

Universidad Tecnológica de La Habana “José Antonio Echeverría”

Facultad de Ingeniería Informática, Filial de Ciencias Técnicas “Diez de Octubre”



ADHOS v1.0

“Herramienta de análisis para las historias demográficas de las especies”

Trabajo de diploma para optar por el título de Ingeniería en Informática

Autor: Héctor Armando Rizo Blanco

Tutor(es):

- **Dr.** Willy Rodríguez Valcarce (Investigador en el Instituto de Matemáticas de Toulouse, Francia)

Email: willyrv@gmail.com

- **Ing.** Jorge Herrera Franklin (Profesor Asistente e Investigador en el Centro de Investigación y Manejo Ambiental del Transporte)

Email: jherrera@ceis.cujae.edu.cu

La Habana. Junio, 2019

Agradecimientos

Hace alrededor de 6 años reinicié mis estudios universitarios con el deseo de un día cumplir mi sueño, sueño que vestía de la incertidumbre de poder llegar al final. Hoy que todo se ha materializado es preciso dar las gracias a las personas que, de alguna forma u otra, y a pesar de las dificultades, estuvieron siempre brindando su apoyo incondicional.

Por tanto:

- *Agradezco a DIOS por haber permitido que sobrepasara con mucha FÉ, momentos difíciles de mi vida vinculada con mi salud, inexperiencia e inmadurez.*
- *A mi padre de manera especial por su dedicación e inmenso amor; el cual una vez me dijo:*

“Hijo, persiste y confía; yo no se si podré estar toda tu vida para apoyarte, sólo se con seguridad que si estaré toda la mía para amarte y ayudarte”

- *A mi amada esposa y mis queridos hijos Ale y Piki por su amor y dedicación.*
- *A mi querida madre, abuelo y familia por todo su apoyo, comprensión y buenos consejos, sobre todo a quienes me inculcaron que en la persistencia y perseverancia está la fórmula del triunfo.*
- *A todos aquellos que me ayudaron a tomar riesgos y a hacer de los temores oportunidades para crecer.*

- *A los profesores y trabajadores de esta sede universitaria, con especial atención a mi tutor y hermano; Jorge Herrera Franklin.*
- *A mi querida madrastra Reumar y mi tía Daya sin las cuales todo hubiese sido más difícil.*
- *Y especialmente a Willy Rodríguez Valcarce; mi segundo tutor, hermano de los años, amigo y compañero de aula en el IPVCE Vladimir Ilich Lenin, hoy profesor e investigador del Instituto de Matemáticas Aplicadas de Toulouse, Francia, quien confió y me dio la oportunidad de desarrollar la aplicación que representa hoy mi tema de tesis.*

¡A todos, mil gracias!

Resumen

La genética de poblaciones es la rama de la Biología que estudia la variación de la diversidad genética a lo largo del tiempo. Uno de los problemas centrales de la genética de poblaciones es la reconstrucción de la historia demográfica de una especie. Actualmente con este objetivo se utilizan diversos programas, siendo unos de los más populares los softwares PSMC (del inglés Pairwise Sequentially Markovian Coalescent) y MSMC (del inglés The Multiple Sequentially Markovian Coalescent). Los resultados obtenidos por estos softwares son interpretados en base a modelos teóricos bien establecidos y aceptados en genética de poblaciones. No obstante, se conoce que dichos modelos asumen determinadas hipótesis que pudieran no estar en correspondencia con la realidad. Estudios recientes han demostrado que la interpretación de los resultados puede ser completamente incorrecta cuando el modelo no se ajusta a la realidad de la especie que se quiere analizar. La aplicación que se describe en este trabajo permitirá interpretar los resultados utilizando un modelo más general de evolución de la población. Se enfocará principalmente en la utilización y comparación de algoritmos metaheurísticos enmarcados en un problema de ajuste de curvas. Se adoptó Scrum como metodología de desarrollo, Python y Javascript como lenguajes de programación y Git como sistema de control de versiones.

Palabras claves: genética de poblaciones, estimación de historia demográfica, NSSC, PSMC, MSMC, metaheurísticas.

Abstract

Population genetics is the branch of Biology that studies the variation of genetic diversity over time. One of the central problems of population genetics is the reconstruction of the demographic history of a species. Currently with this objective various programs are used, being one of the most popular PSMC (Pairwise Sequentially Markovian Coalescent) and MSMC (The Multiple Sequentially Markovian Coalescent) softwares. The results obtained by these softwares are interpreted based on well-established theoretical models accepted in population genetics. However, it is known that these models assume certain hypotheses that may not be in correspondence with reality. Recent studies have shown that the interpretation of the results can be completely incorrect when the model does not conform to the reality of the species to be analyzed. The application described in this work will allow us to interpret the results using a more general model of population evolution. It will focus mainly on the use and comparison of metaheuristic algorithms framed in a problem of curve adjustment. Scrum was adopted as a development methodology, Python and Javascript as programming languages and Git as a version control system.

Keywords: population genetics, demographic history estimation, NSSC, PSMC, MSMC, metaheuristics.

Índice

Introducción	1
Capítulo I. Análisis del proceso	7
1.1 Introducción	7
1.2 Descripción del proceso actual de interpretación de historias demográficas	7
1.3 Diagrama del proceso actual	8
1.4 Análisis crítico del proceso actual	9
1.5 Análisis y optimización del proceso actual	9
1.5.1 Diagrama de Ishikawa	9
1.5.2 Procesos objetos de automatización	10
1.6 Tendencia y tecnologías actuales	11
1.6.1 Identificación de las variables deseadas para la solución	11
1.6.2 Características de las soluciones informáticas estudiadas	13
1.7 Análisis crítico de las fuentes bibliográficas utilizadas	22
1.8 Conclusiones parciales del capítulo	23
Capítulo II: Especificaciones de funcionalidad	24
2.1 Introducción	24
2.2 Definición de interesados	24
2.3 Definición de Temas	25
2.4 Pila del producto	26
2.4.1 Tema 1: Procesamiento de datos	26
2.4.2 Tema 2: Manipulación de datos	28

2.4.3 Tema 3: Almacenamiento de información	31
2.4.4 Tema 4: Configuración	32
2.5 Tipos de pruebas ejecutadas	34
2.5.1 Diseño de los casos de pruebas unitarias	36
2.6 Requisitos no funcionales	38
2.7 Conclusiones parciales del capítulo	41
Capítulo III: Descripción de la solución propuesta	42
3.1 Introducción	42
3.2 Definición del problema de optimización	42
3.2.1 Optimización	42
3.2.2 Definición del problema formal de optimización	43
3.3 Cálculo de la distribución de los tiempos de coalescencia bajo el modelo NSSC	45
3.4 Resolución del problema computacionalmente	46
3.4.1 Metaheurísticas	46
3.4.2 Selección de las metaheurísticas	47
3.5 Arquitectura	50
3.6 Diagramas de Clases de Diseño	52
3.7 Principios de diseño	57
3.7.1 Interfaz de usuario	59
3.8 Tratamiento de errores	60
3.9 Análisis de costos y beneficios de la solución propuesta	61
3.9.1 Costos	62
3.9.2 Beneficios	62

3.10 Conclusiones parciales del capítulo	64
Conclusiones	65
Recomendaciones	67
Referencias bibliográficas	68
Glosario de términos y siglas	72

Índice de figuras

Diagrama del proceso actual	8
Diagrama Ishikawa para el análisis y optimización	10
Actualidad de descargas de Electron	15
Gestor de procesos de negocios BPM (Bonita Studio)	20
Diagrama de Ishikawa	21
Diseño de caso de prueba unitaria (1) y resultados	37
Diseño de caso de prueba unitaria (2) y resultados	38
Arquitectura Modelo - Vista – Controlador	51
Estructuración en capas usando un enfoque de responsabilidades	52
Diagrama de clases de diseño de cargar ficheros	53
Diagrama de clases de la aplicación	53
Clase Utilities_Application	54
Clase Python_Communicator y paquete de scripts de python	54
Clase Visual_Application	55
Clase Logic_Application	56
Clases Models	56
Función Visualize_Communic_Funcion().....	57
Función Visualize_PSMC().....	58
Función Visualize_MSMC().....	58
Función Visualize_NSSC().....	58
Función Visualize_Application().....	58
Ventana principal de la aplicación	60
Información lanzada al intentar cargar un archivo existente	61

Error lanzado al intentar cargar un archivo no válido	61
---	----

Índice de tablas

Variables para cumplir con la solución deseada	11
Interesados del proyecto	25
Temas del proyecto	25
Historias de Usuarios del tema procesamiento de datos	26
Historias de Usuarios del tema manipulación de datos	28
Historias de Usuarios del tema almacenamiento de información	31
Historias de Usuarios del tema configuración	32
Diseño de caso de prueba unitaria (1)	37
Diseño de caso de prueba unitaria (2)	37
Prueba del algoritmo Escalador de colinas	48
Prueba del algoritmo Recocido simulado	49
Prueba del algoritmo Evolución diferencial	49
Prueba del algoritmo Optimización por enjambre de partículas	49

Introducción

La genética es el área de estudio de la biología que busca comprender y explicar cómo se transmite la herencia biológica de generación en generación [1]. Sus principios teóricos son suministrados por la genética de poblaciones¹, donde el conocimiento obtenido es aplicable a la macroevolución [2]. Esto no sería más que la extrapolación en el espacio y en el tiempo de los procesos básicos que se dan en las poblaciones.

