sistema durante un tiempo suficientemente grande en un mercado real, para ver que la estadística del sistema se sigue cumpliendo. Muchos Traders u operadores de mercado suelen hacer esta prueba usando cuentas de demostración que los bróker suelen regalar por un tiempo, es decir son cuentas donde se opera con dinero virtual (normalmente 10.000$) pero los datos del mercado son reales se les suele denominar cuentas "demo". Pero a veces estas cuentas demo no ejecutan las ordenes exactamente igual que una cuenta real, ya que los bróker las suelen tener en servidores de internet situados en otra localización física o con otras características hardware etc.

Por esto lo más recomendable es hacer el forward test en una cuenta real con dinero real, eso sí, operando con el menor tamaño de operación posible, por ejemplo en el mercado de divisas (conocido como Forex) se puede hacer una operación trabajando con distintos tamaños de contrato. Un ejemplo: podemos comprar el euro frente al dólar americano (EURUSD) con un cambio actual de 1,1000 pensando que va a subir 10 puntos (el punto es el 4 decimal de la cotización, como hemos comentado antes en divisas se le conoce como pip), si acertamos y nuestra operación es positiva, la cotización del EURUSD tocará 1,1010 y habremos ganado 10 puntos. Pero a una operación podemos entrar con diferentes pesos de contrato, así podemos trabajar con un multiplicador de 0,01 (microcontratos) con esto cada punto del EURUSD valdrá 0,1$ es decir 10 céntimos de $, otro posible multiplicador es 0,1 (minicontrato) donde cada punto valdrá 1$ y por último podemos operar con un multiplicador de 1 (contrato estándar) donde cada punto valdrá 10$. Así en nuestra operación de ejemplo donde hemos ganado 10 puntos, si operamos con microcontratos habríamos ganado 1$ (10 puntos x 0,1$ el punto), si hubiéramos operado con minicontratos habríamos ganado 10$ y si hubiéramos operado con contratos estándar habríamos ganado 100$, esto se aplica igualmente si nuestra operación es perdedora, la cantidad que perdemos va en función del tipo de contrato con el que estamos trabajando.

Cuando estamos en la fase de prueba del robot o Forward Test, lo ideal es utilizar microlotes (0,01$ por cada punto) así si el robot no cumple las expectativas que ha generado en el backtesting no incurriremos en grandes pérdidas. De esta manera, por ejemplo podemos hacer 100 operaciones, y en cada una podemos perder un máximo de 40 puntos por

operación, lo cual es normal en un sistema de trading, aunque perdamos las 100 (situación poco probable si los resultados del backtesting han sido óptimos), serían 100 operaciones x 40 puntos x 0,01$/punto = 40 euros.

El tiempo de prueba que ha de estar el robot, no tiene una regla fija, ya que lo primero no es lo mismo operar robots cuyo marco temporal son periodos de 5 minutos (como es el que veremos más adelante en un ejemplo real) o si su marco temporal son periodos de 1 día. Si son 5 minutos quiere decir que en un día tendremos 288 periodos o ventanas de 5 minutos, con lo cual tendremos bastantes probabilidades de que se cumplan nuestras reglas del sistema y ocurra alguna operación. Si nuestro marco temporal es 1 día, necesitaremos mucho más tiempo para que ocurran oportunidades que cumplan nuestras reglas así necesitaremos mucho más tiempo de forward test. Lo interesante quizá es hablar de número de operaciones que ha realizado el sistema más que de tiempo funcionando y ese número de operaciones debe ser lo suficientemente grande para que la estadística sea fiable (de forma empírica se puede decir que nunca menos de 100 operaciones).

Una vez que el sistema cumple nuestras expectativas en forward test en una cuenta real, ya se puede pasar a modo operativo real. Aumentando el peso del contrato a nuestro riesgo real, es decir si lo estábamos probando con microcontratos (0,01$ por punto) podemos ponerlo usando minicontratos (0,1$ por punto). Si le ponemos minicontratos o contratos estándar y cuantos ponemos depende de nuestro capital para la inversión, y casi todos los expertos coinciden en que nunca se debe arriesgar más de 0,5% - 2% de nuestro capital para inversión en una operación, así para un capital de 10.000$ (ó euros) suele ser normal arriesgar un 0,5% o sea 50$ para esto se suelen usar minicontratos donde el punto es 1$, así en una operación como mucho podemos tener 50 puntos en contra ya que 50 puntos x 1$ /punto = 50$, así si nuestro capital digamos que es de 20.000$ utilizaremos 2 minicontratos (1$/punto).

Un punto importante y que es materia de investigación es cuando consideramos que el robot ya no es útil y debemos apagarlo. Ya que tenemos el robot funcionando en tiempo real y con dinero real y se supone que proporcionándonos beneficio, habría que monitorizarlo para saber si en algún momento deja de funcionar correctamente, ya que los mercados son cambiantes y dependen de noticias, decisiones de bancos centrales, decisiones

políticas, guerras etc. Puede ser que pasados algunos meses o años el robot tenga cada vez rachas de pérdidas mayores o incluso deje de ganar, esto tendría como consecuencia que tengamos que volver a optimizarlo o puede que incluso apagarlo si la ventaja estadística que en su día tuvo ha desaparecido.

En la figura 3.14 puede verse el diagrama de funcionamiento de un robot en modo tiempo real

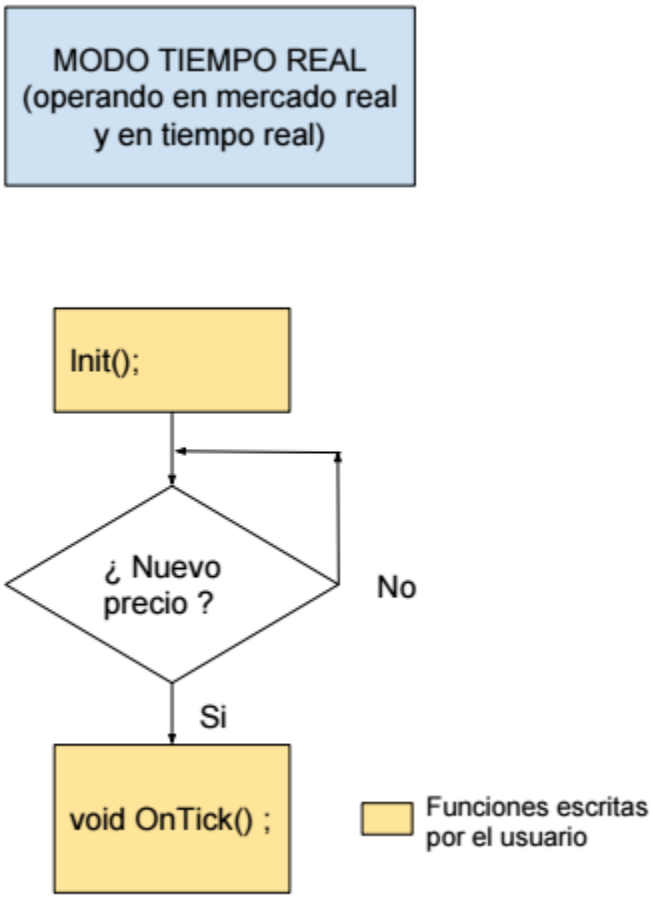


Figura 3.14. Diagrama de funcionamiento de Metatrader en modo Tiempo real

3.2 Descripción de los posibles algoritmos de la fase de optimización

Como hemos visto en el capítulo anterior, la fase de optimización requiere un gran esfuerzo de computación. Vamos a estudiar las estrategias que sigue Metatrader para abordar este problema.

3.2.1 Algoritmo de búsqueda exhaustiva

El algoritmo secuencial, se trata simplemente de ir generando todas las combinaciones posibles que pueden tener los parámetros del sistema (o variables de entrada) para buscar el máximo en la función de optimización. Como principal ventaja es que el algoritmo secuencial nos va a garantizar que vamos a encontrar la mejor solución posible, es decir que vamos a encontrar el conjunto de valores que maximizan la función de optimización. Otra ventaja es que sabemos exactamente los casos que quedan por probar ya que conocemos todas las combinaciones de variables a probar.

El principal inconveniente del algoritmo exhaustivo para la búsqueda del óptimo es que requiere mucho tiempo de computación. Se trata de un buen método cuando el espacio de soluciones posibles es reducido pero siendo a veces imposible de usar cuando el espacio de soluciones es muy grande ya que la solución puede tardar años de computación.

3.2.2 Algoritmo genético

Como hemos visto en el apartado anterior, recorrer todo el espacio de soluciones para optimizar la función objetivo es demasiado costoso, necesitaríamos una técnica que sin probar todos los casos busque el mejor objetivo o al menos alguno aceptable, aquí es donde surgen los algoritmos genéticos.

Los algoritmos genéticos se aplican en tareas de optimización. Por ejemplo en el aprendizaje de redes neuronales, esto es, para seleccionar los valores de peso que permiten obtener el mínimo error. El algoritmo genético se basa en el método de búsqueda aleatoria. El principal problema de la búsqueda aleatoria es que no podemos saber cuánto tiempo se tarda en resolver el problema.

El nombre de algoritmos genéticos viene del que se conoce como padre de los mismos que fue John Holland, investigador de la universidad de Michigan, tras la publicación en 1975 de su libro "Adaptation in Natural and Artificial Systems" [HOL75]. La idea es bastante intuitiva, durante millones de años las diferentes especies se han adaptado para poder sobrevivir en un entorno cambiante, si pensamos que las especies podrían ser las distintas soluciones al problema que es la adaptación al medio, donde la naturaleza va seleccionando las mejores especies hasta que se adapten perfectamente a dicho medio, o problema a resolver. En términos muy generales se podría definir la computación evolutiva como una familia de modelos computacionales inspirados en la evolución [GES10]

La computación evolutiva, donde se incluyen los algoritmos genéticos, tiene su base en la copia de los procesos que ocurren en la selección natural, este último concepto fue introducido, por Charles Darwin [DAR59]. Aunque no se conocen todos los detalles exactos, hay varios hechos probados en experimentos sobre la selección natural:

- La evolución opera sobre los cromosomas más que sobre los organismos.

- La selección natural es el mecanismo que relaciona los cromosomas (genotipo) con la eficacia en la adaptación al entorno del individuo (fenotipo) que representan. Es decir proporciona a individuos más adaptados mayor probabilidad de reproducción.

