



TRABAJO DE FIN DE MÁSTER  
MÁSTER EN INGENIERÍA INFORMÁTICA

# Servicio Web de GPU

---

**Autor**

José Cristóbal López Zafra

**Tutor**

María Isabel García Arenas



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE  
TELECOMUNICACIÓN

Granada, 12 de septiembre de 2016





# Servicio Web de GPU

Funciones de optimización en algoritmos genéticos

José Cristóbal López Zafra



Figura 1: Logo de SWGPU



# Agradecimientos

En primer lugar, me gustaría agradecerle a Maribel su ayuda en todo el trabajo, su dedicación y confianza puesta en mí.

A mis padres, por apoyarme siempre y aguantarme todos esos días de malhumor cuando el trabajo se hacía cuesta arriba.

A Israel, Patricio y compañeros de clase en este máster, en especial a Juan y Miguel, por estar siempre dispuestos a todo, tanto dentro de las clases como fuera.



# Índice general

<b>1. Resumen</b>	<b>1</b>
1.1. Resumen y palabras clave . . . . .	1
1.2. Extended abstract and keywords . . . . .	2
<b>2. Objetivos y motivación</b>	<b>3</b>
2.1. Objetivos . . . . .	4
2.2. Motivación . . . . .	5
<b>3. Estado del arte</b>	<b>7</b>
3.1. Actualidad . . . . .	9
3.2. Clientes . . . . .	11
3.3. Competidores . . . . .	11
3.4. Conclusiones . . . . .	12
<b>4. Resolución del trabajo</b>	<b>13</b>
4.1. Planificación . . . . .	14
4.1.1. Conteo de horas . . . . .	14
4.1.2. Presupuesto . . . . .	15
4.2. Tecnologías . . . . .	17
4.2.1. Servidor . . . . .	17
4.2.2. Cliente . . . . .	19
4.3. Herramientas . . . . .	22
<b>5. Descripción del sistema</b>	<b>25</b>
5.1. Análisis inicial . . . . .	25
5.2. Objetivos . . . . .	26
5.3. Especificación de requisitos . . . . .	27
5.3.1. Requisitos funcionales . . . . .	27
5.3.2. Requisitos no funcionales . . . . .	27
5.3.3. Requisitos de información . . . . .	28
5.3.4. Restricciones . . . . .	30
5.4. Modelo funcional . . . . .	32
5.4.1. Resumen de actores . . . . .	32
5.4.2. Identificación de los casos de uso . . . . .	32

5.4.3.	Descripción de los casos de uso . . . . .	33
5.4.4.	Diagramas de caso de uso . . . . .	39
5.4.5.	Diagrama de comunicación . . . . .	41
5.5.	Flujo de la interfaz . . . . .	42
<b>6.</b>	<b>Desarrollo del sistema</b>	<b>43</b>
6.1.	Arquitectura del sistema . . . . .	43
6.2.	Partes del sistema . . . . .	44
6.3.	Filosofía a seguir . . . . .	46
6.3.1.	Desarrollo general . . . . .	46
6.3.2.	Modularización . . . . .	46
6.3.3.	Control de versiones . . . . .	46
6.3.4.	Desarrollo iterativo incremental . . . . .	47
6.3.5.	Revisiones periódicas del código . . . . .	47
6.3.6.	Documentación del código . . . . .	47
6.4.	Estructura . . . . .	48
6.4.1.	FrontEnd . . . . .	48
6.4.2.	BackEnd . . . . .	48
6.4.3.	Ejecutable a usar por el BackEnd . . . . .	49
6.5.	Pruebas . . . . .	51
<b>7.</b>	<b>Manuales</b>	<b>55</b>
7.1.	Manual de usuario . . . . .	55
7.2.	Manual de despliegue . . . . .	59
7.2.1.	Requisitos para el despliegue . . . . .	59
7.2.2.	Despliegue . . . . .	59
7.2.3.	Despliegue automatizado . . . . .	60
<b>8.</b>	<b>Conclusiones y trabajos futuros</b>	<b>61</b>
8.1.	Conclusiones . . . . .	61
8.2.	Trabajos futuros . . . . .	62



# Índice de figuras

1.	Logo de SWGPU . . . . .	I
2.1.	Evolución de las GPUs respecto a las CPUs . . . . .	5
3.1.	Función y representación de la función de optimización de Ackley . . . . .	8
3.2.	Función y representación de la función de optimización de Rastrigin . . . . .	9
3.3.	Ejemplo de flujo de procesamiento CUDA . . . . .	10
4.1.	Diagrama Gantt de la planificación prevista del trabajo . . .	14
4.2.	Diagrama Gantt de la planificación del trabajo . . . . .	14
5.1.	Diagrama de caso de uso en Gestión del Servicio web . . . . .	39
5.2.	Diagrama de caso de uso en Gestión del algoritmo . . . . .	40
5.3.	Diagrama con la comunicación entre las distintas partes . . .	41
5.4.	Flujo de la interfaz web . . . . .	42
6.1.	Estructura del sistema . . . . .	44
6.2.	Modelo Vista Controlador . . . . .	45
6.3.	Petición de un algoritmo genético mediante el ejecutable . . .	49
6.4.	Salida del algoritmo genético . . . . .	49
6.5.	Especificaciones de las tarjetas 840M y GTX660 . . . . .	51
6.6.	Comparativa de las tarjetas mediante benchmarks de GPU . .	52
6.7.	Comparativa de tiempos entre tarjetas . . . . .	52
7.1.	Captura de la portada del servicio web . . . . .	55
7.2.	Captura de los servicios disponibles . . . . .	56
7.3.	Formulario para lanzar el algoritmo genético . . . . .	56
7.4.	Mensaje de error en el formulario . . . . .	57
7.5.	Resultados obtenidos . . . . .	57
7.6.	Sección de contacto . . . . .	58
7.7.	Captura del despliegue automático . . . . .	60



# Capítulo 1

## Resumen

### 1.1. Resumen y palabras clave

**Palabras clave:** *servicio web, evaluación de algoritmos genéticos, GPU, CUDA.*

En este trabajo se abordará la creación de un conjunto de funcionalidades que permitan ejecutar algoritmos genéticos y ser optimizados. Para un máximo rendimiento se paralelizarán los algoritmos haciendo uso de una GPU. Acceder a esas funcionalidades será posible desde un servicio web que interactúe con ellas.

Al abordar en el proyecto se han encontrado un amplio y variado abanico de retos y problemas relacionados con el diseño, el campo de la inteligencia artificial, el desarrollo software con distintas tecnologías y la forma de conectarlas.

Para resolverlo se implementa un servicio web, que se amplía a un servicio web que además hace uso de GPUs: dicho servicio llama a la GPU para ejecutar un algoritmo evolutivo, y usar una función de evaluación mediante el algoritmo de Ackley o Rastrigin. Esa infraestructura que da soporte a las llamadas del servicio usa la arquitectura de cálculo CUDA.

Al finalizar el trabajo se obtiene un servicio web que permite ejecutar algoritmos genéticos, haciendo uso de la GPU de un servidor externo mediante CUDA.

## 1.2. Extended abstract and keywords

**Keywords:** *web service, evaluation of genetic algorithms, GPU, CUDA.*

The objective of this project will be the creation of a group of functionalities which allows execute genetic algorithms and optimise them. They will be parallelized using a GPU for maximum performance. Access to these functionalities will be possible from a web service that interact with them.

While the project was implemented, a lot of challenges and problems have been found. They were related to the design, the subject of artificial intelligence, software development using different technologies and how to connect them.

A web service is implemented to make the system, and it extends to a web service that also makes use of GPUs: this service use the GPU to execute a genetic algorithm, and use a evaluation function (Ackley or Rastrigin function). The infrastructure that supports service calls use the computing architecture CUDA.

At the end of the project a web service that allows to run genetic algorithms is obtained. This service makes use of the GPU of an external server using CUDA.

## Capítulo 2

# Objetivos y motivación

Un algoritmo es una serie de pasos organizados que describe el proceso que se debe seguir, para dar solución a un problema específico. A principios de la década de 1960, en 1962 John Henry Holland ideó una de las líneas más prometedoras de la inteligencia artificial, la de los algoritmos genéticos, y con su libro “Adaptation in Natural and Artificial Systems” en 1975 logró una mayor popularidad.

Estos algoritmos son llamados así porque se inspiran en la evolución biológica y su base genético-molecular. Estos algoritmos hacen evolucionar una población de individuos sometiénola a acciones aleatorias semejantes a las que actúan en la evolución biológica (mutaciones y recombinaciones genéticas), así como también a una selección de acuerdo con algún criterio, en función del cual se decide cuáles son los individuos más adaptados, que sobreviven, y cuáles los menos aptos, que son descartados.

Para su funcionamiento (el de un algoritmo genético simple, llamado Canónico), al empezar necesita una codificación o representación del problema que se adecue al mismo y una función de ajuste o adaptación al problema. Durante la ejecución del algoritmo, se seleccionan unos individuos para cruzarlos, y al individuo que resulte de este cruce se le hará una mutación. El resultado será un conjunto de individuos (posibles soluciones al problema) que formarán parte de la siguiente población. El algoritmo repetirá el proceso hasta encontrar una solución óptima o las veces que se le indique.

Poseen multitud de aplicaciones, como pueden ser:

- Diseño automatizado para la investigación de diseño de materiales [52], equipamiento industrial o sistemas de comercio en el sector financiero.
- Construcción de árboles filogenéticos. [47]

- Diseño de sistemas de distribución de aguas [51].
- Resolución de equilibrios en Teoría de juegos [49].
- Análisis de expresión de genes [50].

Pueden resolver problemas de alta complejidad, pero esto suele traducirse en operaciones muy costosas en términos de tiempo y recursos. En la actualidad, por ejemplo, hay casos en los que recrear una simulación de la solución propuesta por una iteración puede durar varios días y consumir gran cantidad de procesamiento y recursos asociados.

Estos algoritmos consumen muchos recursos y estos recursos no siempre están disponibles o al alcance de todo el mundo. Por esta razón, este proyecto Fin de Máster propone hacer accesible recursos de computación de bajo coste y alta disponibilidad a usuarios que no tienen la posibilidad de pagar dicho servicio o que no los tienen disponibles en su máquina. Estos recursos de computación son las tarjetas gráficas programables o GPU [21]. Las GPUs ponen a disposición de los usuarios un recurso de computación paralelo muy potente a un precio no muy elevado y están siendo en los últimos años una tendencia en la paralelización de los algoritmos genéticos. Esta tendencia permite el desarrollo de este trabajo Fin de Máster estableciendo un primer paso en el desarrollo de servicios web que permitan acceder a estos recursos y que puedan estar instalados en cualquier máquina de esta escuela, facilitando el acceso a los alumnos a estos recursos sin necesidad de estar presentes en las aulas.

## 2.1. Objetivos

- Desarrollar el conjunto de clases que permita acceder de forma remota a un conjunto de recursos para la ejecución de algoritmos evolutivos paralelos.
- Acceder a un recurso de computación remoto a través de un interfaz web.
- Aprender a utilizar recursos de computación paralela de bajo coste como son las GPUs
- Estudiar el rendimiento obtenido para esta interfaz web en comparación con otras alternativas.

## 2.2. Motivación

Dicha complejidad en la resolución de los algoritmos genéticos lleva a optimizarlos y a trabajar con ellos para lograr reducir el coste de dichas operaciones.

Esto lleva a utilizar una computación paralela, en la que se aprovecha la potencia de computación de los sistemas con los que trabajamos realizando cálculos simultáneamente. Sigue el principio de dividir los problemas grandes en problemas más pequeños, que se solucionan en paralelo para luego obtener la solución del problema inicial.

Por otra parte, en el ámbito de la computación se ha visto una gran evolución de las tarjetas gráficas, ya que grandes fabricantes como NVIDIA, AMD, IBM o Intel han logrado una producción de GPUs de gran alcance, como se puede ver en la Figura 2.1 (pueden incluso superar la frecuencia de reloj de una CPU antigua).

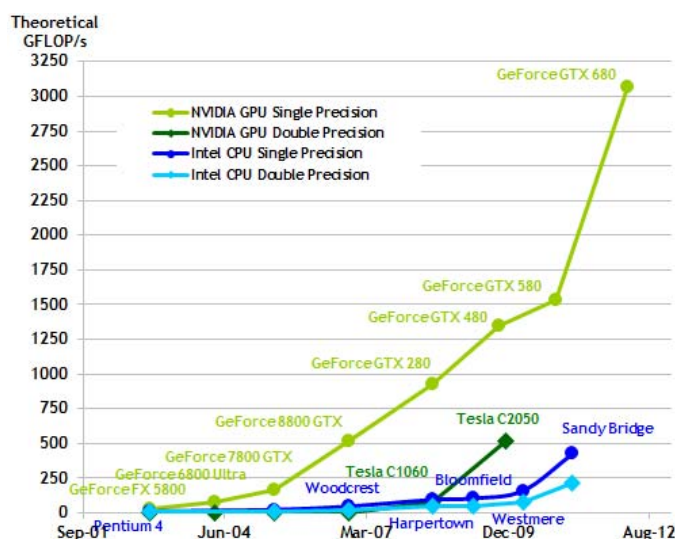


Figura 2.1: Evolución hasta 2012 de las GPUs respecto a las CPUs

El uso de computación paralela, junto la potencia y la posibilidad de usar el paralelismo que ofrecen las GPU hace muy interesante el uso de dichas GPUs para resolver algoritmos genéticos de formas más eficiente.