La detección de los eventos demográficos más importantes ocurridos en el pasado puede mejorar nuestro conocimiento sobre la evolución [3]. Para esto es necesario realizar reconstrucciones de historias demográficas, las cuales no constituyen un proceso trivial y son hoy en día objeto de estudio de muchos científicos en el campo de la biología. La teoría de la coalescencia [4] [5], surgida en parte por la necesidad de estimar parámetros del pasado usando muestras de poblaciones actuales, constituye el avance más importante de la genética de poblaciones de las últimas décadas. La desaparición de bosques o los cambios climáticos pueden reducir drásticamente el número de representantes de una especie en determinada época. Esto trae como consecuencia una reducción en la diversidad que se refleja en una disminución de diversidad genética. Se conoce que este tipo de fenómenos genera ciertos patrones en el *ácido desoxirribonucleico* (ADN en lo adelante) de los individuos y es posible reconstruir los principales cambios en la demografía de la especie. Dichos métodos estadísticos de reconstrucción de historia demográfica están basados en modelos matemáticos que incluyen ciertas hipótesis.

¹ Es el estudio de las fuerzas que alteran la composición genética de una especie. Se ocupa de los mecanismos de cambio microevolutivo: mutación, selección natural, flujo génico y deriva génica.

Hoy en día se dispone de enormes cantidades de datos genéticos obtenidos por medio de proyectos de secuenciación² a gran escala (por ejemplo, el Proyecto Genoma Humano, PGH en lo adelante) [6]. Para analizar este gran volumen de datos es necesario desarrollar algoritmos que permitan un uso eficiente de los modelos matemáticos existentes. Dichos modelos son de gran importancia pues describen la relación entre la historia de las especies y ciertos patrones presentes en las secuencias de ADN de los individuos.

El creciente desarrollo en las tecnologías de secuenciación de nueva generación (NGS, del inglés Next Generation Sequencing) han revolucionado la forma de analizar la historia de las especies [7]. Esto junto a la creación de herramientas de análisis de datos generadas en el PGH, ha posibilitado el desarrollo de diversos métodos y modelos que permiten estimar el tamaño efectivo, el cual nos brinda la posibilidad de cuantificar de cierta forma la diversidad genética de una población. Este desarrollo facilitará y hará posible definir los temas de estudio futuros con vistas a las tareas pendientes. Entre las tecnologías beneficiadas gracias al PGH figuran las de manejo computacional de datos [6], las que permiten, por ejemplo, la reconstrucción de historias demográficas de las distintas especies. Los softwares PSMC y MSMC [8][9] utilizan un método basado en un *Modelo Oculto de Markov*³ (HMM en lo adelante) y en la teoría de la coalescencia. Estos permiten reconstruir la historia demográfica a lo largo del tiempo a partir del ADN completo de un solo individuo, o sea, identifica los períodos en que dicha especie contó con un mayor o menor número de representantes. Los resultados obtenidos por el PSMC y el MSMC se interpretan en base a modelos teóricos que poseen gran aceptación en genética de poblaciones [11]. Sin embargo, parten de hipótesis que no siempre están en correspondencia con la realidad de la

² Es un conjunto de métodos y técnicas bioquímicas cuya finalidad es la determinación del orden de los nucleótidos (A, C, G y T) en un oligonucleótido de ADN.

³ Es un modelo estadístico en el que se asume que el sistema a modelar es un proceso de Markov de parámetros desconocidos.

especie que se desea analizar [10], además de no tener en cuenta aspectos de gran importancia, ejemplo; *la estructura de la población* [11].

Investigaciones recientes [12] han demostrado que las poblaciones estructuradas generan señales de cambio en el tamaño de la población. Debido a que existen pocos métodos inferenciales que permiten dar cuenta de esa estructura, y que los datos genéticos aumentan la precisión de las estimaciones de los parámetros, se hizo necesario el desarrollo de nuevos enfoques.

Por tanto, estudios matemáticos [13] [14] han propuesto un nuevo modelo (NSSC) que considera la estructura y cambios en el tamaño de la población, así como en el *flujo de genes*⁴, lo cual, si bien constituye un aporte a la ciencia moderna, aún tiene limitaciones que generan la situación problemática descrita a continuación.

Hasta el momento es imposible realizar comparaciones visuales entre los gráficos obtenidos por los softwares PSMC y MSMC. Tampoco existe un proceso ágil que grafique el modelo NSSC bajo ciertos parámetros y permita su configuración con el objetivo de ir obteniendo nuevas curvas. Otro aspecto que debe destacarse, es la necesidad de realizar un ajuste automático de los gráficos obtenidos mediante el modelo NSSC; esto para compararlos con los obtenidos por los softwares PSMC y MSMC en vista de mejorar las interpretaciones.

A partir de la situación problemática observada anteriormente, se puede plantear el *problema a resolver* del siguiente modo: Para los investigadores resulta imposible evaluar si el modelo de población estructurada (NSSC) se ajusta más a los datos genéticos que los modelos existentes.

⁴ Significa migración

Es suficiente que el autor se sitúe sobre el escenario de la bioinformática fundamentada en análisis matemáticos, para deducir que el *objeto de estudio* del proyecto es la optimización combinatoria enmarcada en un problema de genética de poblaciones. Su *campo de acción* está enfocado en el *ajuste de curvas* (las curvas basadas en el modelo de población estructurada NSSC y las basadas en los modelos PSMC y MSMC). La aplicación debe permitir realizar dicho ajuste de manera manual y principalmente automática. Para la segunda opción, la implementación de al menos dos algoritmos metaheurísticos será crucial para determinar la eficacia y eficiencia respecto al factor tiempo y precisión.

El proyecto persigue como *objetivo general* el desarrollo de una aplicación informática para mejorar la interpretación de los resultados obtenidos a partir de los softwares de análisis de datos genéticos más populares. Para cumplir dicho objetivo general se definieron los siguientes *objetivos específicos*:

- La aplicación debe permitir ajustar manualmente las curvas obtenidas por el modelo NSSC.
- Se debe implementar (mínimo dos algoritmos metaheurísticos) enfocados al problema del ajuste de curvas.

Los objetivos específicos antes expuestos han llevado a cabo la realización de diversas *tareas* para dar cumplimiento a los mismos. Estas son:

1. Realizar un estudio detallado sobre las ventajas y desventajas del uso de las distintas tecnologías y lenguajes de programación.
2. Investigar sobre el amplio tema de los algoritmos metaheurísticos, enfocado principalmente en el problema del ajuste de curvas.

3. Definir temas para establecer las principales líneas de trabajo a la hora de desarrollar la aplicación y agrupar las características del sistema.
4. Identificar y planificar las historias de usuario o requerimientos que serán recogidas en los temas.
5. Escribir pruebas automatizadas para comprobar el correcto funcionamiento de las historias de usuario.
6. Modelar la arquitectura y las principales historias de usuario a través de diagramas de clases utilizando UML.
7. Codificar las nuevas funcionalidades de manera que pasen las pruebas automatizadas.

- Valor Práctico y estructura del trabajo

La herramienta desarrollada mejorará la interpretación de los resultados obtenidos a partir del uso de softwares de análisis de datos genéticos. Además, será de vital importancia para los investigadores a la hora de evaluar si existe algún otro escenario alternativo que pueda explicar los datos bajo un modelo de población estructurada (NSSC).

Importante destacar también que la mayoría de los usuarios que harán uso de esta herramienta no son expertos en tecnologías e informática, de ahí que el software proporcionará mediante una interfaz amigable, un fácil uso para alcanzar los resultados esperados. Una vez concluido y aprobado el proyecto, estará disponible de manera gratis en *GitHub* para los interesados en el tema, propiciando de esa manera ahorros financieros, y promoviendo el software libre que sin dudas es un paso

importante en la nueva era para que las tecnologías estén más al alcance de cualquier persona.

- CAPÍTULO I: Se enfoca en la fundamentación teórica por medio del análisis del proceso abordado. De igual manera se exponen las tecnologías que se utilizarán, así como la metodología de desarrollo a emplear.
- CAPÍTULO II: Presenta la definición de funcionalidades con el objetivo principal de documentar el proceso de construcción de la aplicación, haciendo uso de los artefactos que propone la metodología de desarrollo a emplear.
- CAPÍTULO III: Aborda el problema de optimización combinatoria relacionado con el ajuste de curvas. Por otra parte, a través de diagramas UML, ayuda a mejorar la comprensión del diseño y arquitectura del sistema.

CAPÍTULO I: Fundamentos teóricos. Análisis del Proceso

1.1 Introducción

La reconstrucción de la historia demográfica de una especie es un proceso actual llevado a cabo en los laboratorios de Evolución y Diversidad Biológica de la Universidad Paul Sabatier en conjunto con la Universidad de Toulouse y el Instituto de Matemáticas de Toulouse (Francia). Expertos de dichas universidades se encuentran en continuas investigaciones con el objetivo de mejorar los resultados de dicho proceso.

El presente capítulo aborda fundamentalmente el proceso ejecutado en el campo de acción, así como su análisis y optimización. Expone los procesos objeto de automatización y se describen las tecnologías y marcos de trabajo empleados. Finalmente se realiza un análisis crítico de las fuentes bibliográficas utilizadas.