- Los procesos evolutivos tienen lugar durante la etapa de reproducción, aunque hay muchos mecanismos que afectan a la reproducción, los más comunes son: **la mutación** causantes de que los cromosomas de los descendientes sean diferentes a los de los padres y el **cruce o recombinación,** que combina los cromosomas de los padres para producir la decendencia.

Pero la adaptación de un individuo al medio no sólo está condicionada por su composición genética. Influyen factores que se adquieren como el aprendizaje etc, aspectos no contemplados en los algoritmos genéticos clásicos pero si en otras variantes.

Los algoritmos genéticos son parte de esa computación evolutiva. A grandes rasgos un Algoritmo genético consiste en codificar las soluciones como si fueran cromosomas, cada uno de estos cromosomas tendrá un valor asociado o fitness, que cuantifica la validez de esa solución al problema. Según dicho valor (o lo buena que sea la solución al problema), tendrá más o menos oportunidades de reproducción. Además de esto, se realizarán con cierta probabilidad mutaciones en los cromosomas. Este proceso hará que los individuos genéticos tiendan a dar soluciones al problema dado. Un parámetro de nuestro sistema de trading es lo que se llama gen en la terminología de algoritmo genético, el conjunto de parámetros, se codifica en una cadena de valores denominada cromosoma (esto sería lo que llamamos en el sistema un set de parámetros del mismo) o una posible solución del problema de optimización. Ver figura. 3.15

Figura 3.15. Cadena de valores o Cromosoma.

En la figura tendríamos un cromosoma compuesto por 5 genes, donde se han codificado los parámetros (genes) en binario, pero hay implementaciones que codifican los parámetros como números enteros o flotantes. En el caso de Metatrader no es conocido debido a que no ha sido publicado.

Como hemos comentado antes, durante la selección natural que hace la naturaleza se producen lo que podríamos llamar procesos u operaciones sobre los cromosomas que los hacen diferentes de los de sus progenitores. Así en los algoritmos genéticos también habrá distintos operadores sobre los cromosomas o posibles soluciones:

Operador de cruce: Este operador hace que cambie la programación de un cromosoma o cromosomas de una generación a la siguiente.

- Seleccionar padre y madre de una población

- Determinar el punto de ruptura dentro del cromosoma (normalmente aleatorio)
- Se concatenan las 2 partes para formar el cromosoma de la descendencia

Posible representación:

Cromosoma_1: 0000000000
Cromosoma_2: 1111111111

Suponemos que el punto de ruptura ocurre después del 3er bit del cromosoma, por lo que:

Cromosoma_1:0000000000>> 000 1111111 Cromosoma_resultante_1
Cromosoma_2:1111111111>> 111 0000000 Cromosoma_resultante_2

Entonces uno de los cromosomas resultantes queda determinado como descendiente con una probabilidad de 0.5.

Operador mutación: Este operador sirve para mantener la diversidad de la población. Este operador altera el valor inicial de uno o más genes de un cromosoma. Por lo tanto, cada bit del cromosoma se invierte con una probabilidad determinada.

La representación quedaría de la siguiente forma:

0000000000 => 0000100000

Operador inversión: Este operador divide el cromosoma en dos partes y cambia las posiciones.

La representación quedaría de la siguiente forma:

000 1111111 >> 1111111 000

Aunque podríamos decir que estos son los operadores básicos se podrían modificar y hacer operadores adicionales.

Algunas implementaciones del algoritmo aplican la llamada **estrategia elitista**, que significa que las unidades más adaptadas tienen garantizado incorporarse a la nueva población. Generalmente, esta aproximación permite acelerar la convergencia del algoritmo genético. El inconveniente de esta estrategia es que aumenta la probabilidad de que el algoritmo se centre en un mínimo local.

Pseudocódigo genérico del algoritmo genético propuesto inicialmente por Holland

```
Inicializar población actual aleatoriamente
MIENTRAS no se cumpla el criterio de terminación
        Crear población temporal vacía.
    SI elitismo: copiar en población temporal mejores individuos
    MIENTRAS población temporal no llena
    Seleccionar padres
    Cruzar padres con probabilidad Pc
    SI se ha producido el cruce
        Mutar uno de los descendientes (Prob. Pm)
        Evaluar descendendientes
        Añadir descendientes a la población temporal
    SINO
        Añadir padres a la población temporal
    FIN SI
    FIN MIENTRAS
    Aumentar contador de generaciones
    Establecer como nueva población la temporal
FIN MIENTRAS
```

Sobre esta base de algoritmo genético se han hecho múltiples variantes.

Los criterios de parada de los algoritmos genéticos suelen ser:

- Se establece un número de generaciones máximo prefijado
- La población ha convergido. Ocurre cuando el 95% de la población tiene el mismo valor para un gen (caso de estar trabajando con codificación binaria) o está en un rango (caso de codificación flotante etc)[7]

4. Ejemplo de Sistema automático en metatrader: "Orden perfecto"

En este capítulo vamos a estudiar un ejemplo real de un sistema de trading automático y vamos a optimizarlo utilizando la herramienta Metatrader 4, así vamos a observar claramente los requerimientos computacionales que ello requiere.

4.1. Descripción del sistema "Orden perfecto".

La estrategia de trading "Orden Perfecto" aparece descrito en libro "Day trading & Swing trading the Currency Market" de la prestigiosa analista y trader norteamericana Kathy Lien [LIE08]. En dicho libro podemos encontrar numerosas estrategias de trading que pueden ser implementadas como sistemas de trading automático. Hay mucha bibliografía sobre estrategias de trading que pueden ser codificadas como robots [SIN13]

Es una estrategia que como dice el título del libro está pensada para ser operada en el mercado de divisas. Se trata de capturar una tendencia en el precio, es decir un intervalo

de tiempo más o menos grande donde el precio va en una dirección clara. (ver figura 4.1 ej. De tendencia alcista)

Figura 4.1. Cadena de valores o Cromosoma.

Podemos ver que según la definición de tendencia [MUR99], buscaremos en el precio mínimos crecientes y máximos crecientes en el caso de una tendencia alcista y al revés mínimos decrecientes y máximos decrecientes en caso de una bajista. Pero esta definición es un poco subjetiva para un software, así que los programadores de sistemas suelen usar medias móviles (una media aritmética simple o exponencial) para detectar la entrada en una tendencia o la salida de la misma. En esta estrategia Kathy LIen nos propone utilizar 5 medias móviles simples de 10, 20, 50, 100 y 200 periodos.

Otro indicador que se usa para ver la fuerza de una tendencia es el ADX (Average Directional Index), este indicador desarrollado por Welles Wilder [WIL78], nos va a dar en forma de un valor entre 0 y 100 la fuerza de una tendencia, normalmente se considera que si el indicador tiene un valor mayor que 20 ó 25 es que hay tendencia. Ver figura 4.2.

Figura 4.2. Indicador Adx sobre un gráfico del EURUSD

Por último le vamos a agregar otro indicador que son las Bandas de Bollinger [BOL02], las bandas de Bollinger es una función matemática que calcula la K veces la desviación estándar (habitualmente la K es 2) a partir de la media del precio, pintándose una línea por encima de dicha media (Banda superior) y otra por debajo de dicha media (Banda inferior), así cuando la cotización de un activo se sale o por arriba o por debajo de las bandas, decimos que tiene alta volatilidad ya que se ha salido de la "normalidad", en nuestro caso vamos a descartar ese precio y el sistema lo va a ignorar. Ver figura 4.3.

Figura 4.3. Indicador Bandas de Bollinger sobre un gráfico del EURUSD

La idea del sistema es entrar en el mercado, cuando todas las medias móviles están alineadas en la misma dirección y además el indicador de tendencia es >20.

Reglas del sistema:

- Regla de entrada: Entraremos al mercado cuando se cumpla que las medias están alineadas es decir se cumple que media(200)> media(100) > media(50) >media(20)>media(10), y la fuerza de la tendencia es importante, es decir Adx(20)>20 y Adx(100)>20 , y está dentro de las bandas de Bollinger (es decir el mercado tiene una volatilidad normal) en caso de cumplirse las 3 cosas entramos comprando un contrato.

- Regla de salida: El sistema permanecerá en la posición que haya conseguido unos puntos de beneficio, que será uno de los valores a optimizar. (En el mercado de divisas se le dice "Pip" a un punto de la cotización, normalmente el 4º decimal, así si el EURUSD pasa de 1,3020 a 1,3030 decimos que ha subido 10 pips)

- Regla de Stop de pérdidas: Fijaremos un nivel de precio por debajo del nivel de entrada que será nuestra máxima perdida aceptada, lo fijaremos en puntos (Pips) y será uno de los valores a optimizar.

Nota: este sistema solo hace compras, es decir solo funciona en un lado del mercado, pero se podría hacer con unas reglas de venta simétricas a las de compra que funcionara también haciendo ventas cuando las medias están alineadas al contrario que en el modo compra.

Una vez que tenemos unas reglas claras y sin ninguna subjetividad, entonces podemos codificarlas en Metatrader 4, usando el entorno de desarrollo que incluye como hemos comentado en la sección 2.3.1.

En la figura 4.4 podemos ver la función principal del robot, Ontick(), esta función es donde está el código que se ejecuta millones de veces, es ahí donde hay que centrarse en optimizar.