Después del estudio realizado en el siguiente capítulo, se opta por usar la arquitectura de cálculo CUDA [37] para crear algoritmos genéticos que se ejecuten en GPUs de NVIDIA [33].

Como se cita en uno de los objetivos, se publicará de manera que cualquier usuario tenga acceso. Esto, junto a una interfaz sencilla, permitirá computar algoritmos genéticos con facilidad y rapidez.





## Capítulo 3

# Estado del arte

Los principios básicos de los algoritmos genéticos fueron establecidos por Holland (1975), y se encuentran bien descritos en varios textos a lo largo de los años, como los estudios de Goldberg (1989), Davis (1991), Michalewicz (1992) o Reeves (1993), afianzando los conceptos y permitiendo trabajar sobre una buena base a futuros investigadores.

Son muchos y muy variados los ámbitos en los que se usan los algoritmos genéticos. Sirven para crear componentes automovilísticos, analizar expresiones de genes o hasta desarrollar aprendizajes de comportamiento para robots, por ejemplo. Incluso se usan en aquellos que otros métodos no consiguen resolver o tiene problemas, o dichos métodos se pueden llegar a conjuntar con los algoritmos genéticos para así mejorarlos.

Aunque no se garantiza que el algoritmo genético encuentre la solución óptima del problema, éste encontrará soluciones de un nivel aceptable, en un tiempo competitivo con el resto de los algoritmos de optimización combinatoria.

Esto hace que se estudie y se intente mejorar y optimizar dichos algoritmos, siendo un campo con mucha transcendencia en la actualidad.

Constan de varias partes, como la selección de candidatos, operadores de cruce, funciones de evaluación y optimización o mutación. En este trabajo nos centraremos en la parte de optimización, donde se evalúan los individuos generados y que serán posibles soluciones al problema. Existen multitud de funciones de optimización, pero nos centraremos en 2: Ackley y Rastrigin.

La función de Ackley se publicó por primera vez en “A connectionist machine for genetic hillclimbing” por David H. Ackley en 1987 y se extendió a

la dimensión arbitraria en “Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms” por Thomas Bäck en 1996.

$$f(x) = 20 + e - 20e^{-\frac{1}{5}\sqrt{\frac{1}{n}\sum_{i=1}^n x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^n \cos(2\pi x_i)}$$

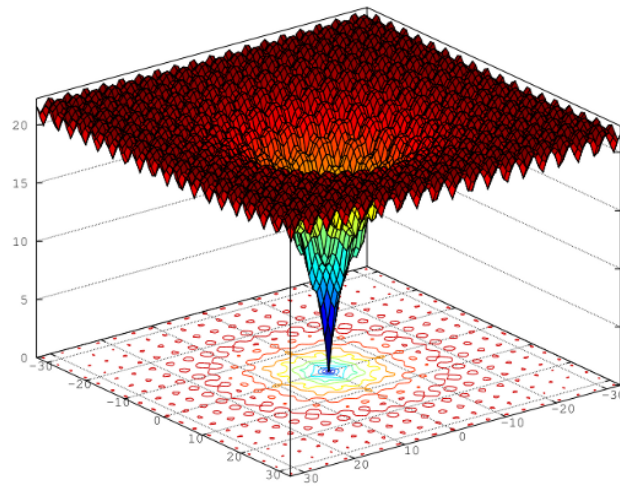


Figura 3.1: Función y representación de la función de optimización de Ackley

Años antes, en 1974, Rastrigin en “Systems of extremal control” propone otra función de optimización, para ser generalizada en 1991 por Mühlenbein.

Este trabajo se centrará en estas 2 funciones de optimización, ofreciendo un algoritmo genético que pueda ser optimizado mediante Ackley o Rastrigin.

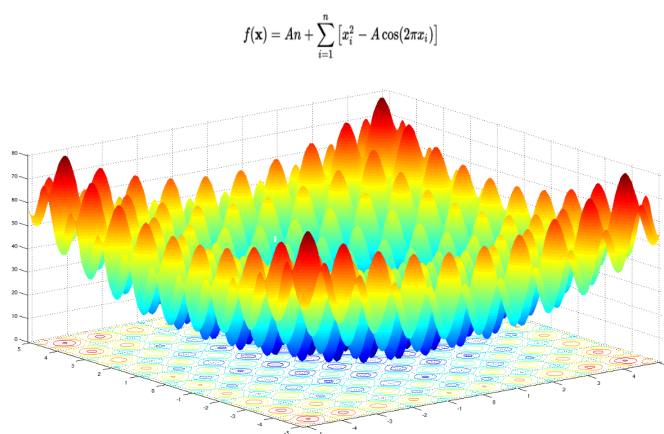


Figura 3.2: Función y representación de la función de optimización de Rastrigin

### 3.1. Actualidad

En la actualidad la mayoría de los usuarios que necesitan procesar un algoritmo genético lo hacen ellos mismos, con programas adaptados de terceros o desarrollados por ellos mismos. Esto conlleva un trabajo extra de estudio, implementación y corrección de dichos programas.

Podemos encontrar algunos programas [1] [28] [20] para ser ejecutados por el cliente, o servicios web que realizan algoritmos genéticos donde la mayoría son ejemplos o demostraciones de algoritmos genéticos y sus soluciones [41] [42].

Pero ninguno usa la computación paralela aprovechando la potencia de las GPUs. Viendo que hay disponible en el ámbito de dicha paralelización vemos que hay grandes empresas que han lanzado lenguajes de programación enfocados a aprovechar el procesamiento mediante la GPU: CUDA [37], AMD OpenCL APP [4], BrookGPU [7], PeakStream [35] o RapidMind [39]:

- CUDA: arquitectura de cálculo paralelo de NVIDIA. Se basa en el lenguaje C y C++, por lo que, junto con la cantidad de dispositivos de la marca, existe una gran comunidad y documentación.
- AMD OpenCL<sup>TM</sup> Accelerated Parallel Processing: herramienta de AMD que permite el cómputo mediante GPUs. Se basa en los lenguajes OpenCL y C++, por lo que se pueden usar para acelerar aplicaciones.
- BrookGPU: programa (en versión *beta*) de la Universidad de Stanford para aprovechar la paralelización en tarjetas gráficas AMD y NVIDIA. Para trabajar con el se usa una extensión de ANSI C.

- PeakStream era un programa para paralelizar el procesamiento con grandes rendimientos en tarjetas AMD, comprado por Google en 2007 [36], y tras esto, a dejado de ofrecer mantenimiento.
- RapidMind, que también se basa en C++, es comprada por Intel en 2009 y pasa a ser Intel Array Building Blocks [40], pero sigue estando en forma experimental.

Tras ver las distintas opciones, con sus distintas ventajas e inconvenientes, se decide centrar en CUDA. Es un proyecto activo, tiene numerosas actualizaciones, cuenta con mucha comunidad para resolver dudas y fomentar su desarrollo, y tiene numerosas facilidades, como un SDK [33] y varias herramientas para sus desarrolladores. Por todo esto se escoge CUDA para desarrollar el trabajo.

CUDA surgió en 2006, cuando NVIDIA lanzó por primera vez al mercado una GPU capaz de renderizar gráficos en 3D y que incluyó la posibilidad de ejecutar, usando CUDA, programas escritos en lenguaje C.

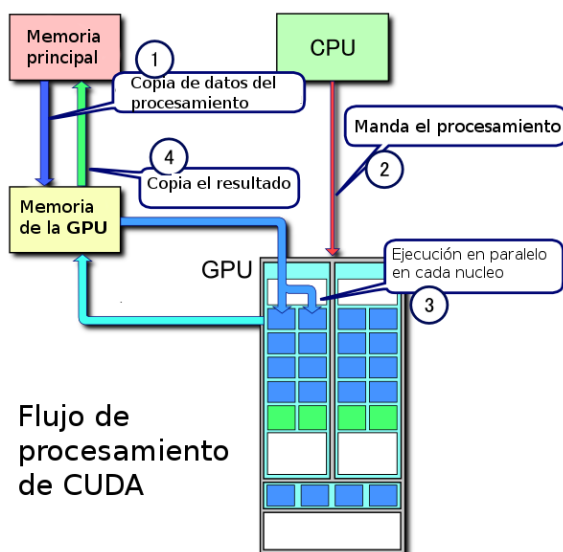


Figura 3.3: Ejemplo de flujo de procesamiento CUDA

Gracias a su eficiencia energética y gran potencia, desde entonces NVIDIA ofrece la capacidad de sus tarjetas gráficas para grandes centros de datos y otras instituciones tanto científicas como gubernamentales, permitiendo dibujar gráficos y ejecutar diversos programas aprovechando.

Incluso se utilizan versiones reducidas para potenciar las aplicaciones tanto de teléfonos móviles, ordenadores portátiles y tablets, entre otros.

En este campo hay algunos proveedores, como Impact [25] o gpuOcelot [22] que ofrecen herramientas para la computación de algoritmos estándar o del código que genere el usuario, pero realizando el cómputo desde las GPUs de los clientes, y nunca enfocados específicamente a resolver algoritmos genéticos.

Otras opciones permiten ejecutar CUDA en servidores externos, pero se necesita enviar una solicitud de uso y adaptarse a sus restricciones y capacidad, como en rCUDA [15] o desplegar toda una instancia en un IaaS y desarrollar dentro: Amazon Web Services [26].

Con esto se puede ver que podemos encontrar servicios que realicen algoritmos genéticos, o servicios que usen CUDA, pero no hay servicios que combinen ambos.

### 3.2. Clientes

¿Que usuarios necesitan computar algoritmos genéticos?

Por ejemplo, cualquier estudiante que quiera comprobar los distintos resultados del algoritmo genético cambiando sus parámetros. También el personal de investigación trabaja con multitud de algoritmos genéticos, y con grandes consumos de recursos. Junto con la potencia de cómputo y la accesibilidad que proporcionaremos simplificaríamos el trabajo de dicho personal.

Además el servicio será accesible a cualquier usuario, y con su interfaz sencilla e intuitiva dichos usuarios no necesitarán una preparación especial para usarlo.

### 3.3. Competidores

Como se cita antes, existen ejemplos o plantillas de algoritmos genéticos [1] [28] [20] que los usuarios pueden usar, pero necesitan para su uso un trabajo extra para su instalación, desarrollo o ajustes. Además de este trabajo extra, se ven limitados por las capacidades de sus dispositivos, pues los algoritmos se tendrán que lanzar en local.

Para evitar dichas limitaciones, podemos usar la paralelización de los algoritmos, pero los trabajos que encontramos se encuentran en una versión de CUDA obsoleta [46] (y simplemente exponiendo el código) o son estudios del servicio sin llegar a ser implementado [48]. Se busca entonces un servicio que ofrezca computación desde un servidor externo. Se pueden ver servicios

que computan directamente CUDA [15] u ofrecen instancias con acceso a GPUs [26] pero nunca especializados en algoritmos genéticos.

### 3.4. Conclusiones

Si se busca un servicio de computación externo (que ofrezca una mayor potencia de computación usando GPUs) de algoritmos genéticos no llegamos a encontrar nada que cumpla con nuestro requisitos: o son servicios en local para lanzar algoritmos genéticos o son servicios externos que nos ofrecen computación aprovechando la paralelización de CUDA, pero sin llegar a ofrecer ninguna aproximación de un algoritmo genético.

Viendo esto se encuentra una demanda que podemos cubrir con nuestro servicio, motivando a su análisis y desarrollo.

## Capítulo 4

# Resolución del trabajo

La planificación del proyecto se realizó al inicio del mismo y luego se fue alterando según las desviaciones que se producían. El inicio del proyecto fue el 1 de Abril, aunque hasta el 1 de Julio se trabajó en él de manera entrecortada y sin profundidad, y la fecha de fin planificada es el 10 de Septiembre.

Se podrán distinguir varias etapas en el trabajo, que se ven mejor en la siguiente imagen (Figura 4.1) de la planificación:

- Preparación y análisis: Aquí se hizo un análisis inicial del proyecto, con tecnologías a usar y un estudio más profundo de algunos conceptos vistos en el curso que se abordarían.
- Desarrollo del algoritmo genético: esta etapa consta de varias partes, ya que primero se trabajó con distintos algoritmos simples, genéticos o que usaban la paralelización a forma de formación y para facilitar un posterior diseño y análisis del algoritmo que se implementará. A medida que se acababa la implementación del algoritmo final, se hicieron pruebas con los resultados de ambas funciones de optimización.
- Desarrollo del servicio web: también consta de varias etapas. Primero se creó el BackEnd, y como el FrontEnd requería de los resultados del algoritmo, no se desarrolló hasta obtener algún resultado de este. Una vez completadas estas partes, se pasa al despliegue con una versión estable.
- Documentación: Aunque se ha ido preparando y almacenando información y recursos durante todo el transcurso del trabajo, la mayor parte de la memoria del trabajo se crea y se plasma en este documento en la etapa final, haciendo las correcciones y mejoras pertinentes.