1.2 Descripción del proceso actual de interpretación de las historias demográficas de las especies

El desarrollo de la secuenciación del ADN ha acelerado significativamente la investigación y los descubrimientos en biología. Las técnicas actuales permiten realizar esta secuenciación a gran velocidad, lo cual ha sido de gran importancia para proyectos de secuenciación a gran escala como el Proyecto Genoma Humano [6]. Los millones de fragmentos de secuencias de ADN que producen las máquinas de secuenciación se ordenan y se ensamblan con programas bioinformáticos sofisticados [7]. Una vez se dispone de la secuencia ensamblada, llega el momento de dar sentido a la misma.

Para reconstruir las historias demográficas de las especies, estos datos son procesados por los softwares más populares (PSMC y MSMC) que tienen en cuenta la mutación y la recombinación, además de considerar que el tamaño efectivo cambia en el transcurso del tiempo. Estos modelos introducen una estructura de Markov a lo largo del genoma. También se basan en la teoría de la coalescencia, que no es más que el proceso que describe el árbol genealógico de un conjunto de individuos retrocediendo en el tiempo hasta alcanzar el ancestro común más reciente o ACMR (del inglés MRCA: Most Recent Common Ancestor). Finalmente, los resultados son interpretados en base a modelos teóricos pertenecientes al área de la genética de poblaciones. En el epígrafe 1.3 se profundizará en sentido general sobre estos modelos teóricos y sus inconvenientes.

1.3 Diagrama del proceso actual

En el epígrafe anterior se realizó una descripción de literal del proceso actual. Para una mejor comprensión de dicho proceso presentamos el siguiente diagrama utilizando la herramienta Bonita Studio:

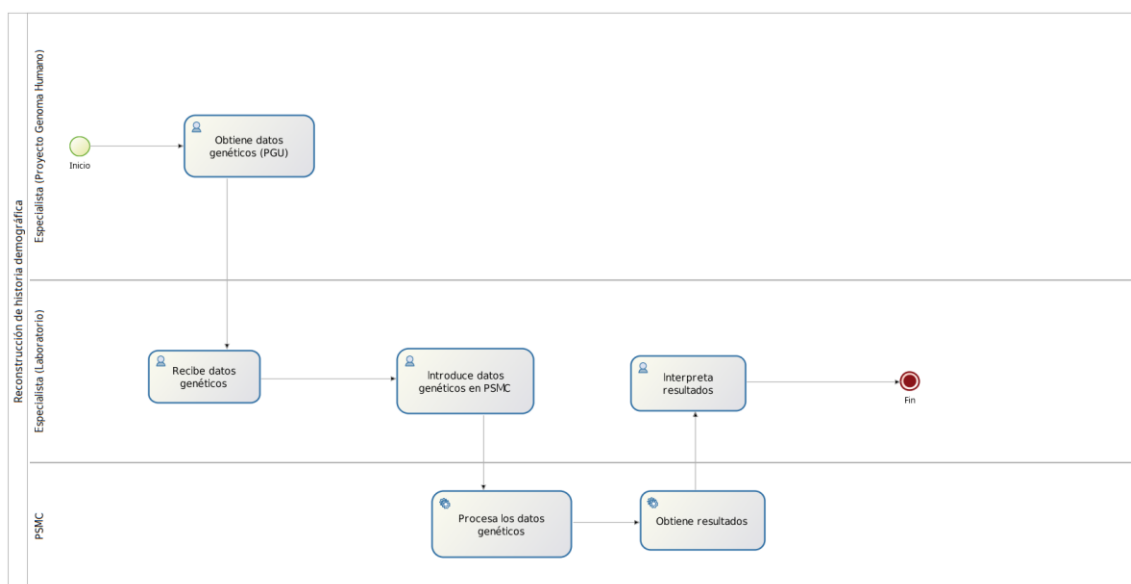


Figura 1. Diagrama del proceso actual

Nota: Detalles sobre la herramienta Bonita Studio serán tratados en epígrafes posteriores.

1.4 Análisis crítico del proceso actual

El proceso de reconstrucción de las historias demográficas de las especies tiene un gran inconveniente. Este radica en el momento de interpretar los resultados obtenidos por los softwares de análisis genéticos.

Como se ha mencionado anteriormente, dichos resultados son interpretados utilizando modelos teóricos bien establecidos y aceptados en genética de poblaciones, por ejemplo; el modelo de Wright-Fisher [11]. Básicamente supone que la población es de tamaño constante, que hay cruzamientos aleatorios entre los individuos (*panmixia*⁵) y que las generaciones no se solapan (discretización de las generaciones). Tampoco tiene en cuenta factores como la mutación, la recombinación, el sexo, *la estructura*, la selección, ni otros aspectos biológicos que pudieran tener cierta influencia en la práctica. Por tanto, cuando se realizan interpretaciones mediante modelos como el antes mencionado, quedan ciertas dudas en la veracidad de estas.

1.5 Análisis y optimización del proceso actual

1.5.1 Diagrama de Ishikawa

Para el análisis del proceso actual se aplicó una técnica de diagramación conocida como espina de pescado o diagrama de Ishikawa:

⁵ Población basada en cruzamientos aleatorios.

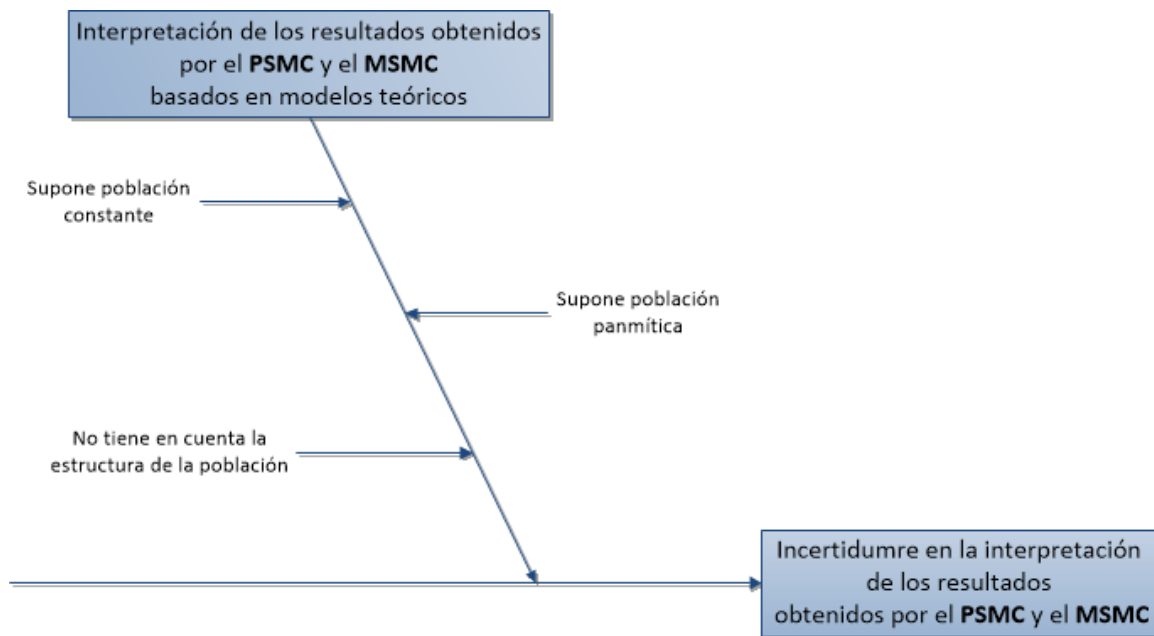


Figura 2. Diagrama Ishikawa para el análisis y optimización

Nota: Detalles sobre el diagrama de Ishikawa serán tratados en epígrafes posteriores.

1.5.2 Procesos objetos de automatización

Por todo lo antes expuesto, expertos en el área han propuesto un nuevo modelo, *the Non-Stationary Structured Coalescent* (NSSC) que incorpora eventos demográficos (cambios en el flujo de genes) para modelos de casi cualquier complejidad y permite interpretar datos genómicos en virtud de esta nueva perspectiva [13] [14]. La incertidumbre en la interpretación actual de los resultados debe reducirse drásticamente debido a la reestructuración de las hipótesis.

Como consecuencia los investigadores se han visto en la necesidad de evaluar si el modelo de población estructurada se ajusta más a los datos genéticos que los modelos existentes.

La inclusión del modelo NSSC al proceso de interpretación actual, genera una serie de sub-procesos que son necesarios automatizar:

1. Graficado del modelo NSSC bajo ciertos parámetros, así como su configuración con el objetivo de ir obteniendo diversas curvas. Esto para permitir un análisis más detallado.
2. Comparaciones visuales de los gráficos obtenidos por el PSMC y el MSMC con el modelo de población estructurada NSSC. Dichas comparaciones estarán basadas en la edición de los principales parámetros de las funciones, lo cual añadirá una nueva perspectiva de interpretación por parte de los biólogos.
3. Ajuste de las curvas del modelo NSSC con respecto a las obtenidas mediante los softwares PSMC y MSMC, en vista de mejorar las interpretaciones.

1.6 Tendencia y tecnologías actuales

Para la creación de software es necesario tener en cuenta ciertos y determinados factores que pueden facilitar el trabajo en dependencia de las características del producto y la experiencia del equipo de desarrollo. Hoy en día se dispone de una variada cantidad de técnicas, herramientas y lenguajes de programación que agilizan el proceso de desarrollo de software. La selección de dichos elementos queda en manos de los integrantes del proyecto, los cuales evalúan las características del producto para implementar las posibles soluciones.