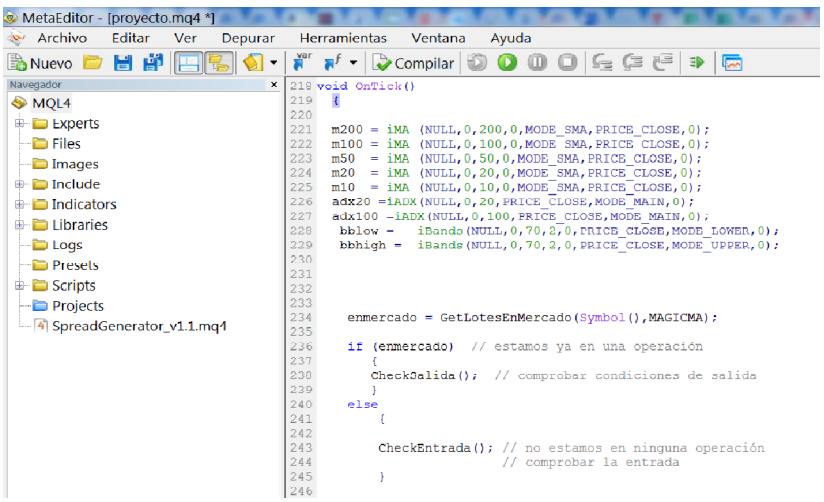
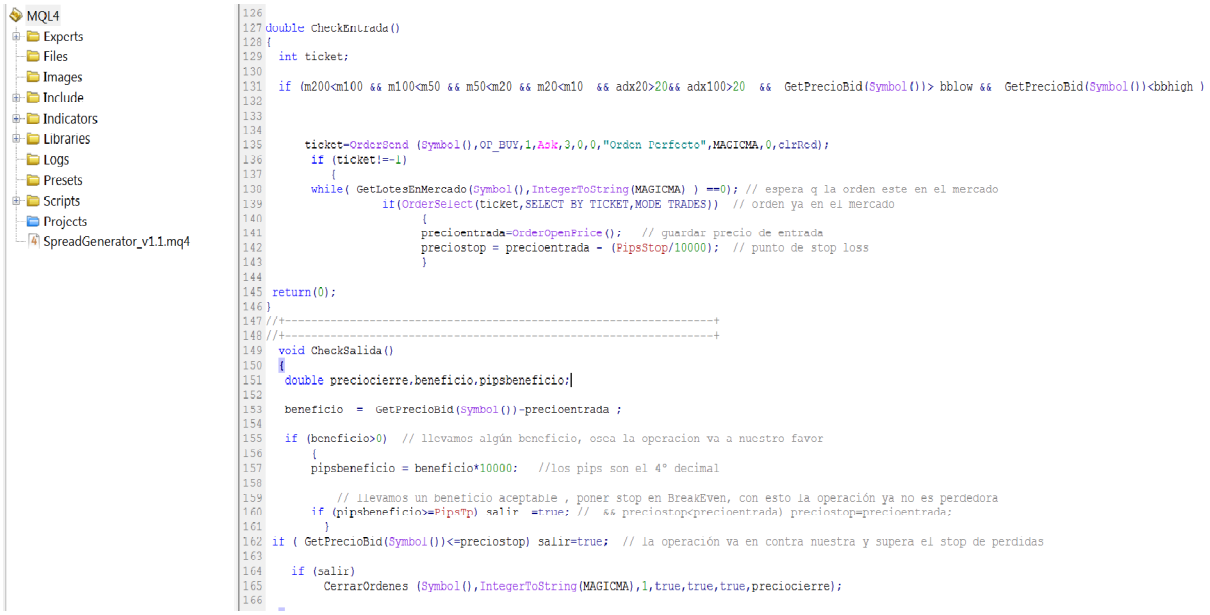


Figura 4.4. Función Ontick()

En la figura 4.5 podemos ver un detalle de las 2 funciones auxiliares utilizadas para gestionar la entrada al mercado y la salida del mismo.




```
126
127 double CheckEnEntrada()
128 {
129  int ticket;
130
131  if (m200<m100 && m100<m50 && m50<m20 && m20<m10  && adx20>20&& adx100>20  &&  GetPrecioBid(Symbol())> bblow &&  GetPrecioBid(Symbol())<bbhigh )
132
133
134
135      ticket=OrderSend (Symbol(),OP_BUY,1,Ask,3,0,0,"Orden Perfecto",MAGICMA,0,clrRed);
136       if (ticket!=-1)
137          {
138       while( GetLotesEnMercado(Symbol(),IntegerToString(MAGICMA) ) ==0); // espera q la orden este en el mercado
139                 if(OrderSelect(ticket,SELECT_BY_TICKET,MODE_TRADES))  // orden ya en el mercado
140                       {
141                       precioentrada=OrderOpenPrice();   // guardar precio de entrada
142                       preciostop = precioentrada - (PipsStop/10000);  // punto de stop loss
143                       }
144
145 return(0);
146 }
147 //+------------------------------------------------------------------+
148 //+------------------------------------------------------------------+
149  void CheckSalida()
150  {
151   double preciocierre,beneficio,pipsbeneficio;
152
153   beneficio  =  GetPrecioBid(Symbol())-precioentrada ;
154
155   if (beneficio>0)  // llevamos algún beneficio, osea la operacion va a nuestro favor
156       {
157       pipsbeneficio = beneficio*10000;   //los pips son el 4º decimal
158
159           // llevamos un beneficio aceptable , poner stop en BreakEven, con esto la operación ya no es perdedora
160       if (pipsbeneficio>=PipsTp) salir  =true; //  && preciostop<precioentrada) preciostop=precioentrada;
161         }
162 if ( GetPrecioBid(Symbol())<=preciostop) salir=true;  // la operación va en contra nuestra y supera el stop de perdidas
163
164    if (salir)
165       CerrarOrdenes (Symbol(),IntegerToString(MAGICMA),1,true,true,true,preciocierre);
166
```

Figura 4.5. Funciones auxiliares de Ontick()

4.2. Fase de Optimización del sistema "Orden perfecto"

Como hemos comentado en el anterior apartado, nuestro sistema de ejemplo va a tener 2 parámetros para ser optimizados, la primera variable será "PipsStop", es decir el número de puntos que aguantaremos una operación perdedora, lo ideal es que sea un número pequeño para que perdamos poco, pero si lo ponemos demasiado pequeño, aumentará la probabilidad de que la operación se cierre en negativo y si el número es demasiado alto, habrá pocas operaciones perdedoras pero las que haya serán de muy grandes pérdidas.

Por otra parte, la segunda variable a optimizar será "PipsTp", es decir será el número de puntos de beneficio conseguidos tras los cuales cerramos la operación en positivo, lo ideal es siempre que sean los mayores posibles, pero si el número es demasiado alto subirán las probabilidades de que el precio se de la vuelta y la operación acabe siendo perdedora, si es demasiado bajo, tendremos un sistema con alta probabilidad de acierto, pero cuando acierta, no ganará mucho dinero.

Nota: las siguiente optimización se va a hacer usando Metratrader4, sobre un equipo con Windows 7 profesional 64 bits, procesador i5-4460 con 4 cores de 3.2 Ghz, 8 Gbytes Ram y disco duro SSD 512 Gbytes. El activo seleccionado será el EURUSD en periodos de 5 minutos. Los datos históricos van desde 3-5-2010 a 30-07-2015 y los datos históricos han sido proporcionados por el bróker Alpari.

Establecemos el valor inicial, final y en que paso de cada variable. Ver figura 4.6

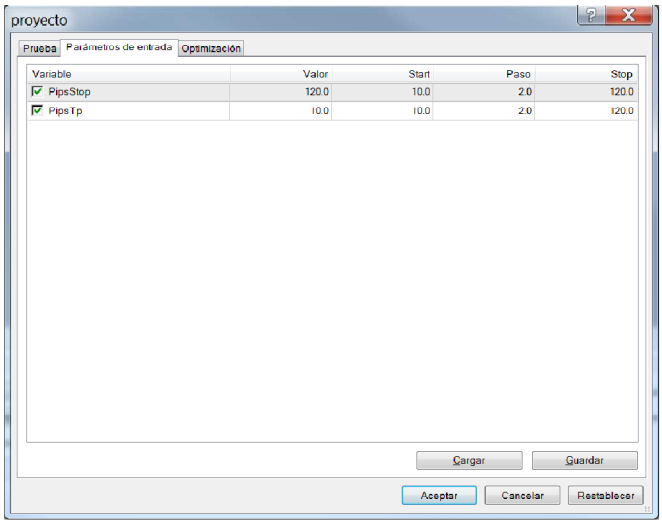


Figura 4.6. Parámetros de entrada a optimizar.

Si desmarcamos el algoritmo genético, entonces se hará una búsqueda exhaustiva, podemos ver en la figura que va a hacer 3136 pruebas, ya que hemos puesto 2 variables con 56 casos cada una hacen un total de 56 *56=3136 y nos dice que va a tardar 132 h , es decir aproximadamente 5 días y medio. Ver figura 4.7.

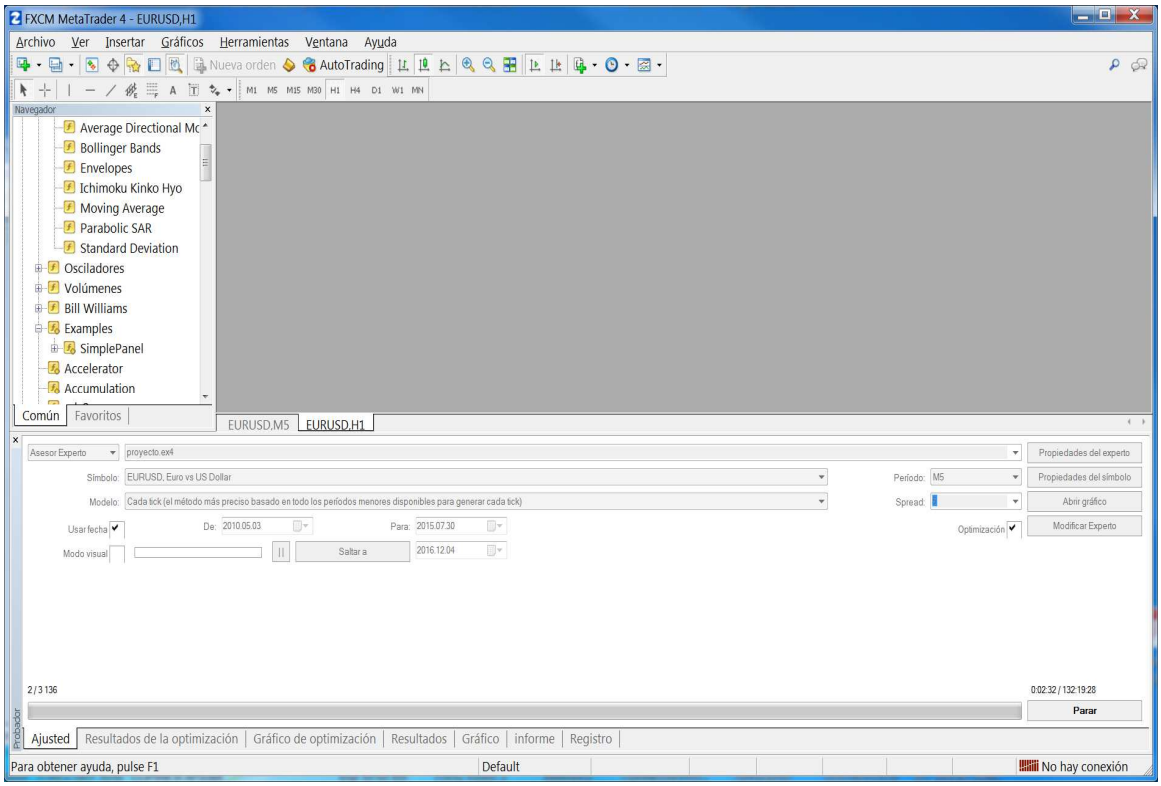


Figura 4.7. Optimización usando búsqueda exhaustiva

Si marcamos en las propiedades del experto que se utilice el algoritmo genético, vemos nada más iniciar nos dice que va a tardar 57 horas aproximadamente. Ver figura 4.8