## 4.1. Planificación

A continuación se muestra mediante un diagrama de Gantt la planificación estimada del trabajo.

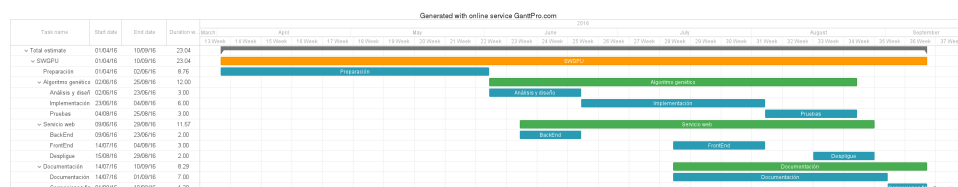


Figura 4.1: Diagrama Gantt de la planificación prevista del trabajo

Y en la siguiente figura se ve el desarrollo que se ha producido, tras desviaciones imprevistas o ajustes de tiempo durante el mismo proceso:

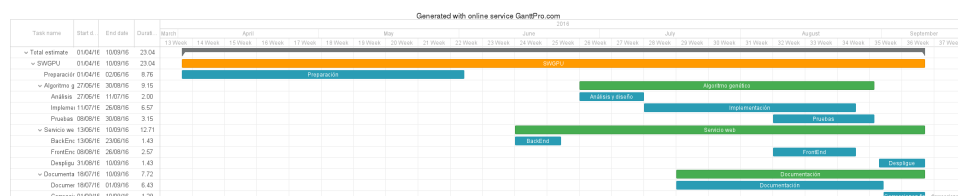


Figura 4.2: Diagrama Gantt de la planificación del trabajo

### 4.1.1. Conteo de horas

#### ■ Estimado

- Preparación:  
8,75 semanas, a 3 días/semana, a 2 horas/día: 52,5 horas
- Desarrollo de algoritmo genético:  
12 semanas, a 4 días/semana, a 4 horas/día: 192 horas
- Desarrollo de servicio web:  
7 semanas, a 5 días/semana, a 2 horas/día: 70 horas
- Documentación:  
8,29 semanas, a 4 días/semana, a 2 horas/día: 66,3 horas

Con lo que se sumaría un total de 380,8 horas.



- Real

- Preparación:  
8,76 semanas, a 3 días/semana, a 2 horas/día: 52,56 horas
- Desarrollo de algoritmo genético:  
11,72 semanas, a 4 días/semana, a 4 horas/día: 187,52 horas
- Desarrollo de servicio web:  
5,43 semanas, a 5 días/semana, a 2 horas/día: 54,2 horas
- Documentación:  
7,72 semanas, a 4 días/semana, a 2 horas/día: 61,76 horas

**Sumando finalmente un total de 356,04 horas.**

#### 4.1.2. Presupuesto

En este punto se verán los presupuesto según el conteo de horas para cada una de las 2 estimaciones y los servicios que se requieren (obviando el alquiler y mantenimiento del lugar de trabajo). Suponiendo la media actual de un programador Junior (Abril 2016) [31]: 11,25€/hora.

- Estimado

- Preparación:  
52,5 horas x 11,25€/hora : 590.62€
- Desarrollo de algoritmo genético:  
192 horas x 11,25€/hora: 2137.50€
- Desarrollo de servicio web  
70 horas x 11,25€/hora: 787.50€
- Documentación:  
66,3 horas x 11,25€/hora: 745.86€
- Ordenador con el que trabajar: se optará por el modelo Acer Aspire V3 [34], que cuenta con una tarjeta gráfica GeForce 840M [44] y que permitirá desarrollar y ejecutar algoritmos mediante CUDA 7.5.: 716€
- *Instancia que permita usar CUDA*  
Se contratarán servicios externos de cómputo mediante CUDA [26] con un coste de 1,88€/h (2,10\$/h). Suponiendo que este servicio estuviese en total disponibilidad (24 horas los 7 días de la semana) alcanzaría en sólo un mes de servicio los 315,84€.

- *Dominio para alojar una web publica del servicio*

En caso de que no fuese posible, se contrataría un dominio por precios de 2,08€ a 4,95€ al mes (con permanencia de 12 meses, en las webs de alojamiento *Hostpapa.es* [3] y *Dondominio.com* [2]): 2,08€

**Presupuesto total estimado: 5.295,40€ (para el desarrollo y sólo un mes en producción)**

■ Real

- Preparación

52,5 horas x 11,25€/hora: 590,62€

- Desarrollo de algoritmo genético

187,52 horas x 11,25€/hora: 2.109,6€

- Desarrollo de servicio web

54,2 x 11,25€/hora: 609,75€

- Documentación

61,76 x 11,25€/hora: 694,8€

- Ordenador Acer Aspire V3: 716€

- *Instancia que permita usar CUDA*

En esta parte se aprovechará el servidor que ofrece la tutora del trabajo María Isabel García Arenas, con gran capacidad de cómputo y de manera gratuita. Esto ayudará de manera muy importante al trabajo, ya que reducirá de manera muy importante el presupuesto final del proyecto.

- *Dominio para alojar una web publica del servicio*

Al igual que en el punto anterior, se aprovechará el servicio de la tutora para lanzar el servicio web, reduciendo también el presupuesto final.

**Presupuesto total real: 4.720,77 €**

Al final se reduce algo el presupuesto respecto al estimado (de 5.295€ a 4.720€), se producen varios cambios en la planificación y sobre todo se aprovecha el servidor ofrecido por la tutora.

## 4.2. Tecnologías

En este apartado se describirán las tecnologías utilizadas en el trabajo. Se desglosarán después de listarlas:

En la parte del servidor:

- Python [38]
- Django [12]
- Apache [6]
- C++ [8]
- CUDA [37]

Y de cara al cliente:

- HTML5 [23]
- CSS3 [11]
- JavaScript [27]
- AngularJS [5]
- jQuery [29]
- JSON [30]
- HTML5UP [24]

### 4.2.1. Servidor

- **Python**

Versión usada: 2.7.6

Python [38] es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible, además de simple y versátil.

Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

- **Django**

Versión usada: 1.10

Django [12] es un framework de alto nivel web de Python. Fomenta el rápido desarrollo y un diseño limpio y pragmático. Se encarga de facilitar la creación de aplicaciones web complejas, ofreciendo las funcionalidades básicas y más útiles al desarrollador, para que pueda centrarse en la escritura de su aplicación.

- **Apache**

Versión usada: 2.4.12

El servidor HTTP Apache [6] es un servidor web de código abierto, multiplataforma, extensible y modular. Además, al ser tan popular tiene un gran soporte de la comunidad.

Implementa el protocolo HTTP y la noción de sitio virtual.

- **C++**

Versión usada: 4.8.8

C++ [8] es un lenguaje de programación que extiende a C añadiendo mecanismos que permiten la manipulación de objetos.

Tiene las facilidades de programación genérica junto a los paradigmas de programación estructurada y programación orientada a objetos.

- **CUDA**

Versión usada: 7.5

CUDA [37] es una arquitectura de cálculo paralelo de NVIDIA que aprovecha la gran potencia de la GPU (unidad de procesamiento gráfico) para proporcionar un incremento extraordinario del rendimiento del sistema.

Dispone de una gran comunidad, ya que es la arquitectura de cálculo paralelo más usada en la actualidad, en parte gracias a su potencia y la facilidad de uso para los desarrolladores.

La plataforma de cálculo paralelo CUDA proporciona unas cuantas extensiones de C y C++ que permiten implementar el paralelismo en el procesamiento de tareas y datos con diferentes niveles de granularidad y con la flexibilidad de usar varios lenguajes, como son C, C++ y Fortran.

Dispone de numerosas facilidades para los desarrolladores, como herramientas de desarrollo, documentación y su *toolkit*.

### 4.2.2. Cliente

- **HTML5**

HTML (HyperText Markup Language) [23] en su la quinta revisión, es el lenguaje básico de la World Wide Web. Es un estándar que sirve de referencia de elaboración de páginas web en sus diferente versiones. Define una estructura básica y un código para la definición de contenido de una página web, como texto, imágenes o vídeos. Usa un sistema para formatear el layout de nuestras páginas, así como hacer algunos ajustes a su aspecto. Los navegadores como Chrome, Firefox, Explorer o Safari puede saber como representar el contenido de una web, con las ubicaciones de todos los elementos.

- **CSS3**

CSS (Cascading Style Sheets) [11] u hoja de estilo en cascada, es un lenguaje usado para definir y crear la presentación de un documento estructurado escrito en HTML o XML2. El World Wide Web Consortium (W3C) es el encargado de formular la especificación de las hojas de estilo, al igual que del lenguaje HTML, que servirán de estándar para los agentes de usuario o navegadores.

Aunque la información de estilo puede ser definida en el mismo documento HTML, la idea que se encuentra detrás del desarrollo de CSS es separar la estructura de un documento de su presentación.

- **JavaScript**

JavaScript [27] es un lenguaje de programación interpretado, que se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

Se utiliza principalmente en su forma del lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas (aunque existe una forma de JavaScript del lado del servidor).

Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM).

- **AngularJS**

Versión usada: 1.5.7

AngularJS [5] es un framework de JavaScript de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones web de una sola página. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Whatever (MVW), en un esfuerzo para hacer que el desarrollo y las pruebas sean más fáciles.

La biblioteca lee el HTML que contiene atributos de las etiquetas personalizadas adicionales, entonces obedece a las directivas de los atributos personalizados, y une las piezas de entrada o salida de la página a un modelo representado por las variables estándar de JavaScript, que además se pueden configurar manualmente, o recuperados de los recursos JSON estáticos o dinámico

- **jQuery**

Versión usada: 1.11.3

jQuery [29] es una biblioteca de JavaScript que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web.

jQuery, al igual que otras bibliotecas, ofrece una serie de funcionalidades basadas en JavaScript de manera muy reducida, por lo que con las funciones propias de esta biblioteca se logran grandes resultados en menos tiempo y espacio. Todo esto hace que sea la biblioteca JavaScript más usada.

- **JSON**

JSON (JavaScript Object Notation) [30] es un formato ligero de intercambio de datos. Gracias a su sencillez es fácil leerlo y escribirlo para humanos, mientras que para las máquinas es simple interpretarlo y generarlo.

Es un formato de texto que es completamente independiente del lenguaje pero utiliza convenciones que son ampliamente conocidos por los programadores de los principales lenguajes.

Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos.

- **HTML5UP**

HTML5UP [24]: Web con múltiples plantillas disponibles (usando HTML y JavaScript) y estilos CSS con los que crear la estructura básica de

proyectos web. Son completamente personalizables y tienen un diseño web adaptable (*responsive*) de manera que la web que creemos se adaptará a cualquier pantalla, punto muy importante en la actualidad, debido a la multitud de dispositivos y tamaños de pantalla que existen.

## 4.3. Herramientas

En este apartado se listarán y desglosarán las herramientas usadas para usar las tecnologías anteriores.

- **Nsight Eclipse Edition**

Versión usada: 7.5

Nsigh (NVIDIA Nsight Eclipse Edition) [32] es la gran plataforma de desarrollo para cálculo heterogéneo. Permite trabajar con potentes herramientas de depuración y análisis del rendimiento para optimizar el funcionamiento de la CPU y la GPU. Esta plataforma permite optimizar el rendimiento de manera intuitiva, identificar y analizar los cuellos de botella y observar el comportamiento de todas las actividades del sistema.

Consta de varias plantillas y ejemplos, además de estar disponible varios sistemas operativos.

- **Git**

Versión usada: 1.91

Git [18] es un software de control de versiones, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente.

Git se considera un sistema de control de versiones con funcionalidad plena y con multitud de características, como la gestión distribuida o la rápida gestión de ramas.

- **Sublime Text**

Versión usada: 2, versión de evaluación

Sublime Text [43] es un editor de texto y de código fuente. Originalmente fue desarrollado como una extensión de Vim y con el tiempo fue creando una identidad propia.

Posee varias características para los desarrolladores, como un *mini-mapa* del código, multi-selección y multi-cursor, búsqueda dinámica y soporte a multitud de lenguajes.

- **Gimp**

Versión usada: 2.8.10

GIMP (GNU Image Manipulation Program) [17] es un programa de edición de imágenes digitales en forma de mapa de bits, tanto dibujos como fotografías.



Tiene multitud de herramientas para el retoque y edición de imágenes o un dibujo libre.

Esta disponible para la mayoría de los sistemas operativos y en varios lenguajes.

- **GanttPRO**

GanttPRO [16] permite la gestión de proyectos mediante diagramas de Gantt.

Mediante su interfaz se puede planificar y gestionar proyectos con facilidad con diagramas de Gantt, mediante tareas, subtareas o duraciones dinámicas.