1.6.1 Identificación de las variables deseadas para la solución

No	Variable	Valor deseado
1	Sistema operativo en que se ejecutará	Multiplataforma (Windows, Linux, Mac OS)
2	Costos de adquisición	Gratis

3	Funcionalidades	Se espera que el conjunto de funcionalidades implementadas den respuesta al objetivo específico planteado.
4	Seguridad	Al ser una aplicación de escritorio, la vulnerabilidad de la misma debe reducirse.
5	Requisitos de instalación y operación	<p>1. Requisitos de software:</p> <ul style="list-style-type: none"> • Instalación de NodeJS • Instalación de Electron • Instalación de Anaconda <p>2. Requisitos de Hardware:</p> <ul style="list-style-type: none"> • Procesador de 32 bits (x86) o 64 bits (x64) a 1 gigahercio (GHz). • Memoria RAM de 1 gigabyte (GB) o superior. • 100 MB mínimo de almacenamiento en disco duro.

Tabla 1. Variables para cumplir con la solución deseada

1.6.2 Características de las soluciones informáticas estudiadas

- Aplicación de escritorio como alternativa escogida

El proyecto desarrollado en este trabajo tiene como uno de sus objetivos específicos y principal, implementar al menos dos metaheurísticas que respondan al problema del ajuste de curvas. Evidentemente el aprovechamiento de los recursos de las PCs en las cuales se ejecutan dichos procesos es sumamente importante. Las aplicaciones de escritorio aprovechan de manera muy eficiente los recursos de la máquina en la cual se ejecuta, siendo bien robustas y el tiempo de respuesta considerablemente rápido. También se debe tener en cuenta que la información que se procesa por la herramienta no se necesita socializar, sino más bien garantizar la portabilidad de la misma en varias plataformas.

- Lenguajes de programación

JavaScript:

JavaScript [15] es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos basado en prototipos, imperativo, débilmente tipado y dinámico.

Se utiliza principalmente del lado del cliente (*client-side*), implementado como parte de un navegador web, permitiendo mejoras en la interfaz de usuario y páginas web dinámicas. También es posible utilizar JavaScript del lado del servidor (*Server-side JavaScript* o *SSJS*). Su uso en aplicaciones externas a la web, por ejemplo en documentos PDF, aplicaciones de escritorio (como es el caso de este software) es también significativo.

Desde el 2012, todos los navegadores modernos soportan completamente ECMAScript 5.1, una versión de JavaScript. Los navegadores más antiguos soportan por lo menos ECMAScript 3. La sexta edición se liberó en julio del 2015 [16].

Python:

La aplicación desarrollada reutilizará código escrito en Python por el equipo de Investigación de la Universidad de Toulouse, Francia. La comunidad de desarrolladores ha creado potentes módulos de cálculo numérico. Hoy en día existen módulos específicos de casi todas las áreas científico-ingenieriles.

Es un lenguaje que se usa en varias áreas de tecnología: web, redes, procesamiento de datos, inteligencia artificial, etc. Gracias al uso de expresiones comunes, Python requiere menos líneas de código para realizar tareas básicas. En promedio, el código escrito en Python es de tres a cinco veces más corto que Java y de cinco a 10 veces más corto que C++. Adicionalmente, Python tiene una librería estándar que permite ejecutar otras funciones y tareas más complejas con mayor facilidad que otros lenguajes [17]. Python es un lenguaje perfecto para cualquier programador principiante que esté decidido a aprender por cuenta propia por la facilidad de su sintaxis.

- **Electron**

Anteriormente conocido como Atom Shell [18], es un framework de código abierto basado en JavaScript, desarrollado y mantenido por la comunidad oficial que apoya el proyecto GitHub, el cual nos permite desarrollar sistemas de escritorio multiplataforma (Cross Platform Desktop Apps), mediante el uso de tecnologías web (JavaScript, HTML y CSS).

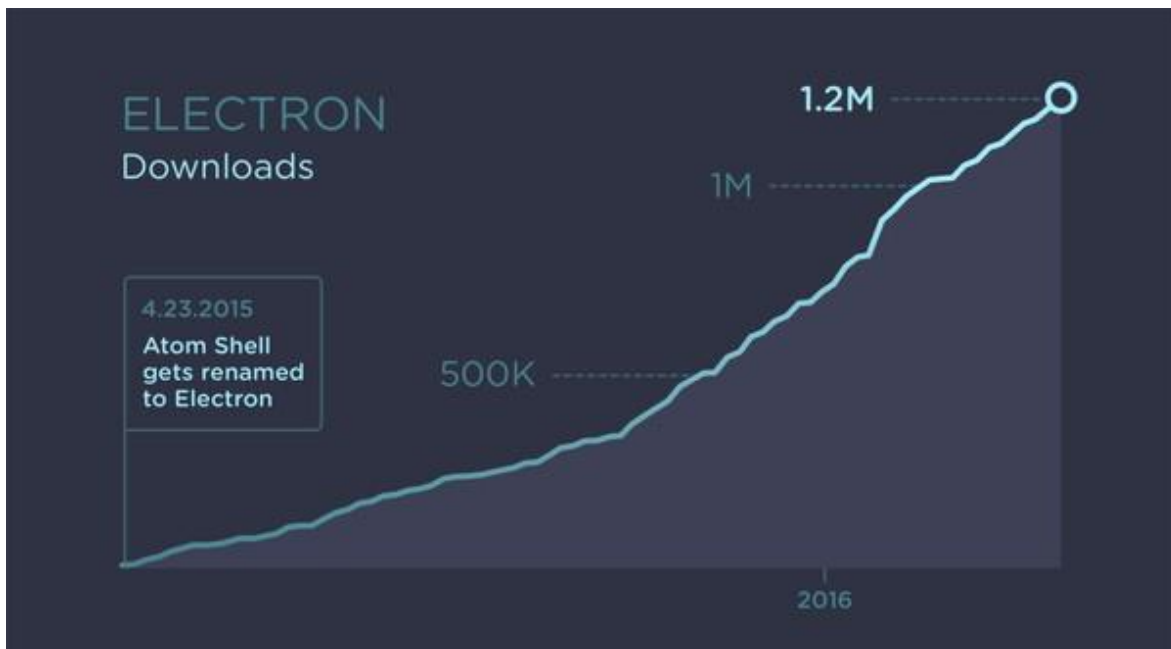


Figura 3. Actualidad de descargas de Electron

- Descripción de la plataforma

Electron funciona bajo un subconjunto mínimo de librerías de varios frameworks como NodeJS, Chromium y el motor V8 JavaScript, además proporciona acceso a APIs nativas enriquecidas mediante el motor de Chromium [18], el cual es controlado mediante JavaScript. Además, enlaza a librerías gráficas del sistema operativo, ya que su GUI se desarrolla mediante HTML, pero podemos aprovechar ciertas características de las librerías nativas mediante Chromium. Compañías como Microsoft, Facebook, Slack, Docker, entre otras, están utilizando este framework como parte de su stack de desarrollo para aplicaciones cross platform.

Ofrece también a los desarrolladores una serie de características, tales como: reporte de fallos, actualizaciones automáticas, depuración y análisis sistemático del contenido de la fuente de datos, menús y notificaciones nativas.

Como beneficio, la comunidad añade una *aplicación demo* que enseña cómo usar las API's de Electron con ejemplos interesantes para iniciar a los interesados en el

desarrollo utilizando este framework. Hoy en día cuenta con un amplio ecosistema de utilidades y aplicaciones complementarias para su uso.

Electron por defecto soporta Node.js v6 y la última versión estable de Chromium. Por tal motivo, se pueden usar casi todas las características de ECMAScript y también funciones como Async / Await. Además, es posible hacer una aplicación integrando funcionalidades de React.js o AngularJS que permitan consumir servicios REST API como si fuera una aplicación web. Por otra parte, permite conectarse a una base de datos debido a que internamente está desarrollado con tecnologías web, con la diferencia que este aprovecha las condiciones y recursos que le brinda el sistema operativo como aplicación de escritorio, con características superiores a una aplicación web, incluyendo la integridad y la seguridad [18].

- Control de versiones (Git)

Los sistemas de control de versiones son programas que tienen como objetivo controlar los cambios en el desarrollo de cualquier tipo de software. De manera muy sencilla permite conocer el estado actual de un proyecto, los cambios que se le han realizado a cualquiera de sus piezas y las personas que intervinieron en ellos, entre otros.

El control de versiones es una de las tareas fundamentales para la administración de un proyecto de desarrollo de software en general. Surge por la necesidad de mantener y llevar el control del código en sus distintos estados. Es absolutamente necesario para el trabajo en equipo, pero resulta útil incluso a desarrolladores independientes.

Git [19] es un sistema de control de versiones distribuido y de código abierto relativamente nuevo. Ofrece las mejores características en la actualidad sin perder la sencillez. En los últimos tiempos no ha parado de crecer y de ser usado por muchos desarrolladores en el mundo. A los programadores los ha ayudado a ser más eficientes

en el trabajo, ya que ha universalizado las herramientas de control de versiones de software que hasta entonces no estaban tan popularizadas. Git es multiplataforma, por lo que podemos usarlo y crear repositorios locales en todos los sistemas operativos más comunes: Windows, Linux o Mac.