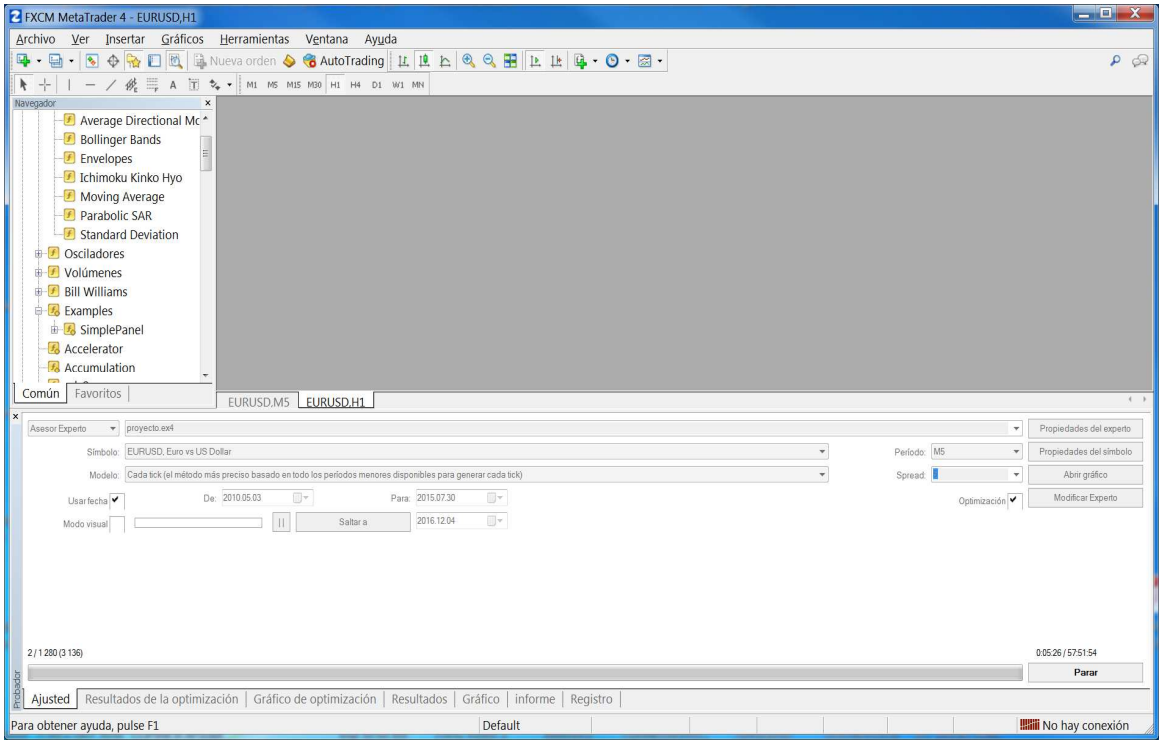


Figura 4.8. Optimización usando algoritmo genético

Pero si esperamos que termine, finalmente tarda 5h 47 minutos, ver figura 4.9. Como hemos dicho en la explicación del algoritmo genético, no son muy precisos en su estimación de tiempo que van a tardar, pero ha reducido el tiempo en 26 veces.

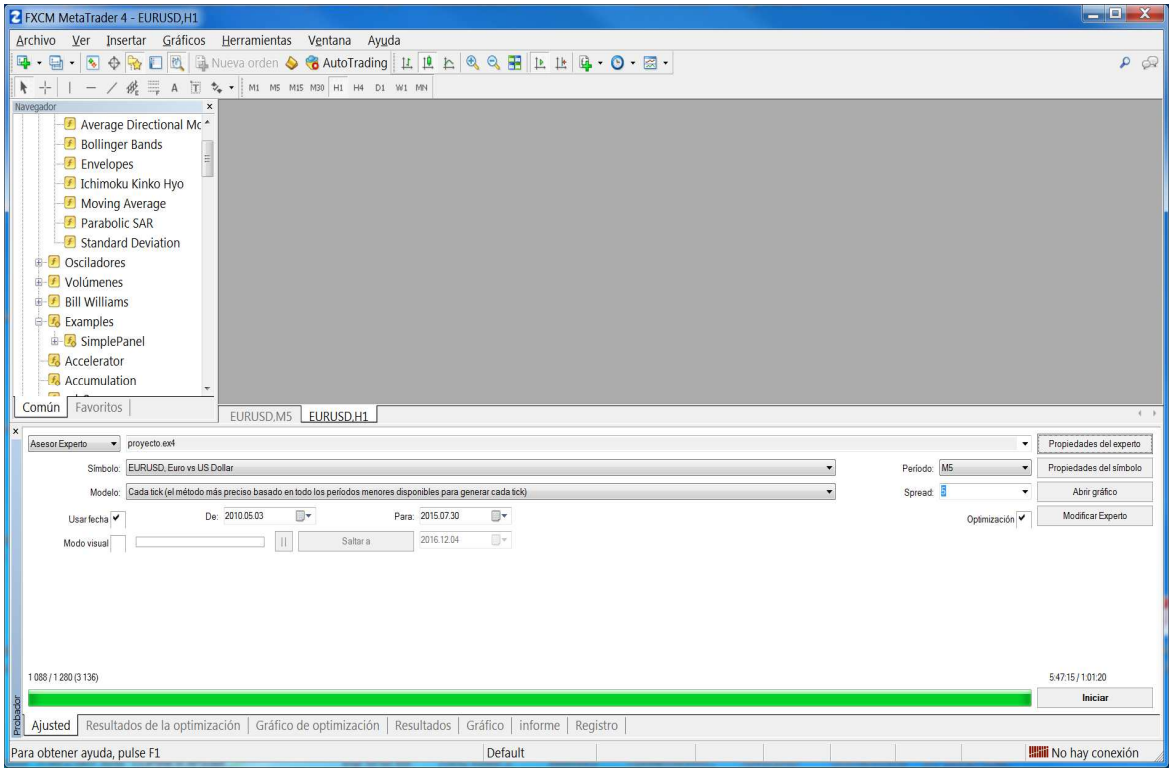


Figura 4.9. Optimización finalizada usando algoritmo genético

En la tabla con el resultado de la optimización podemos ver las combinaciones de parámetros y que beneficios hubieran conseguido, podemos observar que con 120 pips de stop y 40 pips (o puntos) de Tp (take profit o toma de beneficios) se consigue maximizar el beneficio en el periodo estudiado. Como hemos dicho antes no sólo hay que mirar el mayor beneficio, sino que interesa ver con que riesgo, esto se mira mirando la columna de reducción, o en su valor absoluto o en tanto por ciento. Ver figura 4.10.

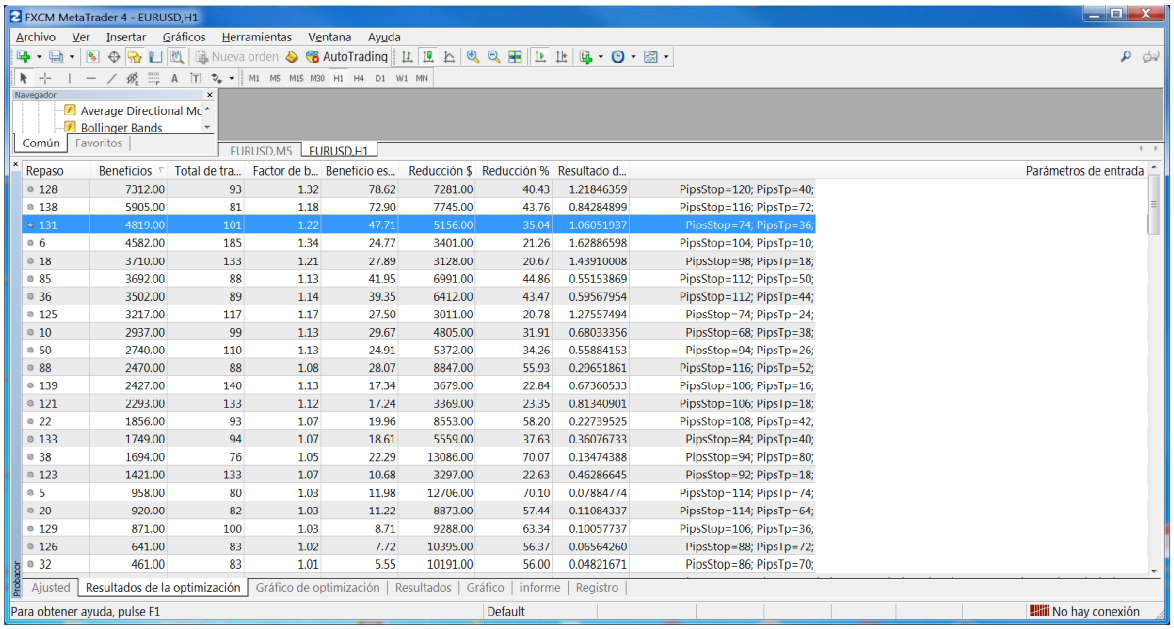


Figura 4.10. Resultado de la optimización

5. Estrategias para incrementar el rendimiento en la Fase de optimización

Ya hemos comprobado con un sistema real de trading, el alto requerimiento de computación que requiere la fase de optimización. En este capítulo vamos a explicar las mejoras conseguidas en el desarrollo de éste proyecto, implementadas en forma de librería que puede ser llamada desde cualquier sistema de trading en Metatrader 4, el nombre de la librería es "Mt4Parallel.Dll" y el objetivo es optimizar un sistema en menos tiempo.