- **Draw.io**

Draw.io [13] es una herramienta para crear varios tipos de diagramas, entre ellos del tipo UML.



## Capítulo 5

# Descripción del sistema

En este capítulo se describe la fase de análisis del proyecto. Algunos requisitos o partes del capítulo fueron modificados o corregidos durante el trabajo. A continuación se muestran los definitivos.

### 5.1. Análisis inicial

Este proyecto fue propuesto desde un principio por María Isabel García Arenas, ya que en su ámbito de investigación requiere algoritmos evolutivos con muchos requerimientos de cómputo, por lo que al paralelizarlos se logran unos resultados más rápidos, y al estar disponible en un servicio web no es necesario un dispositivo específico.

Después de la propuesta del trabajo, y antes del desarrollo del mismo se hicieron diversas reuniones para establecer sus bases y alcance.

Para desarrollarlo se separó el trabajo en un *módulo* con las funcionalidades del algoritmo genético, y otro con el servicio web que pudiese aprovechar dichas funcionalidades.

En la selección de tecnologías a usar se optó por la arquitectura de cómputo CUDA, ya que es actualmente la más potente, como se ve en el capítulo anterior, y se dispone de una tarjeta gráfica NVIDIA de gran potencia.

En la parte del servicio web se optó por el framework python Django, ya que se estudió con profundidad a lo largo del curso y es en la actualidad es la mejor opción para realizar un servicio web.

Dichas tecnologías son actuales, con una gran comunidad detrás, de manera que están en continuo desarrollo y mejora, además de estar bien asentadas, dando robustez al trabajo.

## 5.2. Objetivos

Este trabajo tiene como principal objetivo el desarrollo de unas clases que lancen algoritmos genéticos de forma paralela, y puedan ser usados por un servicio web.

Los principales objetivos, de manera reducida, que se quieren alcanzar con este trabajo son:

- **OBJ. 1** Gestión del servicio web
  - **OBJ. 1.1** Ejecución de algoritmo genético.
  - **OBJ. 1.2** Consulta de la solución.
  - **OBJ. 1.3** Contacto con el administrador o personal de contacto.
- **OBJ. 2** Gestión del algoritmo genético
  - **OBJ. 2.1** Entrada de parámetros.
  - **OBJ. 2.2** Ejecución del algoritmo genético.
  - **OBJ. 2.3** Generar salida en formato JSON.

## 5.3. Especificación de requisitos

### 5.3.1. Requisitos funcionales

En esta sección del capítulo se definirán las características de alto nivel (requisitos funcionales) del sistema que son necesarios para las necesidades del usuario.

- **RF. 1 Gestión del servicio web**

- **RF. 1.1** Ejecución de algoritmo genético.
  - **RF. 1.1.1** Entrada de parámetros .
  - **RF. 1.1.2** Petición al servidor.
- **RF. 1.2** Consulta de la solución.
- **RF. 1.3** Contacto con el administrador o personal de contacto.

- **RF. 2 Gestión del algoritmo genético**

- **RF. 2.1** Entrada de parámetros.
- **RF. 2.2** Ejecución del algoritmo.
  - **RF. 2.2.1** Generación de poblaciones.
  - **RF. 2.2.2** Evaluación de cada individuo.
    - ◊ **RF. 2.2.2.1** Evaluación mediante la función Ackley.
    - ◊ **RF. 2.2.2.2** Evaluación mediante la función Rastrigin.
  - **RF. 2.2.3** Selección de individuos para el cruce.
  - **RF. 2.2.4** Cruce de individuos.
  - **RF. 2.2.5** Mutación del individuo.
  - **RF. 2.2.6** Reemplazo del individuo.
- **RF. 2.3** Generar salida en formato JSON.

### 5.3.2. Requisitos no funcionales

En esta sección se establecen los requisitos no funcionales.

#### Rendimiento

**RNF 1:** Será implementado con tecnologías que optimicen el rendimiento y la eficacia, tanto en el servicio web como en el algoritmo genético.

**Disponibilidad**

**RNF 2:** Se intentará que el servicio esté disponible las 24 horas del día, con una estructura segura y preparada ante los fallos.

**Accesibilidad**

**RNF 3:** Se logrará una gran accesibilidad al tratarse de un servicio web.

**Usabilidad**

**RNF 4:** La interfaz, con el orden y disposición de los elementos será lo más usable posible.

**Interfaz**

**RNF 5:** La interfaz será sencilla e intuitiva.

**RNF 6:** Se adaptará a cualquier tamaño de pantalla, tendrá un diseño adaptativo.

**Estabilidad**

**RNF 7:** Se buscará la mayor estabilidad en el sistema, usando tecnologías robustas y estables.

**Mantenimiento**

**RNF 8:** Se ha desarrollado de manera ordenada y documentando todos las partes, para que el mantenimiento pueda ser realizado por el mismo desarrollador o por terceros de manera sencilla.

**Soporte**

**RNF 9:** Se le facilitará al usuario contacto con alguien responsable del sistema, para que en caso de necesitar soporte le sea fácil y rápido.

**5.3.3. Requisitos de información**

En esta sección se establecen los requisitos de información, que estarán fuertemente relacionados a los requisitos funcionales, ya que será la información mínima para llevarlos a cabo.

- **RI. 1 Gestión del servicio web**

- **RI. 1.1** Ejecución de algoritmo genético.
  - Tamaño de la población
  - Número de cromosomas
  - Valor mínimo
  - Valor máximo
  - Probabilidad de cruce
  - Probabilidad de mutación
  - Número de generaciones
  - Parámetros para la función Ackley
    - ◇ Valor A
    - ◇ Valor B
    - ◇ Valor C
  - Parámetros para la función Rastrigin
    - ◇ Valor A

- **RI. 1.2** Consulta de la solución.

- Hora de consulta
- Rendimiento
- Tiempo de cómputo
- Tamaño del cómputo
- Hebras usadas
- Datos de entrada (RI 1.1)
- Información del dispositivo que ha realizado el algoritmo
- Mejores variables obtenidas
- Mejor fitness

- **RI. 2 Gestión del algoritmo genético**

- **RI. 2.1** Entrada de parámetros.
  - Dispositivo a usar
  - Función de optimización a usar
  - Tamaño de la población
  - Número de cromosomas
  - Valor mínimo
  - Valor máximo
  - Probabilidad de cruce
  - Probabilidad de mutación

- Número de generaciones
  - Parámetros para la función Ackley
    - ◊ Valor A
    - ◊ Valor B
    - ◊ Valor C
  - RI 2.1.2 Parámetros para la función Rastrigin
    - ◊ Valor A
- **RI. 2.2** Generar salida en formato JSON.
    - Rendimiento
    - Tiempo de cómputo
    - Tamaño del cómputo
    - Hebras usadas
    - Datos de entrada (RI 2.1)
    - Información del dispositivo que ha realizado el algoritmo
    - Mejores variables obtenidas
    - Mejor fitness

#### 5.3.4. Restricciones

Las restricciones del sistema serán las siguientes:

**RSTR 1 Gestión del servicio web (Ejecución del algoritmo genético):**

- RSTR 1.1: El tamaño de la población será menor de 10.000.
- RSTR 1.2: Habrá un máximo de 1.000 cromosomas (1.000 variables).
- RSTR 1.3: El valor máximo tendrá que ser más grande el el valor mínimo.
- RSTR 1.4: La probabilidad de cruce tendrá que ser entre 0 y 1.
- RSTR 1.5: La probabilidad de mutación tendrá que ser entre 0 y 1.



**RSTR 2 Gestión del algoritmo genético (Entrada de parámetros):**

- RSTR 2.1: Para escoger la función con la que optimizar introducimos 0 o 1.
- RSTR 2.2: El tamaño de la población será menor de 10.000.
- RSTR 2.3: Habrá un máximo de 1.000 cromosomas (1000 variables).
- RSTR 2.4: El valor máximo tendrá que ser más grande el valor mínimo.
- RSTR 2.5: La probabilidad de cruce tendrá que ser entre 0 y 1.
- RSTR 2.6: La probabilidad de mutación tendrá que ser entre 0 y 1.

## 5.4. Modelo funcional

### 5.4.1. Resumen de actores

- **Usuario**

Descripción: Representa al usuario que quiere hacer uso del sistema.

Tipo: Primario

Responsabilidad: Completar el formulario con los datos que requiera el algoritmo genético.

- **Administrador del sistema**

Descripción: Representa a la persona encargada de mantener, actualizar y gestionar el sistema.

Tipo: Secundario

Responsabilidad: Realizar todas las actividades de gestión y actualización del sistema.

- **Servidor**

Descripción: Representa al servidor, que realizará peticiones al sistema para ofrecérselas al Usuario.

Tipo: Primario

Responsabilidad: Ofrecer el servicio web al usuario y gestionar sus peticiones de cómputo.

### 5.4.2. Identificación de los casos de uso

Los distintos casos de uso serán:

- Para la gestión del servicio web:
  - **CU 1** Entrada de parámetros .
  - **CU 2** Petición al servidor.
  - **CU 3** Consulta de la solución.

- CU 4 Contacto con el administrador o personal de contacto.
- Para la gestión del algoritmo genético:
  - CU 5 Entrada de parámetros.
  - CU 6 Generación de poblaciones.
  - CU 7 Evaluación mediante la función Ackley.
  - CU 8 Evaluación mediante la función Rastrigin.
  - CU 9 Selección de individuos para el cruce.
  - CU 10 Cruce de individuos.
  - CU 11 Mutación del individuo.
  - CU 12 Reemplazo del individuo.
  - CU 13 Generar salida en formato JSON.

#### 5.4.3. Descripción de los casos de uso

- CU 1 Entrada de parámetros.
  - **Actores:** Usuario.
  - **Tipo:** Primario, esencial.
  - **Dependencias:**
  - **Precondición:**
  - **Postcondición:** El sistema hace una petición al servidor con los parámetros introducidos.
  - **Autor:** José Cristóbal López Zafra.
  - **Versión:** 1.0.
  - **Propósito:** Especificar los parámetros del algoritmo genético.
  - **Resumen:** El usuario introduce los parámetros (distintos a los declarados por defecto) que quiere para ejecutar el algoritmo genético.
- CU Petición al servidor .
  - **Actores:** Usuario.
  - **Tipo:** Primario, esencial.
  - **Dependencias:** CU 1.
  - **Precondición:** Los parámetros introducidos deben ser correctos.

- **Postcondición:** Se mandará una petición de cómputo al sistema que devolverá a la web.
  - **Autor:** José Cristóbal López Zafra.
  - **Versión:** 1.0.
  - **Propósito:** Confirmar la petición al servidor.
  - **Resumen:** El usuario realiza la petición de ejecución del algoritmo genético que busca.
- **CU 3** Consulta de la solución.
- **Actores:** Usuario.
  - **Tipo:** Primario, esencial.
  - **Dependencias:** CU 2.
  - **Precondición:** El usuario ha hecho la petición al servidor y este haber acabado.
  - **Postcondición:** Resultados visibles al usuario
  - **Autor:** José Cristóbal López Zafra.
  - **Versión:** 1.0.
  - **Propósito:** Permitir ver al usuario los resultados generados por el sistema.
  - **Resumen:** Una vez que se resuelve el algoritmo genético, el servidor devuelve la solución, maquetada para facilitar su consulta al usuario.
- **CU 4** Contacto con el administrador o personal de contacto.
- **Actores:** Usuario.
  - **Tipo:** Opcional.
  - **Dependencias:**
  - **Precondición:**
  - **Postcondición:** El usuario podrá comunicarse con alguien encargado del sistema.
  - **Autor:** José Cristóbal López Zafra.
  - **Versión:** 1.0.
  - **Propósito:** Poder notificar dudas o sugerencias del sistema.
  - **Resumen:** El usuario podrá mandar un correo o ponerse en contacto con alguien responsable del sistema al usuario, pudiendo notificar así fallos o sugerencias sobre el sistema.

- **CU 5** Entrada de parámetros.
  - **Actores:** Servidor.
  - **Tipo:** Primario, esencial.
  - **Dependencias:** CU 1 y CU 2.
  - **Precondición:**
  - **Postcondición:** Lanzar el algoritmo genético con los parámetros introducidos.
  - **Autor:** José Cristóbal López Zafra.
  - **Versión:** 1.0.
  - **Propósito:** Especificar los parámetros para lanzar el algoritmo genético.
  - **Resumen:** Tras leer los parámetros que especifica el usuario (o usar los declarados por defecto) se lanzará el algoritmo genético con ellos.
- **CU 6** Generación de poblaciones .
  - **Actores:** Servidor.
  - **Tipo:** Primario, esencial.
  - **Dependencias:**
  - **Precondición:**
  - **Postcondición:** Se tendrán una serie de poblaciones inicializadas.
  - **Autor:** José Cristóbal López Zafra.
  - **Versión:** 1.0.
  - **Propósito:** Generar una serie de poblaciones para el algoritmo genético.
  - **Resumen:** Se generan una serie de poblaciones con los valores inicializadas para usar por el algoritmo genético.
- **CU 7** Evaluación mediante la función Ackley.
  - **Actores:** Servidor.
  - **Tipo:** Opcional.
  - **Dependencias:** CU 5 y CU 6.
  - **Precondición:** Tener una serie de individuos en distintas poblaciones para ser evaluados.
  - **Postcondición:** Cada individuo tendrá su *fitness* o *capacidad*.
  - **Autor:** José Cristóbal López Zafra.