- Sistema para el manejo de control de versiones (GitHub)

GitHub constituye una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git. Se utiliza principalmente para la creación de código fuente de programas de computadora. El software que opera GitHub fue escrito en Ruby on Rails. Desde enero de 2010, GitHub opera bajo el nombre de GitHub, Inc. Anteriormente era conocida como Logical Awesome LLC. El código de los proyectos alojados en GitHub se almacena típicamente de forma pública, aunque utilizando una cuenta de pago. También permite hospedar repositorios privados.

Principales características:

1. Contiene gráficos que permiten ver cómo los desarrolladores trabajan en sus repositorios y bifurcaciones del proyecto.
2. Posee funcionalidades como si se tratase de una red social.
3. Constituye una herramienta para el trabajo colaborativo entre programadores.
4. Funciona también como un gestor de proyectos.

- Metodología de desarrollo adoptada (Scrum)

En los últimos años, se han desarrollado dos corrientes en lo referente a procesos de desarrollo; los métodos *pesados o tradicionales* y los *ligeros o ágiles*. A diferencia de las metodologías tradicionales, las ágiles se enfocan en la obtención rápida de una aplicación funcional, esto garantiza una flexibilidad adaptativa ante los cambios que pudieran surgir a lo largo del proceso de desarrollo de la aplicación. Además, tienen una mayor relevancia las interacciones con el cliente, dándole un sentido de implicación y colaboración. El Manifiesto Ágil [20] se considera el punto de partida, y surgió de la necesidad de trazar una definición común para todas las metodologías que cumplieran con determinados principios. Específicamente, el desarrollo ágil se centra en la existencia de equipos de trabajo multifuncionales con autonomía de decisión, y no en grandes divisiones agrupadas por jerarquía y funcionalidad. Además, se basa en iteraciones cortas, de manera que el cliente pueda ofrecer su opinión sobre la calidad de la aplicación, sintiéndose a la vez, partícipe en el proceso de creación.

Scrum [21] es una de las metodologías ágiles de desarrollo de software más exitosas en la actualidad que permite a los desarrolladores crear rápidamente código de alta calidad. Es un marco de trabajo iterativo e incremental basado en la adaptación continua a las circunstancias evolutivas de un proyecto, apoyándose en iteraciones cortas conocidas como Sprints. Scrum emplea una lista priorizada de requisitos llamada Pila del Producto, donde aparecen reflejadas las necesidades del cliente. Los elementos de esta lista se traducen como Historias de Usuario [22] que reflejan las características, comportamientos y funcionalidades que debe tener la aplicación. Está enfocado en el desarrollo, en la entrega continua de software funcional y en mejorar la calidad del proceso de construcción del producto con cada iteración.

Cada componente dentro del marco de trabajo sirve a un propósito específico y es esencial para el éxito de Scrum. Los miembros del equipo pueden ser desarrolladores, probadores, analistas, arquitectos, redactores o diseñadores. El equipo es

multidisciplinario, lo que significa que todos sus miembros poseen las habilidades suficientes para llevar a cabo el trabajo, sin definir un liderazgo de antemano entre los miembros del equipo.

- Gestor de procesos de negocios BPM (Bonita Studio)

La gestión de procesos de negocios (Business Process Management: BPM) consiste en una metodología corporativa. Tiene como objetivo mejorar la eficiencia dentro de las organizaciones por medio de la gestión de los procesos de negocio que se deben modelar, organizar, documentar y optimizar de forma continua [23]. Incluye un conjunto de recursos y actividades interrelacionadas que transforman elementos de entrada en elementos de salida.

BonitaSoft [24] (herramienta utilizada en el epígrafe 1.3) es el primer editor y líder de soluciones BPM (Business Process Management) en software libre. Es una empresa de software creada en el 2009 por Miguel Valdés Faura, Charles Souillard y Rodrigue Le Gall, fundadores del proyecto de código abierto Bonita. Esta solución ha sido descargada más de 140.000 veces por un gran número de empresas, con el fin de desarrollar aplicaciones informáticas. BonitaSoft es partner de Talend y de Bull, y es miembro activo del consorcio OW2 [24]. Uno de los objetivos de BonitaSoft es democratizar el BPM con una solución fácil e intuitiva que permita minimizar el costo de implantación. Esta aspira a convertirse en el líder mundial en soluciones de gestión de procesos empresariales de código abierto (Open Source Business Process Management - BPM), proporcionando soluciones de BPM flexibles y potentes para las organizaciones.

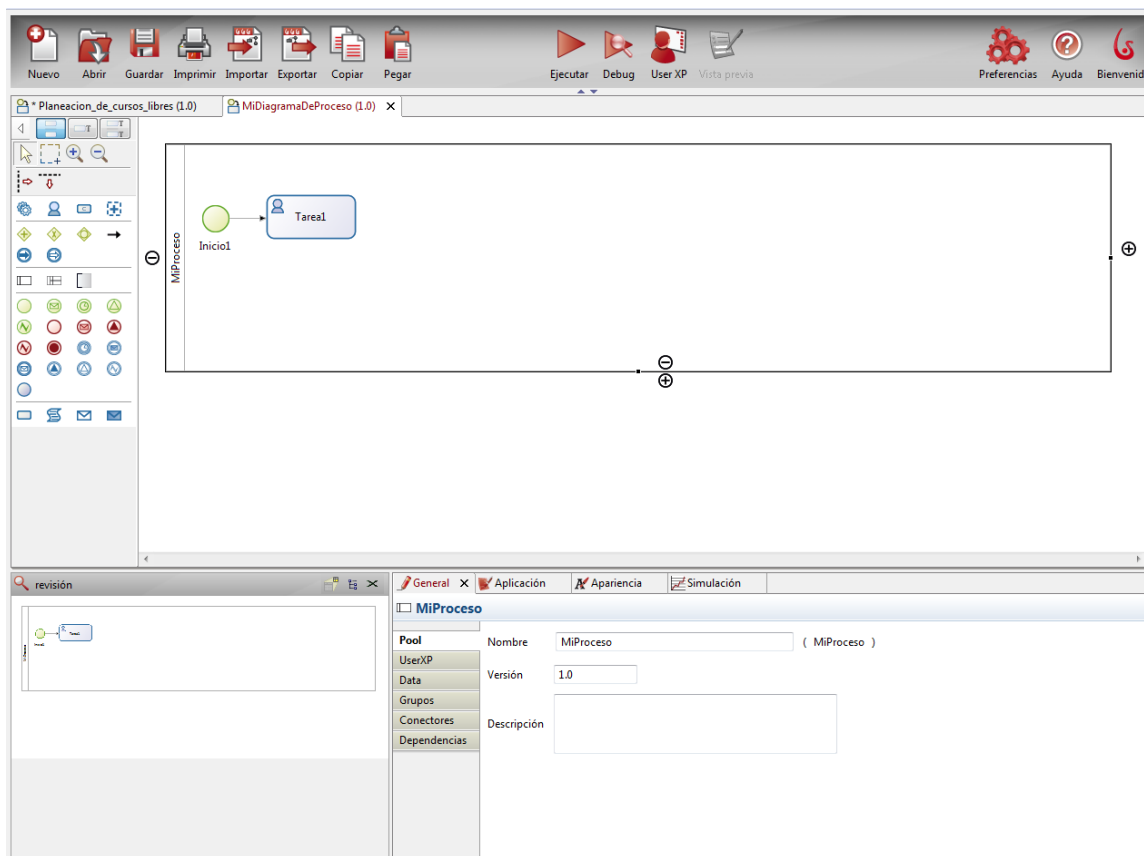


Figura 4. Gestor de procesos de negocios BPM (Bonita Studio)

- Técnicas de diagramación utilizado (Diagrama de Ishikawa)

El diagrama de causa - efecto de Ishikawa [25] (utilizado en el epígrafe 1.5.1) debe su nombre al reconocido ingeniero japonés Kaoru Ishikawa. Este lo introdujo y lo popularizó con éxito durante el análisis de problemas en 1943 en la Universidad de Tokio. Durante una de sus sesiones de capacitación a ingenieros en una empresa metalúrgica, explicó que varios factores pueden agruparse para ser interrelacionados. Este diagrama es también conocido bajo las denominaciones de cadena de causas-consecuencias, diagrama de espina de pescado o “fish-bone”. El diagrama de Ishikawa es un método gráfico que se usa para efectuar un diagnóstico de las posibles causas que provocan ciertos efectos, los cuales pueden ser controlables. Se usa el diagrama de causas-efecto para:

1. Analizar las relaciones causas-efecto.
2. Comunicar las relaciones causas-efecto.
3. Facilitar la resolución de problemas desde el síntoma, pasando por la causa hasta la solución.