Una característica de los sistemas de trading que puede ayudar a la optimización es que todos los cálculos se repiten millones de veces (hay gran redundancia de cálculos), así en una pasada sobre los datos históricos que tenemos para el sistema de prueba, desde el día 3/05/2010 hasta 30/07/2015, para el primer precio disponible en la fecha de origen va a calcular las 5 medias (de 10, 20, 50, 100, 200 periodos), el adx de 20 periodos y las bandas de Bollinger para 100 periodos, en cuanto haya un nuevo precio disponible para nuestro activo (EURUSD) se va a hacer de nuevo todos esos cálculos pero eso sí con el nuevo precio y así hasta llegar a la fecha final, según las medidas tomadas Metatrader nos da hasta 70 millones de cambios de precio en dicho periodo y eso que estamos en el modo Tester, esto luego se

multiplicará por el número pasadas de el optimizador buscando los parámetros óptimos. Luego parece ideal poder utilizar de alguna manera los cálculos de las pasadas anteriores para el cálculo de la pasada actual.

Estudiando el código de cualquier sistema de trading automático, tendremos una parte del sistema que ejecuta las **condiciones de entrada y de salida al mercado**, normalmente este código llama a funciones matemáticas y/o estadísticas (indicadores) ya implementadas en las librerías de Metatrader como son: las medias simples, medias exponenciales, Adx (average directional index), Bandas de Bollinger (Bollinger Bands), Atr (Average True Range) y así hasta 40 indicadores que vienen ya compilados en Metatrader además de otros de terceros que se pueden incorporar.

Otra parte del sistema son las funciones de **gestión de órdenes** en el mercado: enviar una orden de compra ó venta, comprobar si hay ordenes en el mercado, cerrar ordenes que haya en activo etc, estas funciones también son internas de metatrader.

Simulando el sistema de nuestro ejemplo en modo "Tester", es decir simulando el sistema para un periodo histórico de 5 años y con un conjunto de parámetros fijos: Pipsstop =120 y pipsTp = 10 , hemos comprobando que la fracción de tiempo dedicada al cálculo de las funciones matemáticas/estadísticas frente a las funciones de gestión de órdenes y el resto de lógica del sistema de trading del 70% del tiempo de cálculo así que nos centraremos en intentar optimizarlas. Ver figura 5.1.

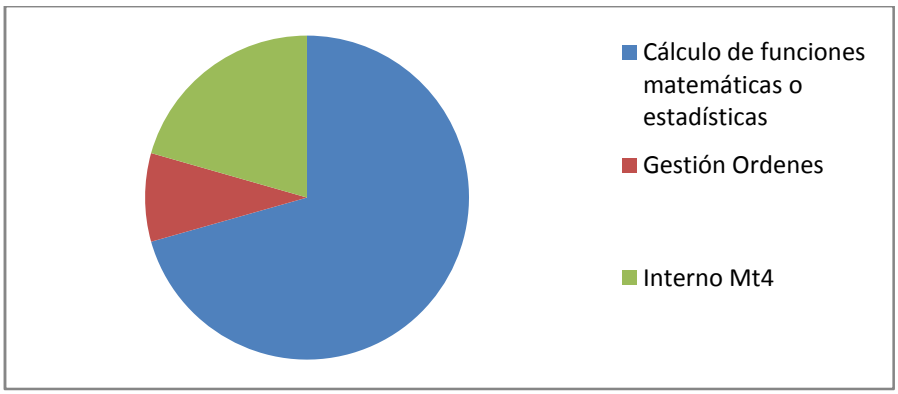


Figura 5.1. Distribución de tiempo en el modo tester de Metatrader

5.1. Optimizando el Secuencial

La primera técnica para mejorar el rendimiento de un algoritmo, es intentar optimizar el algoritmo secuencial existente.

El problema es que al estar implementado en Metatrader 4, no todo el código estará disponible para ser optimizado, ya que no tenemos acceso al código fuente de Metatrader 4. Vamos a intentar localizar las partes del código de un sistema de trading genérico en Metatrader que son susceptibles de ser mejoradas y en particular de nuestro sistema de ejemplo del capítulo anterior "Orden Perfecto".

En el sistema de ejemplo tenemos 5 llamadas a funciones que calculan una medias simples, de distintos periodos de tiempo, una llamada a la función que calcula el indicador Adx y dos llamadas a la función que calcula las Bandas de Bollinger. Ver código fuente en la figura 5.2

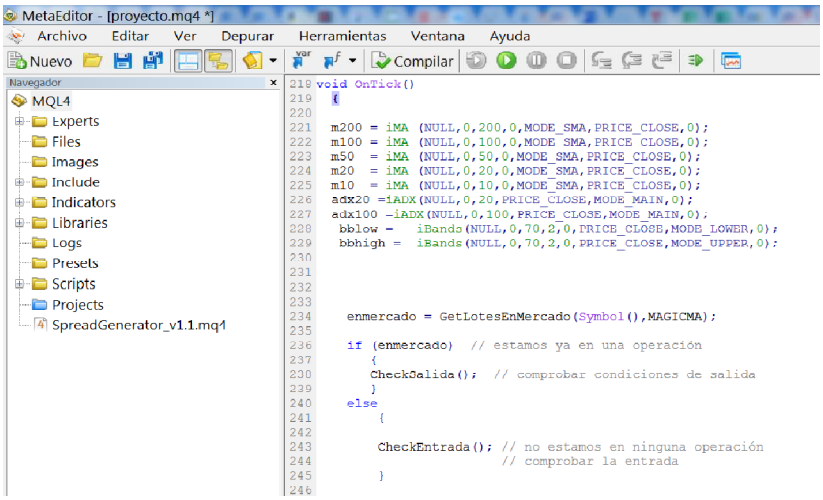


Figura 5.2. Código fuente original del sistema "orden perfecto" implementado en Metatrader.

El código fuente de los algoritmos de dichas funciones o indicadores está disponible en la web de Metatrader para que cualquiera lo pueda estudiar, lo que no podemos hacer es modificarlo y recompilar Metatrader ya que no tenemos acceso a su código fuente completo como hemos dicho al ser una aplicación comercial. La idea sería intentar desarrollar un algoritmo más eficiente incluirlo en una librería ya que Metatrader nos da la posibilidad de hacer llamadas a librerías externas en forma de DLL (Dynamic Link Library, son el sistema de librerías estándar en Windows), y asi reemplazaríamos en el

código de nuestros sistemas las llamadas a funciones menos eficientes en Metatrader por las nuevas más eficientes en la librería externa o DLL, siempre que podemos hacerlas más eficientes como es lógico.

Un ejemplo podría ser la función de la media simple, que no sólo es llamada en nuestro sistema 5 veces por cada cambio de precio (esto ocurre 70 millones de veces en los 5 años de histórico) sino que es un indicador que prácticamente usan el 80% de los sistemas de trading, o sea que la optimización de esta función conseguiría reducir el tiempo en nuestro sistema y en muchos otros.

Observando el código proporcionado por Metatrader nos damos cuenta que está haciendo un algoritmo de media clásico para el cálculo de la SMA (simple moving average):

```
sum = 0;
for(i=1;i<=MA_Period;i++)
        sum+=Close[i];
 sma =sum/MA_Period;
```

Otro algoritmo que podríamos utilizar para calcular la media simple y que además encaja perfectamente en la idea de aprovechar los cálculos anteriores sería el siguiente, media simple de 'n' periodos:

$$Sma(n)_i = Sma(n)_{i-1} + (X_i - X_{i-n}) * (1/n)$$

Es decir la función media la calculamos aprovechando la media de la anterior iteración y quitándole el primer elemento y añadiendo el nuevo elemento, así la hemos reducido al cálculo de una suma, una resta y una multiplicación por una constante (1/n se calculará al inicializar el sistema) independientemente del periodo o número de elementos que tenga dicha media. En medias de periodos grandes por ejemplo 200, donde antes se hacía 200 sumas ahora se hace 1 suma y una resta.

Hemos reescrito la función y la hemos implementado en la librería, ver el extracto del código en la figura 5.3.

```
      // hay menos datos aún que periodo tiene la media, caso especial
 if (offset<funcs[index].period) then
   begin
   media:=0;
   for i:=0 to offset do
        media:=media+ precio[i];
   media := media / (offset+1)

   end
else
    begin     // caso general del algoritmo
     media := funcs[index].lastbar+(precio[offset]-precio[offset-funcs[index].period] ) * funcs[index].period_inverse;
   end;
```

Figura 5.3. Extracto de la implementación de la media simple

Para este proyecto se han intentado abarcar un abanico de técnicas de mejora: en secuencial ,en paralelo y estructuras de datos adicionales, por eso en la parte de secuencial nos hemos centrado en optimizar la función de la media que es la que usa nuestro sistema de ejemplo más intensamente, pero no sólo se ha implementado la media simple (SMA), también se han implementado las otras funciones que requiere el sistema de ejemplo (y que en realidad son usadas por muchos sistemas), Average Directional Index (ADX)[11], Bandas de Bollinger (Bollinger Bands)[12] y también la media exponencial (EMA)[13], en ésta última la mejora no es significativa ya que en esta media, Metatrader ya implementa un algoritmo que aprovecha el cálculo del paso anterior. En versiones futuras se puede trabajar más sobre estas otras funciones para optimizarlas e incluso ampliar la librería con nuevas funciones de uso común. En la figura 5.4 podemos ver las funciones implementadas y como se enlaza nuestra librería en el código de Metatrader.

```
#property copyright "Copyright 2016, Manuel Segura Sagredo"
#property version   "1.00"
#property strict
#include <varios.mqh>

#import "mt4Parallel.dll"

   int pSma (int period,int hebra,int StoreResult);
   int pEma (int period,int hebra,int StoreResult);
   int pAdx (int period,int hebra,int StoreResult);
   int pBB (int period,double desviacionStandar,int hebra,int StoreResult);


   void  Calculate (uint fecha, double CurrentPrecio,double &res[]);
   void  Initialize(int StoreResults);
   void CreateThreads(void);
   void StopThreads(void);
   void SetThreadsParams (int hebras, int mainthreadworks,int prioridad);


#import
```

Figura 5.4 Enlace de las funciones de la librería en un sistema en Metatrader.

El haber implementado todas las funciones que necesitamos en la librería es para buscar su posible ejecución como tareas paralelas como veremos en el siguiente subcapítulo.

5.2. Extracción de paralelismo

La siguiente mejora que se puede implementar es la ejecución en paralelo de las funciones matemáticas requeridas por el sistema de trading, aprovechando las capacidades multicore de los equipos actuales y que Metatrader no aprovecha.

Ya que como hemos comentado en el subcapítulo anterior, no podemos acceder al código fuente de Metatrader para hacer ejecute funciones en paralelo, si podemos hacer que las funciones de nuestra librería que nosotros compilamos si aprovechen el paralelismo, así tendríamos dos formas de abordar el problema:

- Hacer que las funciones tengan una ejecución paralela, por ejemplo el cálculo de una media de 200 periodos sea dividida en 4 procesos en paralelo de 50 periodos y luego se unifiquen los resultados en una única media.