- **Versión:** 1.0.
  - **Propósito:** Obtener un *fitness* de los individuos.
  - **Resumen:** A cada individuo se le evaluará mediante la función Ackley, obteniendo un *fitness* para ser usado en una posterior selección.
- **CU 8 Evaluación mediante la función Rastrigin.**
    - **Actores:** Servidor.
    - **Tipo:** Opcional.
    - **Dependencias:** CU 5 y CU 6.
    - **Precondición:** Tener una serie de individuos en distintas poblaciones para ser evaluados.
    - **Postcondición:** Cada individuo tendrá su *fitness* o *capacidad*.
    - **Autor:** José Cristóbal López Zafra.
    - **Versión:** 1.0.
    - **Propósito:** Obtener un *fitness* de los individuos.
    - **Resumen:** A cada individuo se le evaluará mediante la función Rastrigin, obteniendo un *fitness* para ser usado en una posterior selección.
- **CU 9 Selección de individuos para el cruce.**
    - **Actores:** Servidor.
    - **Tipo:** Primario, esencial.
    - **Dependencias:** CU 5, CU 7 y CU 8.
    - **Precondición:** Los individuos tendrán sus respectivos *fitness*.
    - **Postcondición:** Se seleccionaran los individuos que se considere mejor.
    - **Autor:** José Cristóbal López Zafra.
    - **Versión:** 1.0.
    - **Propósito:** Seleccionar individuos para su posterior cruce.
    - **Resumen:**
- **CU 10 Cruce de individuos .**
    - **Actores:** Servidor.
    - **Tipo:** Primario, esencial.
    - **Dependencias:** CU 5 y CU 9.
    - **Precondición:** Tener individuos seleccionados para su cruce.

- **Postcondición:** Nuevos individuos resultado del cruce de los seleccionados.
  - **Autor:** José Cristóbal López Zafra.
  - **Versión:** 1.0.
  - **Propósito:** Lograr nuevos individuos mejores.
  - **Resumen:** Tras seleccionar los mejores individuos se cruzan para obtener una posible mejora.
- **CU 11 Mutación del individuo.**
    - **Actores:** Servidor.
    - **Tipo:** Primario, esencial.
    - **Referencias:** CU 5 y CU 10.
    - **Precondición:**
    - **Postcondición:** Se modifica parte de un individuo.
    - **Autor:** José Cristóbal López Zafra.
    - **Versión:** 1.0.
    - **Propósito:** Modificar al azar parte del cromosoma de los individuos.
    - **Resumen:** Se modifica al azar parte de los individuos (su cromosoma), permitiendo alcanzar zonas del espacio de búsqueda que no estaban cubiertas por los individuos de la población actual.
  - **CU 12 Reemplazo del individuo.**
    - **Actores:** Servidor.
    - **Tipo:** Primario, esencial.
    - **Referencias:** CU 5 y CU 11.
    - **Precondición:**
    - **Postcondición:** Se logra un mejor individuo.
    - **Autor:** José Cristóbal López Zafra.
    - **Versión:** 1.0.
    - **Propósito:** Actualizar a un mejor individuo.
    - **Resumen:** -una vez realizadas las anteriores funciones genéticas, se seleccionan los mejores individuos para conformar la población de la generación siguiente.

- **CU 13** Generar salida en formato JSON.
  - **Actores:** Servidor.
  - **Tipo:** Primario, esencial.
  - **Referencias:**
  - **Precondición:** Realizar el algoritmo genético.
  - **Postcondición:** Se tendrá en formato JSON la salida del algoritmo genético ejecutado.
  - **Autor:** José Cristóbal López Zafra.
  - **Versión:** 1.0.
  - **Propósito:** Facilitar al usuario la lectura de la salida.
  - **Resumen:** Tras realizar el algoritmo genético se obtiene una salida en formato JSON, que el usuario podrá ver a través de la web mediante su correspondiente maquetado.



#### 5.4.4. Diagramas de caso de uso

##### Gestión del Servicio web

En esta sección se verán las tareas que influyen en el servicio web. Este sistema hará posible que el usuario pueda realizar una petición de algoritmo genético con los parámetros que quiera. Para poder hacer la *petición al sistema* se tendrá que realizar una *entrada de parámetros* correcta, pudiendo, después de ejecutarla, *consultar la petición*.

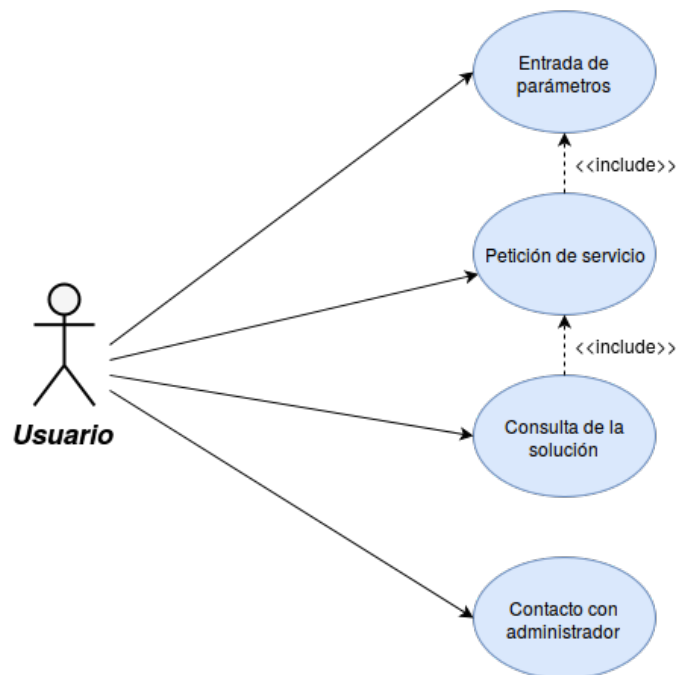


Figura 5.1: Diagrama de caso de uso en Gestión del Servicio web

### Gestión del algoritmo genético

En esta sección se verán las tareas que influyen en el algoritmo genético. El servidor hará uso del algoritmo genético implementado para aprovechar la GPU, con los parámetros que se le pasen (aunque si no se le pasan tendrá unos por defecto que se especifican) y generará una salida en formato JSON para poder trabajar con ella fácilmente. Para ello tendrá que haber ejecutado el algoritmo, y este a su vez realizar las distintas funcionalidades en un orden específico.

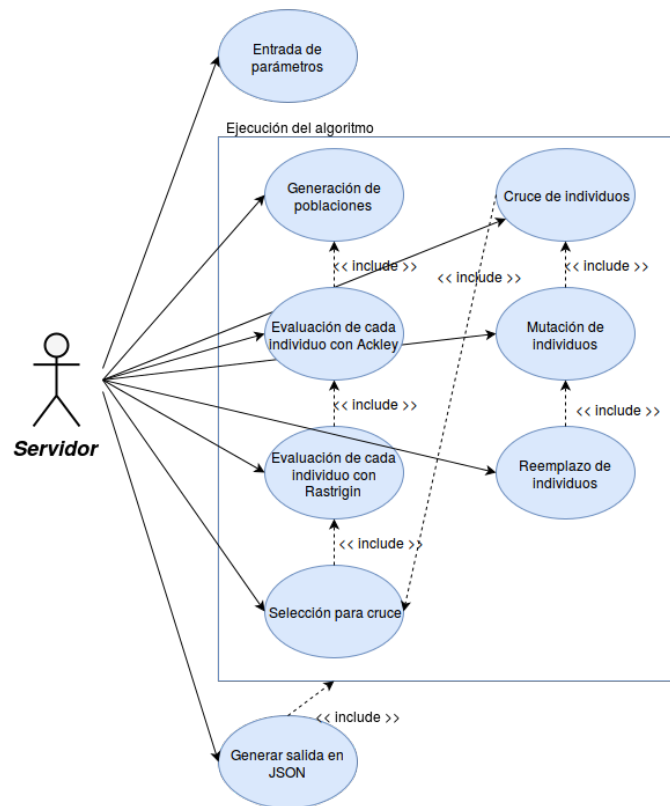


Figura 5.2: Diagrama de caso de uso en Gestión del algoritmo

### 5.4.5. Diagrama de comunicación

A continuación se muestra la comunicación entre las distintas partes simulando una *Petición de servicio*:

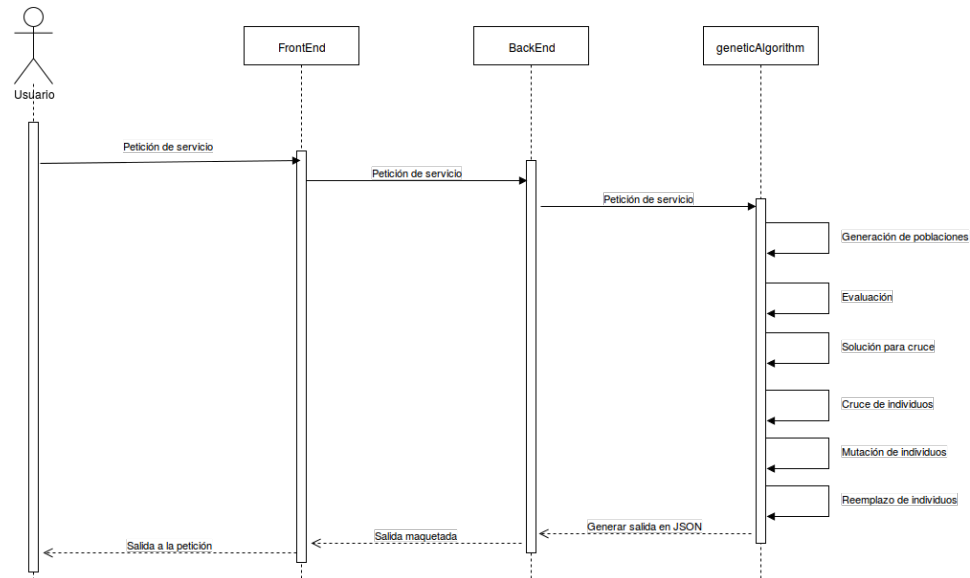


Figura 5.3: Diagrama con la comunicación entre las distintas partes

## 5.5. Flujo de la interfaz

En esta sección se verá la navegación que se puede hacer a través de la web. Será una única pantalla, pero las distintas opciones se mostrarán en varias *escenas* o *pantallas*, para hacer un avance fluido y sin necesidad de retroceder o avanzar perdiendo información entre distintas pantallas.

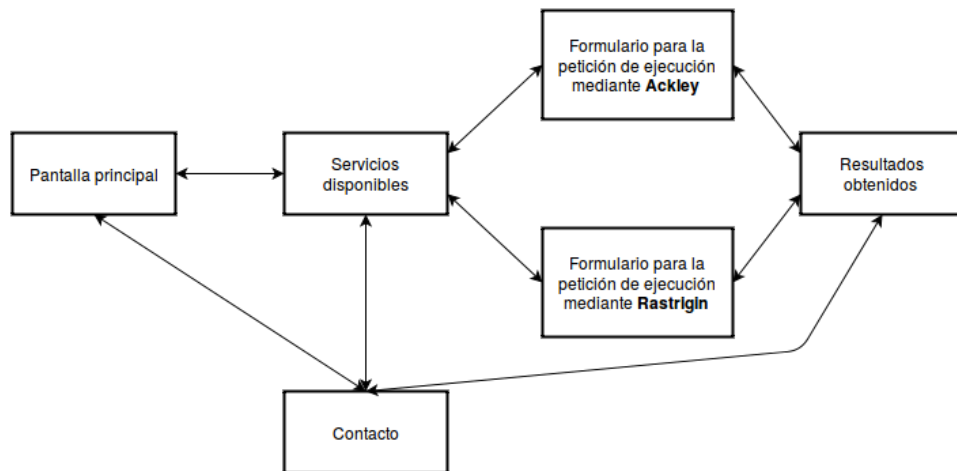


Figura 5.4: Flujo de la interfaz web

## Capítulo 6

# Desarrollo del sistema

En este capítulo veremos la arquitectura del sistema, la *filosofía* o principios seguidos y algunas partes del sistema.

El código del sistema se encuentra en GitHub. Para el sistema completo, que recoge el servicio web y la gestión de la ejecución del algoritmo genético: [SWGPU](#). Y para el módulo que gestiona el algoritmo genético: [geneticAlgorithm](#).

Y se encuentra disponible, en forma de *beta* en [www.genmagic.ugr.es:8000](http://www.genmagic.ugr.es:8000), accesible dentro de la red de la UGR [10].

### 6.1. Arquitectura del sistema

La arquitectura del sistema está compuesta por 2 capas: BackEnd y FrontEnd. La capa de BackEnd se sitúa en el servidor, mientras que la de FrontEnd en el lado del cliente.