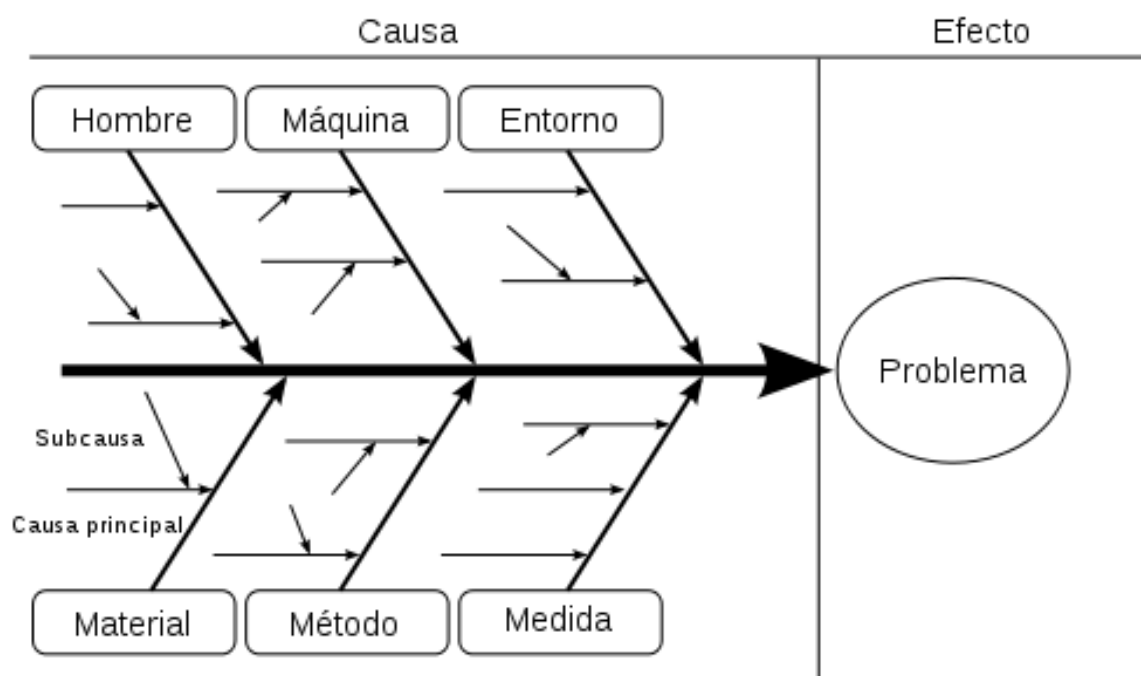


Figura 5. Diagrama de Ishikawa

La figura anterior representa los principales factores (causas) que afectan la característica de calidad en estudio como líneas principales. Recursivamente se continúa el procedimiento de subdivisión hasta que están representados todos los factores factibles de ser identificados. El diagrama de Ishikawa permite apreciar fácilmente y en perspectiva, todos los factores que pueden ser controlados usando distintas metodologías [25]. Al mismo tiempo permite ilustrar las causas que afectan una situación dada, clasificando e interrelacionando las mismas.

1.7 Análisis crítico de las fuentes bibliográficas utilizadas

Comprender a profundidad cualquier tema objeto de estudio en sentido general, conlleva un discernimiento detallado sobre los distintos conceptos presentes en el mismo. Para la conformación de este trabajo se consultaron varias fuentes bibliográficas entre las que se destacan artículos digitales publicados en prestigiosas universidades y centros de estudios genéticos internacionales. Por ejemplo, las referencias [10], [11], [12], [13] y [14] entre otras, forman parte de un conjunto de artículos de primer nivel, publicados por el equipo de investigadores que complementan el desarrollo del presente proyecto. Es necesario destacar que los principales interesados en su realización, lo constituyen especialistas de la Universidad de Tolouse, el Instituto de Matemáticas de Tolouse y el laboratorio de Evolución y Diversidad Biológica de la Universidad Paul Sabatier (Francia). Dichos especialistas facilitaron al autor de este documento, la información especializada sobre el tema.

En cuanto a de desarrollo de software, algunas referencias de tesis de diploma, libros y sitios confiables de internet (o sea, documentaciones oficiales), solidificaron la elaboración del proyecto. Esto para ampliar los conocimientos de las nuevas tecnologías, métodos y metodologías necesarias para la creación de un producto de software de calidad.

1.8 Conclusiones parciales del capítulo

La esencia del capítulo anterior no es más que el punto de partida de lo que será el proceso de desarrollo de una aplicación científica perteneciente al campo de la bioinformática.

El punto fundamental es la mejora de la interpretación de los resultados obtenidos a partir de los softwares de análisis de datos genéticos más populares. Los investigadores no poseen una herramienta para comprobar de manera práctica la veracidad de sus estudios respecto al nuevo método basado en el modelo NSSC.

Para dar cumplimiento a lo antes mencionado, se establecieron las bases de la investigación que proporcionaron el fundamento para el desarrollo de la solución propuesta. El autor realizó una descripción detallada de la metodología adoptada (Scrum), además de algunas técnicas, lenguajes de programación y herramientas utilizadas durante el desarrollo del software. Por último, cabe destacar que la consulta de la bibliografía especializada jugó un papel sumamente importante.

Capítulo II: Especificaciones de funcionalidad

2.1 Introducción

Para obtener una descripción del comportamiento del sistema a desarrollar resulta de suma importancia conocer las necesidades de los interesados. Por tanto, lo primero es definir quiénes son todas aquellas personas que intervienen o tienen interés en el desarrollo del proyecto y pueden aportar conocimiento útil para facilitar la comprensión del mismo. De tal manera, se especifican entonces los temas de trabajo y se recogen una lista de requerimientos llamadas Historias de Usuarios. Estas describen textualmente lo que se demanda desde la perspectiva del cliente y se añaden a la Pila del Producto organizadas por temas para una mejor comprensión y entendimiento. Lo anterior corresponde con la metodología de desarrollo utilizada (SCRUM). Las Historias de Usuario están determinadas a partir de la fusión de las principales características y funcionalidades identificadas por los clientes e interesados en el desarrollo de la aplicación.

Este capítulo destaca también la importancia de las pruebas automatizadas y refleja algunos ejemplos de pruebas unitarias realizadas al software. Por último, se describen los requisitos no funcionales del sistema.

2.2 Definición de interesados

Los interesados son todas aquellas personas, grupos u organizaciones que tienen interés en el desarrollo del proyecto. No tienen que ser siempre personas que interactúan directamente con la aplicación. En dicho caso, la persona no utilizará la herramienta en su trabajo diario, sin embargo, estará consciente de que los beneficios de la informatización repercutirán directamente en los resultados del grupo investigativo al que pertenece.

Interesado	Descripción
Biólogos y Antropólogos de los laboratorios de Evolución y Diversidad Biológica de la Universidad Paul Sabatier, Francia.	Son los encargados de interpretar los resultados obtenidos a partir de los softwares de análisis de datos genéticos más populares.
Investigadores del Instituto de Matemáticas Aplicadas de Toulouse, Francia.	Son aquellos que desarrollaron el modelo matemático NSSC, en vista de lograr una mejor interpretación de los resultados obtenidos.

Tabla 2. Interesados del proyecto

2.3 Definición de Temas

Con el objetivo de organizar el desarrollo de la aplicación, se definen Temas para agrupar las Historias de Usuarios y de esta manera centralizar los esfuerzos. Además, se facilita la comprensión y la identificación de Historias de Usuario en la Pila de Producto. En la siguiente tabla se muestran los Temas propuestos para el desarrollo de este proyecto.

Tema	Descripción
1. Procesamiento de datos	Relacionado con la entrada de datos a la aplicación.
2. Manipulación de datos	Relacionado con el graficado, ajuste manual y automático de las funciones. El ajuste automático es la pieza clave de este tema.
3. Almacenamiento de información	Relacionado con la salva de los archivos de la aplicación.
4. Configuración	Relacionado con la edición y configuración de los elementos de la aplicación.

Tabla 3. Temas del proyecto

2.4 Pila del producto

La pila del producto es la lista ordenada de todo aquello que el cliente necesita en la aplicación. En las siguientes tablas, se muestran todas las Historias de Usuario que fueron definidas para el desarrollo de este proyecto. Se puede apreciar la estructura de las Historias de Usuario: Nombre, Cómo, Necesito, Para y Estimación (Est.) [22]. El *Nombre* define cómo se llamará la Historia de Usuario; debe ser único, corto y claro. El *Cómo* indica el interesado que ejecutará la funcionalidad en estudio. El *Necesito* define la necesidad que satisface la Historia de Usuario. El *Para* establece lo que va a resolver la Historia de Usuario. La *Estimación* indica la complejidad que tendrá para el programador y está basada en su experiencia, en este caso, se utiliza la serie de Fibonacci [26], desde el 1 hasta el 21. Estas historias están contenidas en la Pila del Producto y son organizadas por Temas para facilitar su comprensión y sentido.

Nota: A continuación (de manera genérica) se usará el término *Investigador* al referirse a un *Biólogo* o *Antropólogo*.