- Separar las funciones a calcular y repartirlas en distintos procesos en paralelo, siempre que no existan dependencias, o sea hacer tareas paralelas[PAT14]

Tras algunas medidas experimentales hemos visto que el rendimiento siempre es peor en paralelo que en secuencial abordando el problema desde el punto de vista primero, es decir dividiendo la función en varios procesos paralelos. Esto se debe a que los cálculos no llegan a tener la suficiente complejidad como para que merezca la pena partir el problema en 4 o más hebras, partir una media de 200 periodos por ejemplo en 4 partes de 50 periodos, tiene más penalización por la gestión de las hebras que la mejora que se pueda obtener.

Abordaremos el problema desde el punto de vista segundo, es decir intentaremos enviar a cada proceso paralelo una función completa para ser calculada.

Tras probar varias técnicas de programación paralela en Windows como son la nueva implementación de bucles paralelos y Tareas en Delphi XE 10 (parallel library), librerías de terceros como son Omnithread[8], lo que mejor rendimiento nos ha dado ha sido el uso de los threads a nivel de sistema operativo o Windows threading [GOV11].

Estas funciones de bajo nivel para el tratamiento de hebras en Windows, Delphi las encapsula como clases dentro de la librería: system.threading, haciendo que su uso sea relativamente sencillo [HAR00], como vemos en la figura, heredamos una clase de Tthread y así se crea la clase que va a contener nuestro tipo de hebra personalizado, las variables de ésta clase serán privadas a la hebra y el método que hace el cálculo continuamente es el método "Execute". Es Interesante destacar que para optimizar el cómputo, se guardan los índices de las funciones que le tocan a la hebra en una variable de tipo vector. Es decir hay un vector global con las funciones a calcular. Esto es una optimización para no tener que preguntar continuamente en el cálculo de la hebra que funciones le toca ejecutar a la misma. Lo calculamos al crear la misma y anotamos los índices de esas funciones en el vector global (ver el vector "func" en la clase de la hebra, en la figura 5.5).

También siempre que podamos tendremos calculado en variables globales los límites de bucles etc, para eso esta la variable nf_limit, que guarda el número de funciones a ejecutar por la hebra -1 y evitar su cálculo en cada iteración.

```
TMyThread = class(TThread)
private
  { Private declarations }
  idx:integer;   // indice de la hebra
   // lista de funciones que le tocan a esta hebra para
   //ya tenerlas preparadas y no recorrer la lista de funciones en cada iteración
  func:array of integer;
 // length(func) -1 , limite superior del bucle for , lo tenemos calculado por velocidad
  nf_limit:integer;
protected
  procedure Execute; override;
public
  SW:TStopWatch;
  job:integer;
  cont:integer;
  constructor Create(idx:integer;run:boolean); overload;
  destructor Destroy;override;
end;
```

Figura 5.5. Extracto de la implementación de la clase que gestiona las hebras.

En la figura 5.6 podemos ver el código del método Execute:

```
procedure TMyThread.Execute;
var i:integer;
 begin
 while not self.Terminated do
       begin
         if job=1 then
          begin
               sw.Start;
          for i:=0 to nf_limit do
                    begin
                    funcs[self.func[i]].funcion (self.func[i]);
                    inc(cont);
                     end;
                   job:=0;
                   sw.Stop;

          end;
       end;
 end;
```

Figura 5.6. Extracto de la implementación del método Execute de una hebra de la librería.

Para poder llamar en paralelo a las funciones del sistema de trading, lo que necesitamos hacer, como comentábamos en el subcapítulo de la optimización del código secuencial, es codificar de nuevo todas las funciones que utiliza el sistema, ya que el código de las funciones que utiliza Metratrader está ya compilado para ser secuencial y no se puede cambiar. Así que hemos codificado de nuevo todas las funciones en la librería (algunas más optimizadas) y esto nos va a permitir que las llamemos en paralelo, es lo que se hace dentro del método "Execute" de la hebra, llamar a todas la funciones que han sido asignadas a esta hebra.

Lo que es en el código fuente del sistema de trading en metatrader, la inicialización del programa (función Oninit()) cambia un poco respecto a la de un sistema estándar en Metatrader, ya que tendremos que especificar qué funciones va a calcular en cada iteración ya que para que sea lo más óptimo posible, especificamos las funciones a calcular al iniciar el programa y en cada iteración simplemente se pasa el nuevo precio en una función y la misma devuelve un vector con los cálculos de todas las funciones ya hechos, para minimizar el paso de parámetros y de funciones entre el sistema de trading y la librería, ver figura 5.7.

```
SetThreadsParams (hebras,mainthreadworks,threadpriority);
ArrayResize(resu,20);
Initialize (StoreResults);
pSma(200,1,0);// SmaPeriod,ThreadNo,StoreResult
pSma(100,1,0);
pSma(50,0,0);
pSma(20,2,0);
pSma(10,0,0);

pAdx(20,2,1);// AdxPeriod,ThreadNo,StoreResult
pAdx(100,1,1);
pBB(70,2,0,1);// BollingerBandsPeriod,StdDesviation,ThreadNo,StoreResult
CreateThreads();
```

Figura 5.7. Fragmento de código de inicialización de la librería

Podemos observar en el fragmento de código que la primera función establece el número de hebras que utilizaremos para paralelizar funciones y si la hebra principal del programa también hace cálculos o no, normalmente este parámetro siempre será 1, es decir True ya que según las mediciones de tiempo suele dar mejor rendimiento. También hay que comentar que aparte de los parámetros específicos de cada función (periodo de la media, desviación estándar en las Bandas de Bollinger etcétera) cada función siempre tiene dos parámetros adicionales y que son la hebra a la que le vamos a enviar la función para ser ejecutada (penúltimo parámetro) y si almacenamos el cálculo en memoria o no. El almacenamiento del cálculo lo veremos en la siguiente sección. Respecto al parámetro de hebra que ejecuta la función, nos sirve para balancear la carga que recibirá cada hebra.

Esta tarea de balancear la carga de las hebras es vital ya que la función que devuelve el precio esperará hasta que todas las hebras hayan terminado su trabajo para devolver el resultado, luego la hebra que más tarde será la que condicionará el tiempo. La idea es que no haya hebras mucho tiempo ociosas. [RAU12]

El balanceo de carga para este ejemplo se hace de forma manual, para ello se ha implementado en la librería un Profiler el cual nos va a escribir en un fichero log al final del proceso el tiempo de trabajo que ha tenido cada hebra con una precisión de microsegundos. Lo que hacemos es establecer en que hebra se ejecuta cada función intentado repartir la carga equitativamente. Esto en éste sistema de ejemplo que estamos estudiando no va a ser difícil, ya que tenemos 8 funciones y esto nos da juego a repartirlas entre 2, 3 ó 4 hebras, pero en sistemas que tengan pocas funciones o que una de las funciones tengan un tiempo muchísimo mayor que las demás el balanceo de carga será imposible y la paralelización de los procesos se verá penalizada. Ver en la figura 5.8, 3 hebras con los ticks de CPU ejecutados por cada una (1 milisegundo=10000 ticks) , el contador indica el nº de funciones ejecutadas por esa hebra, en este caso suman 575.257.176 funciones ejecutadas (8 x 70 millones

aproximadamente). En este caso la hebra 0 está trabajando bastante menos y la 1 más que las demás.

```
Hebra 0 ticks=28737845  cont= 71907147
Hebra 1 ticks=95565144  cont=287628588
Hebra 2 ticks=67315615  cont=215721441
```

Figura 5.8 Ticks de Cpu y número de funciones ejecutadas por hebra

Tendremos que establecer en el penúltimo parámetro de las llamadas a las funciones la hebra que lo ejecuta 0,1,2 ... para conseguir equilibrar la carga. Si pasamos -1 en dicho parámetro la librería automáticamente le asignará una hebra.

Como mejora de este proyecto podríamos implementar algún algoritmo de balanceo de carga dinámico, que estudie el tiempo de ejecución de cada función y reparta el trabajo equitativamente entre cada hebra siempre que sea posible. [MAT11]

Volviendo al código del sistema, también cambiará la función de las iteraciones (OnTick()) que es la que es llamada en cada cambio de precio como dijimos al describir un programa en Metrader, ver figura 5.9

```
// función que envía el nuevo precio que hay en el evento
// OnTick(), espera el calculo de las hebras y recoge
// en el vector 'resu' los resultados
 Calculate (Time[0],Close[0],resu);
 m200 =resu[0];
 m100 =resu[1];
 m50 =resu[2];
 m20 =resu[3];
 m10 =resu[4];
 adx20 =resu[5];
 adx100=resu[6];

 bblow=resu[7];
 bbhigh=resu[8];
```

Figura 5.9 Código que hace la llamada al cálculo de las funciones.

Ahora como vemos en el fragmento de código, hay una función que inserta el nuevo precio que ha llegado al sistema y la hora actual, dicha función envía a las hebras el nuevo precio y espera a que calculen el resultado en paralelo y recoge en un vector los

resultados, asignándolos a las variables necesarias. Esto cambia un poco la filosofía, ya que antes al ser el código secuencial simplemente se calcula la función y se espera a que devuelva el resultado y así con todas, ver figura 5.10

```
m200 = iMA (NULL,0,200,0,MODE_SMA,PRICE_CLOSE,0);
m100 = iMA (NULL,0,100,0,MODE_SMA,PRICE_CLOSE,0);
m50  = iMA (NULL,0,50,0,MODE_SMA,PRICE_CLOSE,0);
m20  = iMA (NULL,0,20,0,MODE_SMA,PRICE_CLOSE,0);
m10  = iMA (NULL,0,10,0,MODE_SMA,PRICE_CLOSE,0);
adx20 =iADX(NULL,0,20,PRICE_CLOSE,MODE_MAIN,0);
adx100 =iADX(NULL,0,100,PRICE_CLOSE,MODE_MAIN,0);
bblow =   iBands(NULL,0,bbperiod,2,0,PRICE_CLOSE,MODE_LOWER,0);
bbhigh =  iBands(NULL,0,bbperiod,2,0,PRICE_CLOSE,MODE_UPPER,0);
```

Figura 5.10. Detalle del código original y las llamadas en secuencial a las funciones.