La estructura básica de la web la formará desde el lado BackEnd (en el servidor) mediante Django (usando python) y será servida por el servidor Apache.

El cliente, desde el FrontEnd, verá la web mediante HTML5, CSS3 y JavaScript, y podrá interactuar con ella fácilmente gracias a las funcionalidades de jQuery y AngularJS

Otra vez en el BackEnd, tras la petición del cliente, el algoritmo genético se ejecutará en C++ y CUDA. La respuesta se le enviará en formato JSON, para ser maquetada de forma correspondiente.

En la siguiente imagen (Figura 6.1) se muestra con sus distintos elementos:

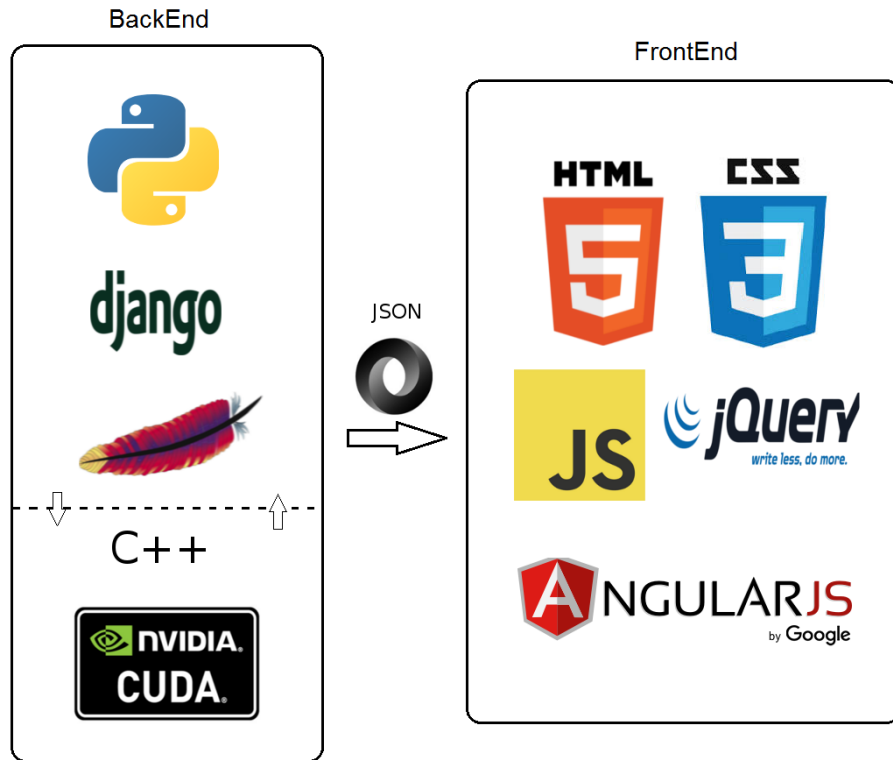


Figura 6.1: Estructura del sistema

## 6.2. Partes del sistema

### Backend

Será el lado del servidor de la aplicación. Basado en Django (y disponible mediante el servidor Apache), con éste tenemos acceso a los recursos del servidor, entre ellos la GPU. Con Django formaremos la petición según los parámetros que ha especificado el cliente, y se hará uso de la GPU mediante CUDA. Django tendrá la salida del cálculo, y la devolverá en un formato determinado.

Se puede decir que el Backend tiene 2 partes: el servidor web y el procesado dentro del servidor.

### FrontEnd

Lado que ve el cliente, el FrontEnd. Con las tecnologías básicas web (HTML5, CSS y JavaScript) se creará una interfaz para el cliente. Mediante

jQuery y AngularJS podrá interactuar fácilmente y con fluidez, además de servir la respuesta que le da el servidor.

AngularJS se encarga de la comunicación con el BackEnd a través de servicios, y con esos datos actualiza la interfaz. Conforma un modelo MVC (Modelo Vista Controlador), que separa los datos y su gestión (componente de modelo) de la aplicación de la interfaz de usuario (vista) y el módulo encargado de gestionar los eventos y las comunicaciones (controlador).

MVC propone la construcción de tres componentes distintos (modelo, vista y controlador): por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario.

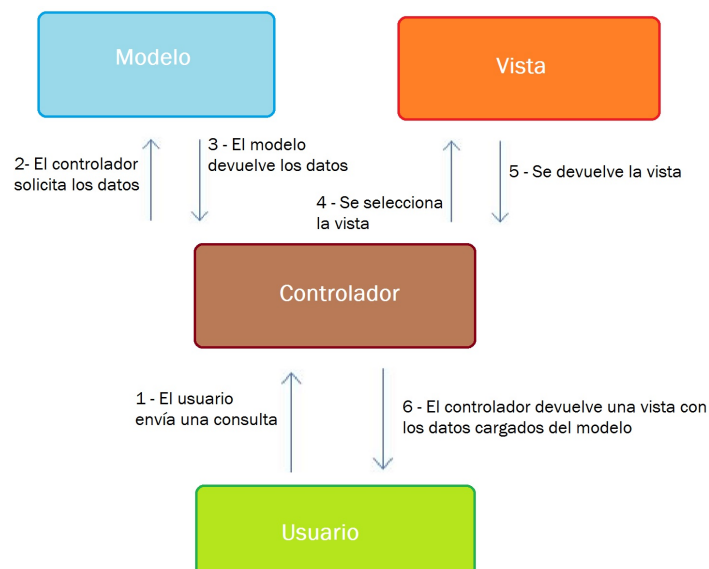


Figura 6.2: Modelo Vista Controlador

## 6.3. Filosofía a seguir

A la hora de implementar y de realizar el trabajo en general se ha intentado ser lo más ordenado y pulcro posible, esto en un principio puede ralentizar el proceso, pero a largo plazo hace que el desarrollo, actualización o mantenimiento del sistema sea más rápido y fácil. Estos son algunos de los criterios empleados en el desarrollo.

### 6.3.1. Desarrollo general

Las funciones del sistema se han desarrollado de la manera más general posible para así favorecer la reutilización de código y facilitar su legibilidad. También son más fáciles de mantener puesto que al ser de ámbito más general son más mantenibles.

### 6.3.2. Modularización

Se ha desarrollado el código en distintos módulos. De esta forma se evita que al hacer cambios en un módulo se propague a los demás, lo que hace el código más mantenible y eficiente.

### 6.3.3. Control de versiones

Se apostó por un sistema de control de versiones, en mi caso Git mediante la plataforma GitHub [19], ya que favorece el mantenimiento de las distintas versiones de código de una manera sencilla y rápida.

El manejo de Git es sencillo, ya que con unas cuantas ordenes se puede manejar sin problemas, y GitHub está provisto de una interfaz muy intuitiva.

Algunas de las órdenes que se han usado para el trabajo son:

*git clone[URL del proyecto]*: se descarga el proyecto del repositorio Git.

*git add [archivos]*: añade los archivos con los cambios deseados en un "paquete" para el commit.

*git commit*: subida de los archivos especificados al repositorio local.



*git push*: propaga los cambios locales al repositorio.

*git pull origin*: Actualiza la versión del código.

*git status*: muestra el estado de los archivos.

#### 6.3.4. Desarrollo iterativo incremental

El desarrollo de las funcionalidades se hace de forma progresiva, de modo que primero se implementan las más críticas y prioritarias en primer lugar para poder tener siempre un producto funcional.

#### 6.3.5. Revisiones periódicas del código

El código es revisado tras cada interacción de desarrollo, con el fin de mantenerlo lo más pulcro y estructurado posible. De este modo se evitan repeticiones en el código o dejar partes incompletas. Así se proporciona un mayor nivel de calidad a código producido.

#### 6.3.6. Documentación del código

El código ha sido documentado, de modo que es más mantenible por terceros y por el mismo autor. Cada funcionalidad ha sido detallada debidamente, de manera clara y concisa, sin extender demasiado la explicación.

## 6.4. Estructura

### 6.4.1. FrontEnd

Sigue el modelo que recomienda AngularJS, como antes se cita, se basa en el modelo MVC. Los distintos tipos de archivos que formarán las *vistas* (en distintas ubicaciones según la forma de Django) son:

- HTML (dentro del directorio *templates*) será la plantilla visible del sistema.
- CSS: (en el directorio *static/assets/css*) estilos que usará HTML.
- JavaScript (dentro de *static/assets/js*) mediante jQuery y AngularJS permitirán interactuar con el sistema.
- Las imágenes y distintas librerías que se usen también se ubicarán en *static/assets*.

### 6.4.2. BackEnd

Se hará uso de Django para crearlo y gestionarlo. A continuación se citan sus partes importantes que se usan en el proyecto:

- **manage.py**: archivo para gestionar el proyecto (inspeccionarlo en busca de problemas, lanzar el servidor o cargar datos).
- **settings.py**: configuraciones del proyecto (dirección de algunos directorios o carga de módulos con distintas funcionalidades)

Dentro del directorio *swgpu* (que será un *package python*):

- **urls.py**: URL declaradas dentro del proyecto *swgpu*.
- **views.py**: para gestionar las vistas y las funcionalidades del proyecto.
- La parte de FrontEnd antes descrita se situará en los directorios *templates* y *static*. Django hará uso de los distintos archivos dentro para lanzar y trabajar con el servicio web.

### 6.4.3. Ejecutable a usar por el BackEnd

El sistema hará uso de la GPU del servidor (BackEnd) mediante el ejecutable *geneticAlgorithm* (ubicado en el directorio *bin*), escrito en C++ y CUDA. De esta manera se facilitará el servicio, ya que al tratarse de un ejecutable al que sólo hay que realizar una petición con los parámetros deseados se cubren los posibles fallos y la salida que dará.

En la siguiente imagen se ve una captura de petición de ayuda sobre las variables a introducir mediante el ejecutable:

```
jcrisobal@jcrisobal-Aspire-V3-572G:~/Escritorio/geneticAlgorithm$ ./geneticAlgorithm ?
Usage -device=n (n >= 0 for deviceID)
Values to the genetic algorithm:
- -max_gen= maximum number of generations (100 by default)
- -min= minimum individual value
- -max= maximum individual value
- -p_mutation= probability of mutation (0.15 by default)
- -population_size= population size (50 by default)
- -p_crossover= probability of crossover (0.8 by default)
- -n_vars= number of problem variables (10 by default)
Values to Ackley optimization Test Function:
- -a= (20 by default) -b= (0.2 by default) -c= (2*PI by default)
  and min=-32.768 and max=32.768 by default
Values to Rastrigin optimization Test Function:
- -A_R= (10 by default)
  and min=-5.12 and max=5.12 by default
Execute Ackley by default, -rastrigin=1 to execute rastrigin function
```

Figura 6.3: Petición de un algoritmo genético mediante el ejecutable

Y otra captura con una petición que ejecutaría el algoritmo genético con los parámetros especificados además de los declarados por defecto:

```
jcrisobal@jcrisobal-Aspire-V3-572G:~/Escritorio/geneticAlgorithm$ ./geneticAlgorithm -max_gen=200 -min=-30 -max=30
{"calculo":{"nombre":"Algoritmo genético usando la función de Ackley mediante CUDA",
"dispositivo":"GPU Device 0: GeForce 840M ",
"capacidad_computo":" 5.0 ",
"info_input":{"a":"20.000000",
"b":"0.200000",
"c":"6.283185",
"n_generations":"200",
"minimal_value":"-30.0000",
"maximum_value":"30.0000",
"p_mutation":"0.150000",
"p_crossover":"0.800000",
"population_size":"50",
"n_vars":"10"},
"resultado_AG":{"generations":"200",
"values":{"0":"-16.894902",
"1":"-26.288294",
"2":"2.432316",
"3":"-26.040928",
"4":"-17.105127",
"5":"-27.949032",
```

Figura 6.4: Salida del algoritmo genético

y a continuación se ve la salida completa en formato JSON:

```

1 {
2   "calculo": {
3     "nombre": "Algoritmo genético usando la función de
4       Ackley mediante CUDA",
5     "dispositivo": "GPU Device 0: GeForce 840M ",
6     "capacidad_computo": " 5.0 ",
7     "info_input": {
8       "a": "20.000000",
9       "b": "0.200000",
10      "c": "6.283185",
11      "n_generations": "200",
12      "minimal_value": "-30.0000",
13      "maximum_value": "30.0000",
14      "p_mutation": "0.150000",
15      "p_crossover": "0.800000",
16      "population_size": "50",
17      "n_vars": "10"
18    },
19    "resultado_AG": {
20      "generations": "200",
21      "values": {
22        "0": "-16.894902",
23        "1": "-26.288294",
24        "2": "2.432316",
25        "3": "-26.040928",
26        "4": "-17.105127",
27        "5": "-27.949032",
28        "6": "25.647144",
29        "7": "-7.420090",
30        "8": "-0.034110",
31        "9": "13.586363"
32      },
33      "best_fitness": "0.000000"
34    },
35    "datos_computo": {
36      "performance": "9846.59 Flop/s",
37      "time": "50.779 msec",
38      "size": "500 ps",
39      "workgroupSize": "50 threads/block"
40    }
41  }

```

Se optó por un ejecutable porque si cada vez que se realizara una petición hubiese que compilar el programa con el algoritmo genético, comprobar que no hubiese fallos ni generase errores, y no se garantizaría una salida formateada correctamente, además de generar una petición que consume más recursos y tiempo.