2.4.1 Tema 1: Procesamiento de datos

Nombre	Cómo	Necesito	Para	E
Cargar ficheros (.psmc)	Investigador	Extraer los datos de los ficheros generados por el software: (PSMC)	Utilizarlos en la construcción de las historias demográficas de las especies	13
Cargar ficheros (.msmc)	Investigador	Extraer los datos de los ficheros generados por el software: (MSMC)	Utilizarlos en la construcción de las historias demográficas de las especies	13

Cargar ficheros (.psmcp)	Investigador	Extraer los datos de los ficheros (.psmcp) generados por el software: (ADHOS)	Utilizarlos en la construcción de las historias demográficas de las especies que han sido personalizadas por el usuario	10
Cargar ficheros (.msmcp)	Investigador	Extraer los datos de los ficheros (.msmcp) generados por el software: (ADHOS)	Utilizarlos en la construcción de las historias demográficas de las especies que han sido personalizadas por el usuario	10
Cargar ficheros (.nssc)	Investigador	Extraer los datos de los ficheros (.nssc) generados por el software: (ADHOS)	Utilizarlos en el graficado de la curva IICR (<i>Inverse instantaneous coalescence rate</i>) la cual corresponde al modelo NSSC	10
Cargar ficheros (.snssc)	Investigador	Extraer los datos de los ficheros (.snssc) generados por el software:	Utilizarlos en la continuación de la construcción de una curva IICR	10

		(ADHOS)		
Cargar ficheros (.adhos)	Investigador	Extraer los datos de los ficheros (.adhos) generados por el software: (ADHOS)	Retomar un escenario de análisis de curvas previamente hecho	12

Tabla 4. Historias de Usuarios del tema procesamiento de datos

2.4.2 Tema 2: Manipulación de datos

Nombre	Cómo	Necesito	Para	E
Visualizar las historias demográficas obtenidas por el PSMC	Investigador	Graficar las funciones obtenidas a partir del software PSMC	Realizar un análisis visual de los resultados	10
Visualizar las historias demográficas obtenidas por el MSMC	Investigador	Graficar las funciones obtenidas a partir del software MSMC	Realizar una análisis visual de los resultados	10
Abrir una ventana para la creación de un modelo NSSC (tipo GENERAL)	Investigador	Una ventana que permita introducir los datos necesarios	Preparar el escenario con vista a graficar la curva IICR basada en el modelo NSSC	15
Abrir una ventana para la creación de un modelo	Investigador	Una ventana que permita introducir los	Preparar el escenario con vista a graficar la	15

NSSC (tipo SYMMETRICAL)		datos necesarios	curva IICR basada en el modelo NSSC	
Visualizar las curvas IICR (tipo GENERAL y SYMMETRICAL)	Investigador	Graficar los distintos tipos de curvas IICR	Permitir comparar las historias demográficas de las especies y las curvas IICR	17
Cargar el escenario de una curva IICR	Investigador	Acceder a los parámetros de las curvas IICR ya construidas y visualizadas	Editarlos e ir obteniendo nuevas curvas. Ayuda también en el ajuste manual que mejora la interpretación de las historias demográficas de las especies	15
Escalar los distintos gráficos	Investigador	Escalar los gráficos obtenidos por los softwares PSMC, MSMC y ADHOS, con respecto a los parámetros μ , S y N_{ref}	Ayudar en el ajuste manual de las curvas y otros posibles análisis	8
Reiniciar la escala de un gráfico seleccionado	Investigador	Volver al estado inicial de un gráfico escalado	Nuevamente realizar escalas partiendo del estado inicial	5
Reiniciar la escala	Investigador	Tener la	Nuevamente	6

de todos los gráficos		posibilidad de volver al estado inicial de todos los gráficos escalados	realizar escalas partiendo del estado inicial de estos. Además, no tener que hacerlo de manera independiente si no fuese necesario	
Calcular la distancia entre dos curvas basadas en distintos modelos (PSMC/MSMC y NSSC)	Investigador	Conocer la distancia entre dos curvas seleccionadas (una tiene que ser la que representa la historia demográfica de una especie y la otra la curva IICR)	Utilizarla en el ajuste de curvas	18
Actualizar y mostrar distancia en pantalla	Investigador	Observar en pantalla la distancia en cada instante de la reconstrucción de una curva	Tener mejor certeza en el ajuste de curvas que se realiza	12
Ajuste automático de dos curvas basadas en distintos modelos (PSMC/MSMC y NSSC)	Investigador	Ajustar de manera automática dos curvas seleccionadas (una tiene que ser la que representa	Mejorar la interpretación de las historias demográficas de las especies	21

		la historia demográfica de una especie y la otra la curva IICR)		
--	--	---	--	--

Tabla 5. Historias de Usuarios del tema manipulación de datos

2.4.3 Tema 3: Almacenamiento de información

Nombre	Cómo	Necesito	Para	E
Guardar y guardar como... archivos de extensión (.psmc)	Investigador	Guardar la información de los archivos (.psmc) donde su información fue manipulada, en otra extensión (.psmc)	Hacer persistente la información y poder manipularla sin alterar el archivo original (.psmc) que le dio origen	8
Guardar y guardar como... archivos de extensión (.msmc)	Investigador	Guardar la información de los archivos (.msmc) donde su información fue manipulada, en otra extensión (.msmc)	Hacer persistente la información y poder manipularla sin alterar el archivo original (.msmc) que le dio origen	8
Guardar archivos de extensión (.nssc)	Investigador	Guardar los cambios hechos a la información de un archivo (.nssc) ya existente	Recuperar el último estado de la información antes de ser guardada	8

Guardar archivos de extensión (.snssc)	Investigador	Guardar los cambios hechos durante la construcción de una curva IICR	Si ocurriera algún imprevisto, no tener que volver a entrar todos los datos, debido a que en ocasiones pudiese tornarse muy trabajoso	15
Guardar aplicación (.adhos)	Investigador	Guardar el escenario perteneciente a una aplicación	Retomar el estado guardado en vista de continuar el análisis	10

Tabla 6. Historias de Usuarios del tema almacenamiento de información

2.4.4 Tema 4: Configuración

Nombre	Cómo	Necesito	Para	E
Editar color de cada función	Investigador	Cambiar el color de las funciones	Identificarlas de manera personalizada	5
Eliminar función	Investigador	Eliminar una función en caso que sea necesario	Trabajar sólo con las que son imprescindibles	5
Limpiar la aplicación	Investigador	Eliminar todas las funciones	Dejar el escenario limpio para reiniciar un nuevo estudio	6
Enfocar función (zoom)	Investigador	Enfocar una función determinada	Observar mejor sus características	10

Definir un modelo de población estructurada (NSSC)	Investigador	Tener la opción de seleccionar el modelo de población que deseo crear	Luego dar entrada a la información en dependencia del tipo de modelo	5
Añadir evento demográfico	Investigador	En la ventana de creación de un modelo NSSC, tener la opción de añadir evento demográfico	En caso de surgir algún cambio de idea en la configuración, no tener que volver a la ventana principal	11
Eliminar evento demográfico	Investigador	En la ventana de creación de un modelo NSSC, tener la opción de eliminar evento demográfico	En caso de surgir algún cambio de idea en la configuración, no tener que volver a la ventana principal	11
Añadir isla	Investigador	En la ventana de creación de un modelo NSSC, tener la opción de añadir isla	En caso de surgir algún cambio de idea en la configuración, no tener que volver a la ventana principal	10
Eliminar isla	Investigador	En la ventana de creación de un modelo NSSC, tener la opción de	En caso de surgir algún cambio de idea en la configuración, no	10

		eliminar isla	tener que volver a la ventana principal	
--	--	---------------	---	--

Tabla 7. Historias de Usuarios del tema configuración

2.5 Tipos de pruebas ejecutadas

- Pruebas unitarias

La realización de pruebas al código ha formado parte del desarrollo de software desde sus inicios. A lo largo de los años, la construcción de pruebas ha ido incrementando su alcance y han aflorado diferentes niveles entre la que se destacan las pruebas unitarias.

En programación, una prueba unitaria [27] es una forma de comprobar el correcto funcionamiento de una unidad de código. Por ejemplo, en diseño estructurado o en diseño funcional es una función o procedimiento, en diseño orientado a objetos es una clase. Esto sirve para asegurar que cada unidad funcione correctamente y eficientemente por separado. Además de verificar que el código hace lo que tiene que hacer, verificamos que sea correcto el nombre, los nombres y tipos de los parámetros, el tipo de lo que se devuelve, que si el estado inicial es válido entonces el estado final es válido. La idea es escribir casos de prueba para cada función no trivial o método en el módulo, de forma que cada caso sea independiente del resto.

En el presente proyecto se decidió hacer uso del tipo de pruebas descritas anteriormente fundamentado en las ventajas que nos proporcionan. Estas se describen a continuación:

1. Fomentan el cambio

Las pruebas unitarias facilitan que el programador cambie el código para mejorar su estructura (lo que se conoce como refactorización), puesto que permiten hacer pruebas sobre los cambios y así asegurarse que en estos no han sido introducido errores.

2. Simplifican la integración

Permiten llegar a la fase de integración con un grado alto de seguridad de que el código está funcionando correctamente. De esta manera se facilitan las pruebas de integración.

3. Documentan el código

Las propias pruebas son documentación del código, puesto que ahí se puede ver cómo utilizarlas.

4. Separación de la interfaz y la implementación

Dado que la única interacción entre los casos de prueba y las unidades bajo prueba son las interfaces de estas últimas, se puede cambiar cualquiera de los dos sin afectar al otro, a veces usando objetos maquetados (mock object - maqueta) que habilitan de forma aislada (unitaria) el comportamiento de objetos complejos.

5. Los errores están más acotados y son más fáciles de localizar

Las pruebas unitarias posibilitan desenmascarar y localizar los posibles errores en la aplicación.

- Frameworks utilizados para las pruebas unitarias (*UnitTest* para Python y *Mocha* para JavaScript)

1. El módulo *UnitTest* [28] es la versión para Python del framework de pruebas unitarias originalmente desarrollado por Kent Beck para Smalltalk. Este módulo está disponible en la librería estándar y soporta: automatización de pruebas, funciones `setUp()` y `tearDown()` con código a ejecutar antes y después de las pruebas (esta característica es llamada Fixtures), agregación de pruebas en colecciones (test suites) e independencia de pruebas. El módulo exporta la clase `unittest.TestCase` de la cual se deben derivar todas las clases que representan los casos de prueba, que a su vez pueden contener una o más pruebas unitarias. Exporta además, un conjunto de funciones `assert` predefinidas, que facilitan la escritura de las pruebas.