También en este punto hay que comentar que el sustituir cálculos de funciones que hace Metatrader por los implementados en la librería que sean más eficientes, como es natural pueden ser combinados entre sí, es decir usar la media simple de la librería que es muy rápida con otras funciones que no tengamos implementadas en la librería y sí en Metatrader. Eso sí, si queremos usar cálculos en paralelo por fuerza habrá que hacerlo usando exclusivamente funciones de la librería.

5.3. Utilizando estructuras de datos para evitar cálculos redundantes

La última mejora implementada en la librería, viene de nuevo a trabajar sobre el hecho de que la fase de optimización consiste en dar muchas pasadas haciendo muchos cálculos redundantes sobre los datos históricos probando combinaciones de los parámetros de entrada. En nuestro ejemplo, tenemos 2 parámetros de entrada los puntos de Beneficio donde cerrar la operación y los puntos máximos de pérdida donde también cerraremos la operación. Luego si vamos a hacer miles de pasadas calculando las mismas medias, Adx y bandas de Bollinger, podríamos hacer esos cálculos en la primera pasada y guardar el

resultado para las siguientes. Para esto vamos a implementar una estructura de datos donde se guardará el resultado de las funciones que vayamos calculando.

Si volvemos a ver la figura 5.7 sobre la inicialización del sistema, en las llamadas a la librería, el ultimo parámetro de las funciones es 1 si usamos ésta característica de almacenar un resultado ya calculado ó 0 si no la usamos, con esto lo que haremos es que una vez calculada esa función para el precio actual del activo, guardamos su resultado en memoria, para la siguiente vez que se nos vuelva a pedir ese cálculo.

El inconveniente que tiene esta técnica es su gran consumo de memoria RAM, para 5 años de datos históricos necesitamos unos 600 Mbytes de datos por cada función que agregamos a la estructuar de datos, en el sistema de prueba y en un equipo con un procesador Intel i5 y 8 Gbytes de Ram, nos permite como mucho usar esta técnica de memoria con 4 funciones. Para seleccionar a qué funciones, de las 8 que tenemos en el sistema de ejemplo, le aplicaremos esta técnica, vemos el log que hemos comentado en la sección anterior, podemos ver que funciones consumen más tiempo y así asignarles a ellas la característica de memorizar el resultado. En nuestro caso, el cálculo de las bandas de Bollinger consume más que las otras funciones, así que le vamos guardando los resultados, en realidad consume como si fueran 2 funciones, ya que las bandas de Bollinger hacen 2 cálculos, la banda superior y la inferior, y la siguiente función que más consume es el Adx, como lo calculamos 2 veces, memorizamos también esas dos funciones. Si seguimos asignando almacenando más funciones en esta estructura de datos, el sistema nos dará un mensaje de error y no funcionará por falta de memoria ya que ya hemos asignado toda.

Tendremos que hacer una ejecución previa en modo tester para que el sistema calcule todos los datos y los almacene en la memoria, una vez hecho esto, si repetimos el cálculo y damos una segunda pasada, la mejora es notable también. Hay que recordar aquí que la mejora la vamos buscando para el modo optimización, así que dar una pasada de más no va a influir nada, cuando en el modo optimización puede que repitamos el cálculo cientos o miles de veces.

Tras esta mejora volvemos a ajustar el balanceo de carga de las hebras ya que ha variado al usar memoria caché. Algunas hebras habían reducido su carga así que volvemos a balancear y testeamos con todas las mejoras juntas.

Una posible mejora de la estructura de datos para reducir el consumo de memoria RAM, sería paginar a disco los resultados en lugar de tenerlo todo en memoria principal, ya que las pasadas se hacen desde los datos más antiguos a los más modernos, esto nos da juego para poder ir cargándolos por ejemplo de año en año y asi no consumir tanta RAM. Eso sí la escritura en disco será más lenta que en memoria así que habría que minimizar el nº de lecturas del disco.

6. Resultados obtenidos y trabajos futuros

Los siguiente experimentos se van a hacer usando Metratrader 4, sobre un equipo con Windows 7 profesional 64 bits, procesador i5-4460 con 4 cores de 3.2 Ghz, 8 Gbytes RAM y disco duro SSD 512 Gbytes. El activo seleccionado será el EURUSD en periodos de 5 minutos. Los datos históricos van desde 3-5-2010 a 30-07-2015 y los datos históricos han sido proporcionados por el bróker Alpari. Se han fijado como valores de los parámetros del sistema, PipsStop=120 y PipsTp =10 y el sistema es "Orden Perfecto" .

Las medidas de tiempo que se utilizarán para comparar las distintas optimizaciones se hacen en el modo "Tester" es decir modo histórico de 5 años de datos y con unos parámetros del sistema siempre iguales. Ya que el modo optimización lo que hace es dar muchas pasadas sobre los mismos datos buscando los parámetros óptimos, así que para nuestra medición es más práctico utilizar el modo Tester (una sola pasada) que es más rápido de cronometrar. Pero la importancia de los resultados serán extrapolables al modo Optimizador que es el que realmente nos interesa, así 10 segundos que se mejore el sistema en el modo Tester se multiplicarán por miles de veces que son las pasadas que hace el modo Optimizador buscando los parámetros óptimos

Como podemos ver en la figura 6.1 el sistema tarda 170 segundos con su código original es decir 2′ 50″.

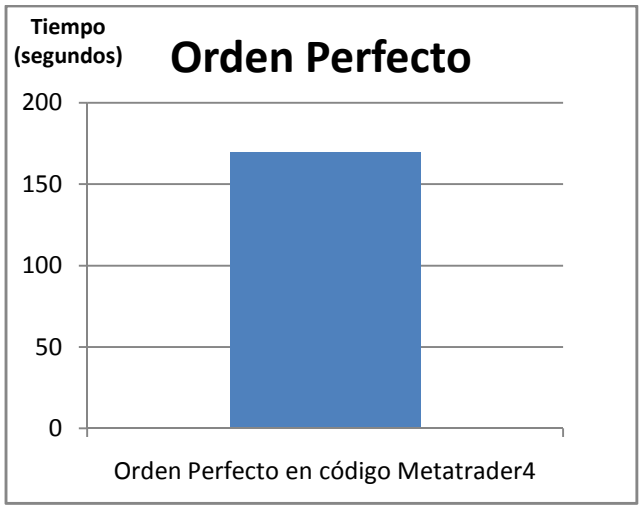


Figura 6.1 Tiempo de ejecución del sistema "Orden Perfecto" en modo Tester.

6.1 Evaluación de la optimización del código secuencial

Mejorando el algoritmo de la media simple, la mejora de tiempo en el robot o sistema de trading es notable, ver figura 6.2:

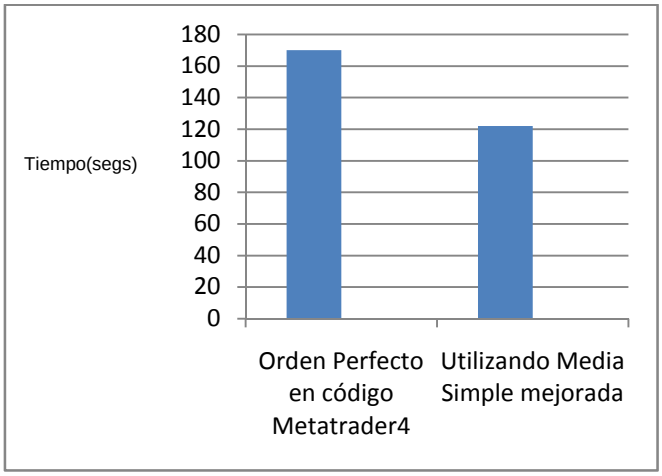


Figura 6.2 Tiempo de ejecución del sistema "Orden Perfecto" en modo Tester.

El tiempo total en el cálculo del sistema es 122 segundos, frente a los 170 originales, lo cual hace que sea un 70% del tiempo original.

6.2 Evaluación del paralelismo

Para la implementación del paralelismo, medidos los tiempos para distinto número hebras, obtenemos la siguiente tabla de tiempos:

Hebras	Tiempo
1	2,02
2	1,55
3	1,49
4	1,56

Tabla 6.1 Tiempo de ejecución del sistema "Orden Perfecto" con distintas hebras.

El tiempo total en el cálculo del sistema queda ahora en 109 segundos, frente a los 170 originales, lo cual hace que sea un 64% del tiempo original, podemos ver la comparativa en la figura 6.3.

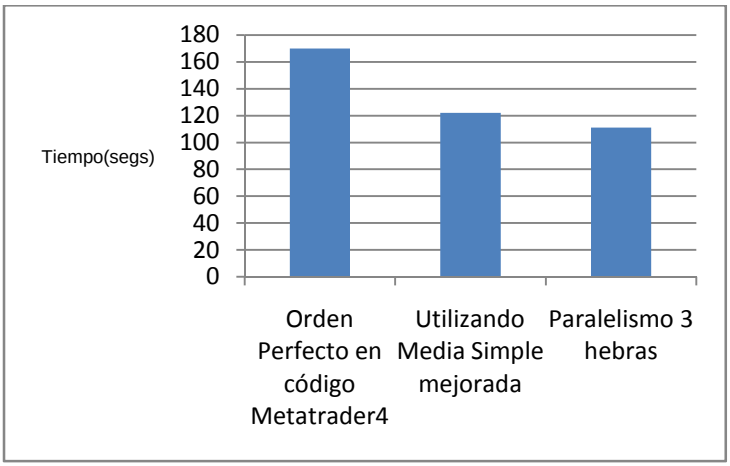


Figura 6.3 Tiempo de ejecución del sistema "Orden Perfecto" usando paralelismo

6.3 Evaluación de uso estructuras de datos auxiliares

En tercer lugar probamos la última mejora implementada que es el uso de una estructura de datos para almacenar resultados ya calculados.