Pese a todas esas desventajas, se podría permitir al usuario cambiar el código fuente, o añadir funcionalidades y con un buen uso por su parte y una correcta prevención de fallos se conseguiría una funcionalidad mayor y llena de posibilidades. Pero como esta posibilidad no se ofrece en este trabajo, se hablará en el capítulo 8 sobre ella.

## 6.5. Pruebas

A continuación se probará el rendimiento en 2 dispositivos distintos con sus respectivas especificaciones para realizar el cómputo. Con esto se verá que el sistema no depende de un modelo específico de tarjeta y presenta buenos resultados para varios modelos.

Se probará para una tarjeta GeForce 840M [44] y GeForce GTX 660 Ti [45]. Primero se verá una comparativa entre ambas, para luego ver los resultados obtenidos al realizar los mismo cálculos.

	GeForce 840M	GeForce GTX 660 Ti
Velocidad Núcleo	<b>1029 MHz</b>	835 MHz
Calidad resolución	4,12	<b>4,56</b>
Memoria GPU	<b>2048 MB</b>	<b>2048 MB</b>
Velocidad memoria	<b>1001 MHz</b>	<b>1000 MHz</b>
Bus de memoria	64 Bit	<b>128 Bit</b>
Tipo de memoria	DDR3	<b>GDDR5</b>
Tecnología de <i>GPU display</i>	<b>28nm</b>	<b>28nm</b>
Unidades de mapeado de textura	24	<b>32</b>
Ratio de textura	24,7 GTexel/s	<b>26,7 GTexel/s</b>
Unidades de salida de renderizado	8	<b>16</b>
Ratio de pixeles	8,2 GPixel/s	<b>13,4 GPixel/s</b>
Soporte CUDA	<b>SI</b>	<b>SI</b>
Capacidad CUDA	<b>5,0</b>	3,0

Figura 6.5: Especificaciones de las tarjetas 840M y GTX660

Como nota hay que destacar el parámetro de *capability* (capacidad), que será el que mayores restricciones o mejoras imponga en sus distintas versiones. Cuanto mayor sea esta, más funcionalidades podrá realizar y soportará más requisitos técnicos.

En la siguiente gráfica (Figura 6.6) se verá la diferencia de ambas en benchmarks para medir procesamiento de gráficos (Parallax, MRender, Gravity y Splatting):

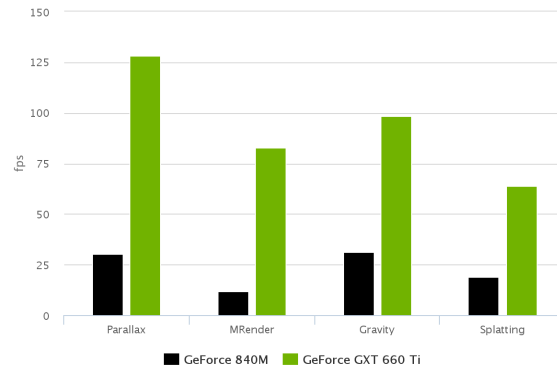


Figura 6.6: Comparativa de las tarjetas mediante benchmarks de GPU

Pero en el ámbito de la computación mediante GPU con CUDA tendremos que fijarse en la capacidad de cada modelo de tarjeta, ya que una versión más actualizada puede lograr mejores resultados aunque tenga peores especificaciones [14].

Para ello se verán los tiempos que emplean ambas tarjetas (con distintas capacidades [9]) para varios números de generaciones:



Figura 6.7: Comparativa de tiempos entre tarjetas

Y se ve que la tarjeta con mejores prestaciones pero peor capacidad (GeForce GTX 660 Ti) tarda más en ejecutar los mismos procesamientos

que hace la tarjeta con peores prestaciones pero mejor capacidad (GeForce 840M).

Como conclusión se llega a que a la hora de escoger el dispositivo a usar por el sistema para realizar los cálculos no sólo habrá que tener en cuenta los requisitos, si no que el factor de capacidad será decisivo para el rendimiento del cálculo y con ello del sistema.

También que ambas producen buenos resultados en el sistema, por lo que, se podría usar cualquiera en el sistema.





# Capítulo 7

## Manuales

### 7.1. Manual de usuario

El cliente o usuario tendrá una primera impresión de la *portada* del servicio web. Simplemente se le mostrará el logo y título del servicio y podrá avanzar hacia las funcionalidades mediante un botón, o haciendo *scroll* hacia abajo.



Figura 7.1: Captura de la portada del servicio web

Una vez avance podrá ver los servicios disponibles: ejecutar un algoritmo genético y que use la función de evaluación de Ackley o de Rastrigin. Para acceder a dichos servicios sólo tendrá que *clickar* sobre el servicio que quiera y se desplegará.

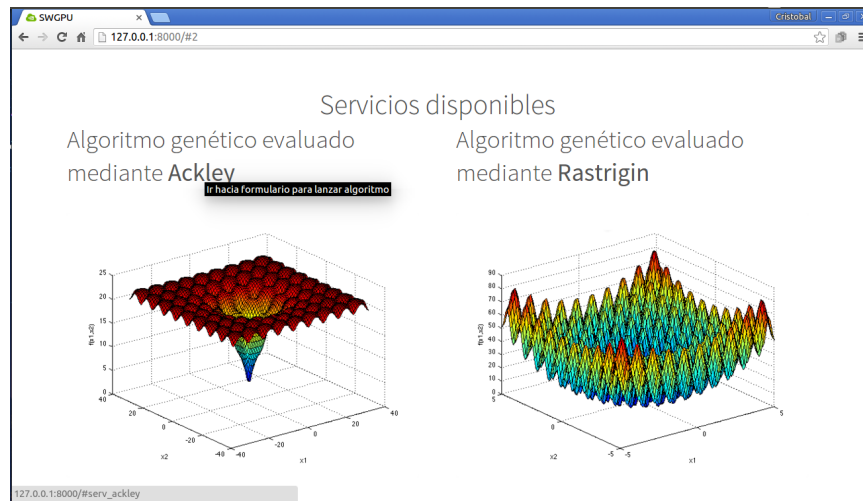


Figura 7.2: Captura de los servicios disponibles

Una vez elegido el servicio se desplegará su correspondiente formulario. Para acceder a él el usuario sólo tendrá que *clickar* sobre el título en la pantalla de *Servicios disponibles* o avanzar hacia el mediante *scroll*. Una vez en el formulario se le mostrarán los valores por defecto de cada parámetro, pudiendo cambiar cada uno de manera sencilla:

Algoritmo genético evaluado mediante Ackley		
Tamaño de la población	Número de generaciones	Valores para la función de optimización Ackley
50	100	
Tamaño del cromosoma	Probabilidad de mutación	
5	0,15	
Valor mínimo	Probabilidad de cruce	
-32,768	0,8	Valor para el parámetro a
32,768		20
		Valor para el parámetro b
		0,2
		Valor para el parámetro c
		6,283185307179586
REALIZAR ALGORITMO GENÉTICO		

Figura 7.3: Formulario para lanzar el algoritmo genético

También se asegurará que se introducen los valores correctamente:

Tamaño del cromosoma  
5

Probabilidad de mutación  
0,15

Valor mínimo  
35,25

Probabilidad de cruce  
0,8

Valor máximo  
32,768

Valor máximo debe ser mayor que el mínimo

REALIZAR ALGORITMO GENÉTICO

Figura 7.4: Mensaje de error en el formulario

Y una vez realizado el algoritmo genético, ya se podrá acceder a los resultados. Para ello sólo hay que avanzar hacia abajo o mediante el botón de *Ver resultados*. Se mostrará información sobre la petición (hora y parámetros), datos sobre el cómputo (dispositivo usado o tiempo invertido) y los resultados obtenidos.

Resultados obtenidos

Fecha de petición: Thu Sep 08 2016 12:11:37 GMT+0200 (CEST)

Algoritmo genético usando la función de Ackley mediante CUDA

Dispositivo: GeForce 840M  
Rendimiento: 7265.43 Flop/s  
Tiempo: 34.410 msec

Tras 100 generaciones con el mejor 'fitness' de 0.000000 se obtienen como resultado:

Variable	Valor
0	-18.453739
1	-28.713828
2	2.656738
3	-28.443639
4	-18.683361

**Para la entrada:**  
Número de generaciones: 100  
Valor mínimo: -32.768002  
Valor máximo: 32.768002  
Para 5 variables (tamaño del cromosoma)  
Probabilidad de mutación: 15%  
Probabilidad de cruce: 80%  
Y una población de 50 individuos  
Con las variables propias de la función de optimización *Ackley*:  
A: 20.000000 B: 0.200000 C: 6.283185

Figura 7.5: Resultados obtenidos

Por último se muestra al final (aunque se puede acceder siempre a él) una sección de contacto. Se podrá poner en contacto con el administrador del sistema.

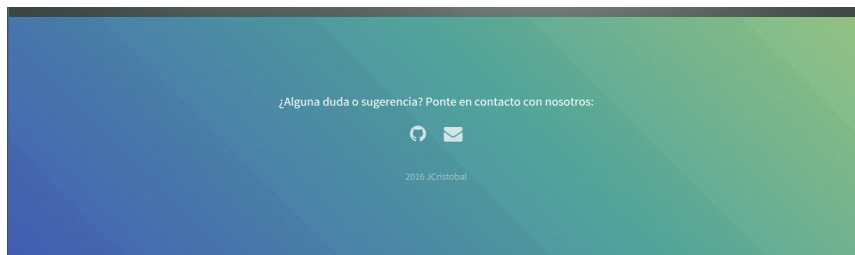


Figura 7.6: Sección de contacto

Además, como añadido, se facilita la ayuda sobre parámetros y ejecución del ejecutable *geneticAlgorithm* como parte independiente, sin requerir del servicio web, como se puede ver en la Figura 6.3 del capítulo anterior.

## 7.2. Manual de despliegue

A continuación se explicará la forma y requisitos de despliegue del sistema.

### 7.2.1. Requisitos para el despliegue

Para el correcto despliegue de la aplicación habrá que contar con sus 2 partes: el servicio web y el procesamiento con GPU mediante CUDA.

Para lanzar el servicio web:

- Apache (versión 2.4 o superior)
- Python (Python 2)
- Django (versión 1.10 o superior)

Y para el procesamiento mediante CUDA:

- Dispositivo NVIDIA
- Drivers NVIDIA (versión 352.xx)
- CUDA 7.5
- C++ (versión 4.x)

### 7.2.2. Despliegue

Una vez que se cumplen los requisitos, sólo habrá que lanzar el servicio web, ya que el procesamiento usando la GPU se gestiona mediante el ejecutable *geneticAlgorithm* dirigido por el servicio web.

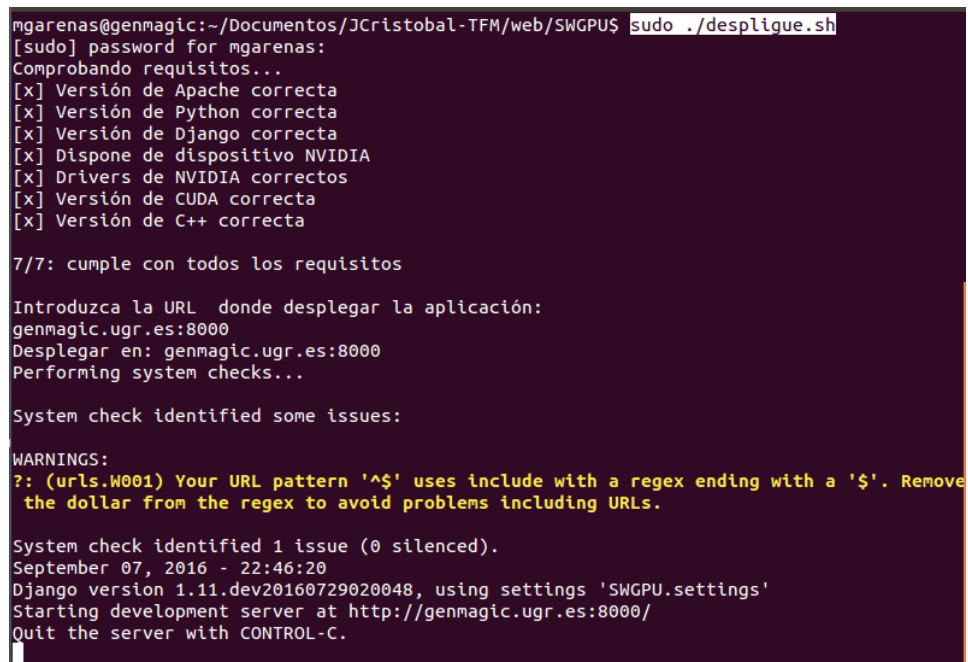
Para ello el servidor Apache tendrá que estar activo (se puede activar ejecutando *service apache2 restart*) y lanzar el proyecto Django con el proyecto: dentro del directorio SWGPU ejecutar, junto con la URL donde lanzar el sistema *python manage.py runserver [URL]*

### 7.2.3. Despliegue automatizado

También se proporcionará el script o archivo *despliegue.sh* para automatizar la tarea de despliegue.

Simplemente habrá que ejecutarlo (dentro del *directorio SWGPU*, mediante la orden **sudo ./despliegue.sh**) y luego especificar la URL donde se desplegará. El script comprobará los requisitos antes citados, preparará el servidor y lanzará el sistema.

En la siguiente captura se ve como se lanza el servidor:

A terminal window with a dark background and light-colored text. The prompt is 'mgarenas@genmagic:~/Documentos/JCristobal-TFM/web/SWGPU\$'. The user enters 'sudo ./despliegue.sh'. The terminal shows a password prompt, then a series of checks for various requirements (Apache, Python, Django, NVIDIA device/drivers, CUDA, C++), all of which are marked as correct with '[x]'. It then asks for a URL, and the user enters 'genmagic.ugr.es:8000'. The script performs system checks, identifies a warning about a regex pattern in the URL, and then starts the Django development server at the specified URL. The terminal output is as follows:

```
mgarenas@genmagic:~/Documentos/JCristobal-TFM/web/SWGPU$ sudo ./despliegue.sh
[sudo] password for mgarenas:
Comprobando requisitos...
[x] Versión de Apache correcta
[x] Versión de Python correcta
[x] Versión de Django correcta
[x] Dispone de dispositivo NVIDIA
[x] Drivers de NVIDIA correctos
[x] Versión de CUDA correcta
[x] Versión de C++ correcta

7/7: cumple con todos los requisitos

Introduzca la URL donde desplegar la aplicación:
genmagic.ugr.es:8000
Desplegar en: genmagic.ugr.es:8000
Performing system checks...

System check identified some issues:

WARNINGS:
?: (urls.W001) Your URL pattern '^$' uses include with a regex ending with a '$'. Remove
the dollar from the regex to avoid problems including URLs.

System check identified 1 issue (0 silenced).
September 07, 2016 - 22:46:20
Django version 1.11.dev20160729020048, using settings 'SWGPU.settings'
Starting development server at http://genmagic.ugr.es:8000/
Quit the server with CONTROL-C.
```

Figura 7.7: Captura del despliegue automático

## Capítulo 8

# Conclusiones y trabajos futuros

### 8.1. Conclusiones

Una vez realizado el trabajo, se verán los objetivos (citados en el capítulo 2) y su respectivo avance.

- *Desarrollar el conjunto de clases que permita acceder de forma remota a un conjunto de recursos para la ejecución de algoritmos evolutivos paralelos.*

Se han logrado cumplir, además reuniéndolas en el archivo ejecutable *geneticAlgorithm*. Se ha desarrollado como un módulo independiente, para poder ser usado por cualquier servicio, y no ser dependiente del servicio web, controlando la entrada de parámetros y control de fallos.

- *Acceder a un recurso de computación remoto a través de un interfaz web.*

También se ha creado un servicio web, de manera que sea accesible desde cualquier servidor web. Dicho servicio facilita la petición del algoritmo genético, y se obtendrán los resultados mediante el uso de la GPU del servidor.

- *Aprender a utilizar recursos de computación paralela de bajo coste como son las GPUs.*

Para lograr este objetivo se ha analizado, probado y desarrollado mediante CUDA, aprovechando la computación paralela de una GPU y aprendiendo a usar y gestionar sus recursos.

- *Estudiar el rendimiento obtenido para esta interfaz web en comparación con otras alternativas.*

El servicio se ha desplegado y probado en el servicio web, generando resultados sin necesidad de instalar software ni limitaciones técnicas, ventaja principal ante otras alternativas. Con la interfaz de usuario se logra una buena experiencia de usuario, y una mejora en el rendimiento, ya que se optimiza el uso de recursos mediante la GPU.

Se a publicado el servicio, en forma de *beta*. Se encuentra accesible en [www.genmagic.ugr.es:8000](http://www.genmagic.ugr.es:8000), accesible dentro de la red de la UGR [10].

## 8.2. Trabajos futuros

Además de mantener y gestionar la versión actual del servicio web, se podría trabajar en la mejora de la experiencia de usuario.

También se refinará el algoritmo, logrando un mejor aprovechamiento de los recursos y mejorando los resultados.

Como se comenta en el capítulo 6 (en la subsección *Ejecutable a usar por el BackEnd*) se estudiará la posibilidad de que el usuario pudiera añadir funcionalidades o cualquier procesamiento usando la GPU mediante CUDA.

De esta manera se podría ejecutar cualquier opción de optimización, o fuera del ámbito de los algoritmos genéticos cualquier funcionalidad que requiera el usuario.

Esto requeriría un conocimiento mayor por parte del usuario. Además supondría un mayor consumo de recursos, pero con un buen uso del usuario y una correcta prevención de fallos se conseguiría una funcionalidad mayor y llena de posibilidades.



# Bibliografía

- [1] Algoritmo genético en Python. <http://robologs.net/2015/09/01/como-programar-un-algoritmo-genetico-parte-ii-implementacion-en-python/>, (Acceso en Abril 2016)
- [2] Alojamiento web DonDominio. <https://www.dondominio.com/>, (Acceso en Agosto 2016)
- [3] Alojamiento web HostPapa. <https://www.hostpapa.es/>, (Acceso en Agosto 2016)
- [4] AMD OpenCL Accelerated Parallel Processing. <http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/>, (Acceso en Agosto 2016)
- [5] AngularJS. <https://angularjs.org/>, (Acceso en Mayo 2016)
- [6] Apache. <https://httpd.apache.org/>, (Acceso en Mayo 2016)
- [7] BrookGPU. <http://graphics.stanford.edu/projects/brookgpu/>, (Acceso en Abril 2016)
- [8] C+++. <https://es.wikipedia.org/wiki/C%2B%2B>, (Acceso en Mayo 2016)
- [9] Capacidades de tarjetas NVIDIA. <https://developer.nvidia.com/cuda-gpus>, (Acceso en Mayo 2016)
- [10] Conexión a la VPN de la UGR. <https://csirc.ugr.es/informatica/RedUGR/VPN/ConfVPN/>, (Acceso en Agosto 2016)
- [11] CSS3. <https://www.w3.org/TR/CSS/>, (Acceso en Mayo 2016)

- [12] Django.  
<https://www.djangoproject.com/>, (Acceso en Mayo 2016)
- [13] Draw.io.  
<https://www.draw.io/>, (Acceso en Mayo 2016)
- [14] Especificación de las capacidades de CUDA.  
[https://en.wikipedia.org/wiki/CUDA#Version\\_features\\_and\\_specifications](https://en.wikipedia.org/wiki/CUDA#Version_features_and_specifications), (Acceso en Mayo 2016)
- [15] Framework rCUDA para ejecutar programas CUDA en un servidor externo.  
<http://www.rcuda.net/index.php/what-s-rcuda.html>, (Acceso en Abril 2016)
- [16] GanttPRO.  
<https://app.ganttpro.com>, (Acceso en Mayo 2016)
- [17] Gimp.  
<https://www.gimp.org/>, (Acceso en Mayo 2016)
- [18] Git.  
<https://git-scm.com/>, (Acceso en Mayo 2016)
- [19] GitHub.  
<https://github.com/>, (Acceso en Mayo 2016)
- [20] 'Global Optimization Toolbox' para MatLab.  
<http://es.mathworks.com/products/global-optimization/>, (Acceso en Abril 2016)
- [21] GPGPU.  
<https://es.wikipedia.org/wiki/GPGPU>, (Acceso en Abril 2016)
- [22] gpuocelot: Framework dinámico de compilación que soporta CUDA.  
<https://code.google.com/archive/p/gpuocelot/>, (Acceso en Abril 2016)
- [23] HTML5.  
<https://www.w3.org/TR/html5/>, (Acceso en Mayo 2016)
- [24] HTML5UP.  
<https://html5up.net/>, (Acceso en Mayo 2016)
- [25] Impact: CUDA framework.  
<http://impact.crhc.illinois.edu/mcuda.aspx>, (Acceso en Abril 2016)

- [26] Instancia en Amazon Web Services que permita usar la GPU. <https://aws.amazon.com/blogs/aws/new-ec2-instance-type-the-cluster-gpu-instance/>, (Acceso en Julio 2016)
- [27] JavaScript. <https://www.javascript.com/>, (Acceso en Mayo 2016)
- [28] JGAP: Framework java para realizar algoritmos genéticos. <http://jgap.sourceforge.net/>, (Acceso en Abril 2016)
- [29] jQuery. <http://jquery.com/>, (Acceso en Mayo 2016)
- [30] JSON. <http://www.json.org/>, (Acceso en Mayo 2016)
- [31] Medias de sueldo para programadores Junior. <http://espana.jobtonic.es/salary/26526/16258.html?currency=EUR>, (Acceso en Agosto 2016)
- [32] Nsight. <http://www.nvidia.es/object/nsight-es.html>, (Acceso en Abril 2016)
- [33] NVIDIA Developer. <https://developer.nvidia.com/>, (Acceso en Abril 2016)
- [34] Ordenador Acer Aspire V3 572G. [https://www.pccomponentes.com/acer-aspire-v3-572g-intel-i7-4510u-8gb-1tb-gt840m-15-gclid=CjwKEAjwmMS-BRCm5dn51JLbp1wSJACc61tFa9QMUK\\_O\\_7taFqbiPPt6PmLIJZG5pXgfePoyueXTEhoCfsvw\\_wcB](https://www.pccomponentes.com/acer-aspire-v3-572g-intel-i7-4510u-8gb-1tb-gt840m-15-gclid=CjwKEAjwmMS-BRCm5dn51JLbp1wSJACc61tFa9QMUK_O_7taFqbiPPt6PmLIJZG5pXgfePoyueXTEhoCfsvw_wcB), (Acceso en Agosto 2016)
- [35] PeackStream. <https://en.wikipedia.org/wiki/PeakStream>, (Acceso en Abril 2016)
- [36] PeackStream comprado por Google. <https://www.crunchbase.com/organization/peakstream#/entity>, (Acceso en Abril 2016)
- [37] Procesamiento paralelo mediante CUDA. <http://www.nvidia.es/object/cuda-parallel-computing-es.html>, (Acceso en Abril 2016)
- [38] Python. <https://www.python.org/>, (Acceso en Mayo 2016)

- [39] RapidMind.  
<https://en.wikipedia.org/wiki/RapidMind>, (Acceso en Abril 2016)
- [40] RapidMind es comprada por Intel.  
<https://software.intel.com/en-us/articles/intel-array-building-blocks>, (Acceso en Abril 2016)
- [41] Simulación de algoritmo genético del aprendizaje de un robot a andar.  
[http://rednuht.org/genetic\\_walkers/](http://rednuht.org/genetic_walkers/), (Acceso en Mayo 2016)
- [42] Simulación de algoritmo genético en la creación de la forma de un vehículo.  
<http://gencar.co/>, (Acceso en Mayo 2016)
- [43] Sublime Text.  
<https://www.sublimetext.com/>, (Acceso en Mayo 2016)
- [44] Tarjeta gráfica NVIDIA GeForce 840M.  
<http://www.geforce.com/hardware/notebook-gpus/geforce-840m/specifications>, (Acceso en Agosto 2016)
- [45] Tarjeta gráfica NVIDIA GeForce GTX 660 Ti.  
<http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-660ti/specifications>, (Acceso en Agosto 2016)
- [46] Trabajo sobre la paralelización de algoritmos basados en poblaciones por medio de GPGPU.  
<http://posgrado.itlp.edu.mx/uploads/4f335b50b576b.pdf>, (Acceso en Mayo 2016)
- [47] Cerutti, F., B.L.G.T.L..G.M.: Taxon ordering in phylogenetic trees by means of evolutionary algorithms. *BioData Mining*, 4, 20. (2011), (Acceso en Mayo 2016)
- [48] Iliana Castro Liera, Marco Antonio Castro Liera, J.A.C.: Optimización en paralelo basada en poblaciones usando GPGPU pp. 24–28 (Enero 2012), (Acceso en Julio 2016)
- [49] Miller, J.H.: The coevolution of automata in the repeated Prisoner's Dilemma. pp. 87–112 (Enero 1996), (Acceso en Mayo 2016)
- [50] Ooi1, C.H., Tan2, P.: Genetic algorithms applied to multi-class prediction for the analysis of gene expression data . pp. 37–44 (Enero 2003), (Acceso en Mayo 2016)
- [51] Savic, D., Walters, G.: Genetic Algorithms for Least-Cost Design of Water Distribution Networks. (Marzo 1997), (Acceso en Mayo 2016)

- 
- [52] Shimoyama, K., Seo, K., Nishiwaki, T., Jeong, S., Obayashi, S.: Material design optimization for a sport shoe sole by evolutionary computation and FEM analysis. In: IEEE Congress on Evolutionary Computation. pp. 1–7 (July 2010), (Acceso en Mayo 2016)