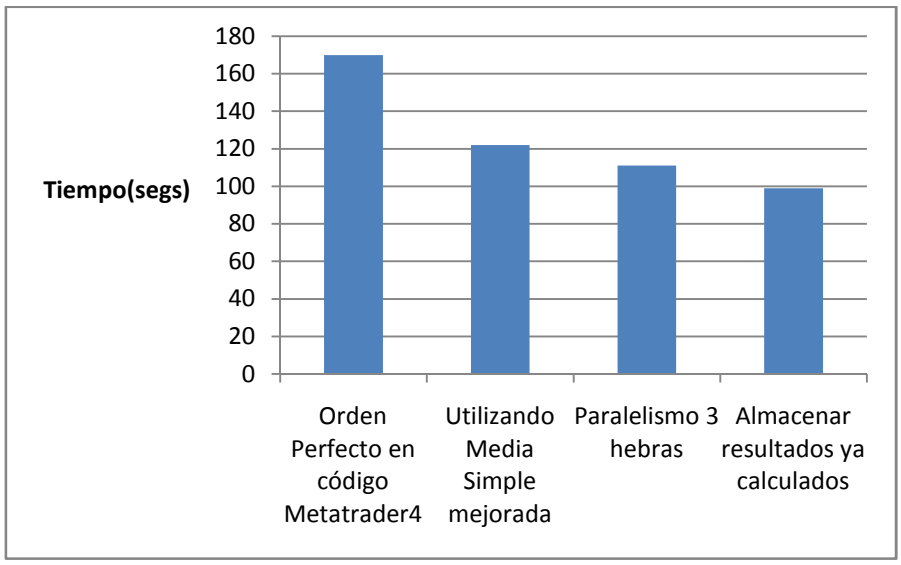


Figura 6.4 Tiempo de ejecución del sistema "Orden Perfecto" almacenando resultados

El tiempo total en el cálculo del sistema queda ahora en 93 segundos, frente a los 170 originales, lo cual hace que sea un 54,70% del tiempo original.

Ya que hemos medido los tiempos haciendo una pasada, vamos a volver a optimizar el sistema de ejemplo, el "Orden Perfecto" pero esta vez usando las mejoras desarrolladas.

Optimizando el sistema, si recordamos con el algoritmo genético tardó 5h y 47 minutos, frente a las 57h de la búsqueda exhaustiva, pues bien, optimizando con el algoritmo genético y aplicando las 3 mejoras aportadas, nos da un tiempo de 2h 47 minutos.
Ver figura 6.5.

Es importante decir que estas mejoras no son exclusivas de éste sistema de ejemplo que hemos utilizado, sino que son aplicables a muchísimos otros, ya que casi todos utilizan medias u otras funciones que utilizan medias o alguna de las funciones implementadas, casi todos aplican varias funciones y en todos se pueden aprovechar la redundancia de cálculos. También hay que decir que tampoco se requiere un uso de Hardware adicional.

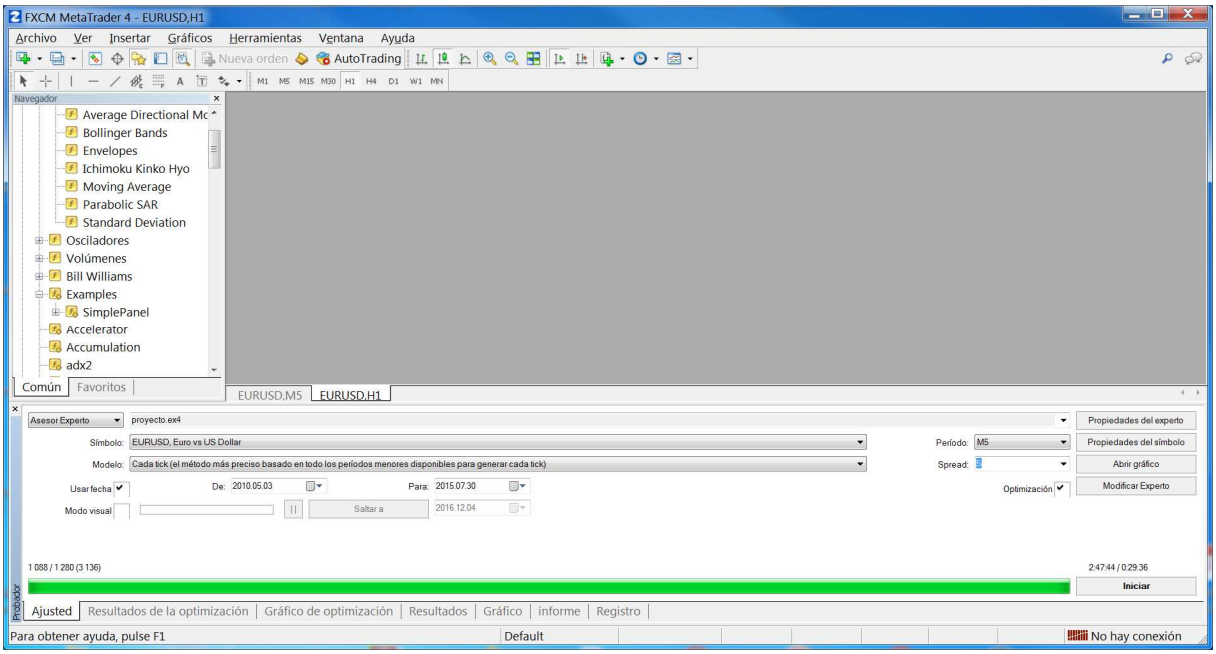


Figura 6.5 Optimización del sistema incluyendo las 3 mejoras.

6.4 Trabajo futuros

Como posible trabajos futuros no cabe duda que intentar mejorar el rendimiento del algoritmo genético de Metatrader, estaría en una de las primeras posiciones, ya que por la tecnología usada en el desarrollo de Metatrader no se ha usado paralelismo y puede ser un algoritmo que aproveche las capacidades multi-core actuales.

También se podría investigar la aceleración del problema de optimización usando técnicas de computación distribuida, para repartir el problema entre varias CPUs.

7. Conclusiones

En el objetivo principal del trabajo fin de grado estaba poder reducir en tiempo la fase de optimización de los sistemas de trading en Metatrader 4, que puede llegar a ser casi inabordable en muchas ocasiones.

Hemos reducido el tiempo a un 54% de su tiempo original para un sistema de ejemplo, el "orden perfecto", luego hemos cumplido de forma bastante satisfactoria el requisito principal del proyecto.

Un objetivo casi de la misma importancia, era respetar el código fuente de los sistemas ya existentes en Metatrader 4, ya que era otra prioridad que la amplia variedad de sistemas ya existentes en Metatrader 4, sigan funcionando. Aunque es cierto que tenemos que hacer modificaciones en el código, se han reducido a pequeños cambios en la inicialización y luego reemplazar las llamadas a las antiguas funciones por los resultados que nos da la función de cálculo de nuestra librería. Así que podemos decir que respetamos el código existente.

Otra conclusión es que para que el paralelismo de funciones merezca la pena, el sistema de trading debe tener suficientes funciones que calcular, ya que si tiene pocas, no merece la pena hacer el cálculo utilizando hebras.

Por último decir que la mejora de código secuencial a veces puede tan espectacular que haga que no se aprecie tanto otras posibles mejoras como el paralelismo.

BIBLIOGRAFIA

[BOL02] Bollinger, John, "Bollinger on Bollinger Bands", McGraw Hill, 2002

[HOL75] Holland,John, "Adaptation in Natural and Artificial Systems", Mit Press, 1975

[LIE08] Lien, Kathy – "Day Trading & Swing trading the Currency Market", Wiley Trading, 2008.

[MUR99] Muphy, John J. "Análisis Técnico de los mercados financieros"

[WIL98] Wilder, Welles. "New Concepts in technical Trading systems", 1978

[GES10] Gestal,Marcos. Rivero,Daniel. Rabuñal,Juan Ramón. Dorado, Julián. Pazos, Alejandro. "Introducción a los Algoritmos Genéticos y a la Programación Genética", Universidad de A Coruña, 2010

[DAR59] Darwin, Charles. "The Origin of Species by Means of Natural Selection", www.gutenberg.org/ebooks/2009 ,1859

[MAT11] Matloff, Norm. "Programming on parallel machines", University of California, Davis, 2011.

[GOV11] Gove, Darryl. "Multicore Application Programming: for Windows, Linux and Oracle Solaris", Addison-Wesley 2011.

[HAR00]Harvey, Martin. "Multithreading – The Delphi Way",University of Cambridge, 2000.

[SIN13] Singh, Mario. "17 Proven Currency Trading Strategies – How to Profit in the Forex Market", Wiley Trading Series. 2013

[PAT14] David A Patterson, John L Hennessy, "Computer Organization and Design: The Hardware/Software Interface" Fifth Edition, Edt. The Morgan-Kaufman, 2014

[RAU12] T Rauber, G Runger. "Parallel Programming: for Multicore and Cluster Systems". Second Edition. Edt. Springer, 2012

RECURSOS ONLINE

[1] http://www.tecnicasdetrading.com/2010/10/expert-advisors-metatrader4.html

[2] http://www.pullback.es/que-es-el-trading-automatico/

[3] http://www.metatrader4.com

[4] http://www.prorealtime.com/es/pdf/probuilder.pdf

[5] http:// www.visualchart.com

[6] http://www.rankia.com/blog/ea-expert-advisors/1837244-webinar-ventajas-desventajas-metratader-5-pablo-ortiz

[7] http://www.mql5.com/es/articles/1409

[8] http://www.omnithreadlibrary.com

[10] http://quivofx.com/expert-advisor/

[11] https://www.mql5.com/es/code/7955

[12] http://www.metatrader4.com/es/trading-platform/help/analytics/tech_indicators/bollinger_bands

[13] https://www.mql5.com/es/code/7534

UNIVERSIDAD DE ALMERÍA

Resumen/Abstract

En este Trabajo Fin de Grado se ha estudiado el problema de la optimización computacional de sistemas de trading, en particular sobre una de las plataformas más conocidas internacionalmente como es MetaTrader.

El objetivo ha sido la mejora del rendimiento en dicho proceso de optimización, mediante la mejora de los algoritmos de cálculo actuales, realización de cálculos en paralelo y otras mejoras.

Otro principio que se ha seguido para llegar a conseguir este objetivo es el respeto del código fuente de los sistemas ya existentes en Metatrader 4 en la medida de lo posible.

Se ha implementado una solución en forma de librería o extensión de Metatrader y con la cual hemos podido probar experimentalmente que ambos objetivos han sido cumplidos satisfactoriamente.

This project is focused on the high computational requirements of trading systems optimization, specifically on Metatrader, the most famous trading software in the world.

The goal has been the improvement of performance in this optimization process by means of making faster currents algorithms, by the application of parallel computing and more progresses.

An important principle that has guided this work has been to keep previous source code in Metatrader 4 with the minimal changes as much as possible.

The solution has been implemented as Metatrader library (DLL) through which we have tested in experimental mode that both goals have been successfully achieved.

GRADO EN INGENIERÍA INFORMÁTICA, 2016/2017