2. *Mocha* [29] es un marco de prueba de JavaScript con numerosas funciones que se ejecuta en Node.js y en el navegador, lo que hace que las pruebas asíncronas sean simples y divertidas. Las pruebas de Mocha se ejecutan en serie, lo que permite informes flexibles y precisos, al tiempo que se asignan excepciones no detectadas a los casos de prueba correctos.

2.5.1 Diseño de los casos de pruebas unitarias

Teniendo en cuenta que la aplicación gira fundamentalmente alrededor de la manipulación de tres tipos de funciones (las correspondientes a los modelos PSMC, MSMC y NSSC), se decidió diseñar tres casos de pruebas fundamentales presentadas a continuación:

1. Comprobar que el método `get_Model_NSSC()` de la clase *Python_Communicator*, garantiza que los vectores recibidos del script `get_Model_NSSC.py` son los correctos:

Prueba	Precondiciones	Acción	Respuesta
Funcionamiento del método <i>get_Model_NSSC()</i>	Deben existir los vectores generados directamente desde el script de python.	Transferir los vectores desde el script de python hacia el framework Electron.	Deben coincidir los vectores para su correcta representación gráfica.

Tabla 8. Diseño de caso de prueba unitaria (1)

The screenshot shows a code editor with a file named `test.js`. The code is a JavaScript test using `describe` and `it` blocks. It calls `Python_Communicator.get_Model_NSSC` with a sampling vector and a scenario. It then compares the result with a predefined vector using `Application_Uilities.Equals` and asserts the result is true. The terminal below shows the command `npm test` being executed, which runs the test successfully, displaying the message "Good NSSC vectors received of get_Model_NSSC.py" and "1 passing (12ms)".

Figura 6. Diseño de caso de prueba unitaria (1) y resultados

2. Comprobar que los métodos *get_PSMC_results()* y *get_MSMC_results()* realizan una lectura correcta de los archivos *Dai_upper.psmc* y *example_ouput.msmc*.

Prueba	Precondiciones	Acción	Respuesta
Funcionamiento de los métodos <i>get_PSMC_results()</i> y <i>get_MSMC_results()</i>	1. Deben existir los archivos <i>Dai_upper.psmc</i> y <i>example_ouput.msmc</i> 2. Se debe haber	Leer los archivos mediante código escrito en python.	Deben coincidir los valores de los parámetros extraídos manualmente con

	realizado una construcción de los parámetros extraídos manualmente de los archivos.		los leídos a partir del código python.
--	---	--	--

Tabla 9. Diseño de caso de prueba unitaria (2)

The screenshot shows a code editor with a Python file named `test_Get_File_Results.py`. The code defines a class `Test_My_Python_Scripts` that inherits from `unittest.TestCase`. It contains two test methods: `test_get_Information_Dai_Upper_PSMC` and `test_get_Information_Example_Output_MSMC`. The first method checks the results of `get_PSMC_results` for specific values of `rho`, `theta`, and `x_vector`. The second method checks the results of `get_MSMC_results` for a specific `x_vector`. The `if __name__ == '__main__':` block calls `unittest.main()`.

Below the code editor, a terminal window shows the execution of the test. The command `python3 test_Get_File_Results.py` is run, and the output indicates that 2 tests were passed successfully in 0.001s.

```

class Test_My_Python_Scripts(unittest.TestCase):
    def test_get_Information_Dai_Upper_PSMC(self):
        results = get_PSMC_results('Dai_upper.psmc', 'Dai_upper')
        is_Correct = results['rho'] == 0.008744363 and results['theta'] == 0.059070832 and results['x_vector'] == [0.0, 0.008822, 0.018421, 0.028868, 0.040236, 0.0]
        self.assertTrue(is_Correct)
    def test_get_Information_Example_Output_MSMC(self):
        results = get_MSMC_results('example_output.msmc', 'example_output')
        is_Correct = results['x_vector'] == [0.0, 2.36759e-06, 4.79668e-06, 7.29055e-06, 9.85276e-06, 1.24872e-05, 1.51979e-05, 1.79896e-05, 2.08672e-05, 2.38362e-05]
        self.assertTrue(is_Correct)
if __name__ == '__main__':
    unittest.main()

```

```

hector@hector-PC:/media/hector/Datos I/software in progress whit SADMIN(unit test)/Testing$ python3 test_Get_File_Results.py
..
Ran 2 tests in 0.001s

OK
hector@hector-PC:/media/hector/Datos I/software in progress whit SADMIN(unit test)/Testing$

```

Figura 7. Diseño de caso de prueba unitaria (2) y resultados

2.6 Requisitos no funcionales

Los requerimientos no funcionales son propiedades o cualidades que el producto de software debe tener. Estos constituyen una parte significativa de la especificación, añaden funcionalidad al producto y hacen que sea fácil de usar, seguro e interactivo [30].

- Apariencia o interfaz externa

La aplicación debe ofrecer una interfaz sencilla, facilitando la interacción con el usuario, evitando de este modo cualquier tipo de dificultad al acceder a los datos. El diseño debe estar dirigido a contribuir como guía al usuario. Los colores deben responder a las peticiones del cliente. En este caso se destacan el negro, blanco y azul.

- Rendimiento

El rendimiento en una herramienta se evalúa en cuanto a la rapidez de ejecución de cada uno de los algoritmos implementados. En el proyecto a desarrollar se deberá contar con un alto rendimiento debido a que se ejecutan procesos muy complejos que no deben colapsar el software.

- Soporte

El sistema deberá quedar alojado en GitHub. Esto implicará una evolución constante en cuanto a mejora, perfeccionamiento y mantenimiento.

- Portabilidad

La aplicación deberá ser multiplataforma (funcionamiento en Linux, Mac y Windows).

- Legales

El producto saldrá bajo una licencia de código abierto que permitirá que el código fuente sea modificado y redistribuido libremente y sin tener que pagar al autor original.

- Idioma

El software debe estar diseñado completamente en idioma inglés.

- Hardware

Procesador de 32 bits (x86) o 64 bits (x64) a 1 gigahercio (GHz), memoria RAM de 1 gigabyte (GB) o superior y espacio de almacenamiento en el disco duro de 100 MB.

- Software

La implementación de la aplicación puede ser realizada desde cualquier sistema operativo, dado que Electron (framework utilizado) está diseñado para el desarrollo de aplicaciones multiplataforma. De esta manera se necesitará tener instalado Electron JS, NodeJS y Python 3.X.

2.7 Conclusiones parciales del capítulo

El capítulo recién concluido estuvo orientado principalmente al desarrollo de la aplicación. De manera iterativa e incremental se describieron ejemplos y escenarios elaborados en estrecha colaboración con el dueño del producto y otros interesados. La pila de producto nutrida por los antes mencionados, sirvió de guía a través de todo el proceso y garantizó mediante el cumplimiento de cada historia de usuario, que se obtuviera como resultado un producto de software funcional. Esto último motivó a todos los involucrados en el desarrollo de la aplicación y en cortos plazos ya se podía de forma tangible comprobar cada una de las funcionalidades. Es necesario destacar que los investigadores del Instituto de Matemáticas Aplicadas de Toulouse, Francia, fueron los que más aportaron en la conformación de las historias de usuario. De manera directa fueron los más beneficiados. Con la aplicación funcionando, hoy en día se está comprobando en la práctica los resultados de sus estudios. Claro está, el destino final de la herramienta apunta a los biólogos y antropólogos de los laboratorios de Evolución y Diversidad Biológica de la Universidad Paul Sabatier, Francia, quienes serán beneficiados de manera indirecta como resultado de la investigación.

Por último, vale mencionar que los diseños de las pruebas hechas constituyeron un punto clave en la realización del software. Además de documentar el código, sirvieron para dar seguridad acerca del funcionamiento de algunas de las características más importantes. De esta manera garantizaron el correcto desenvolvimiento del producto final de software en esta etapa.

Capítulo III: Descripción de la solución propuesta

3.1 Introducción

En el presente capítulo se brinda una vista global de la arquitectura del sistema con el apoyo de artefactos UML. Para una mejor comprensión de la modelación, se exponen distintos diagramas de clases que forman parte de este trabajo. También se abordan temas como la arquitectura del software, los patrones de diseño utilizados, el tratamiento de errores, entre otros.

Además, se describe detalladamente el problema de optimización combinatoria tratado y su solución. El estudio y uso de algoritmos metaheurísticos constituyen el eje fundamental para la resolución del objetivo específico más relevante de este proyecto.

3.2 Definición del problema de optimización

3.2.1 Optimización

Según Andrea Antoniou [31], la *optimización en general* es el proceso de conseguir la mejor solución, si es posible, para actualizar la solución actual, dependiendo del tipo de problema.

En la práctica, esto se puede traducir en obtener las máximas ganancias o reducir los costos, minimizar las pérdidas, etc. Por consiguiente, la palabra “óptimo” se toma para figurar “máximo” o “mínimo” dependiendo de los sucesos. La práctica de optimización, por otro lado, es la colección de técnicas, métodos, procedimientos, y algoritmos que pueden usarse para encontrar el óptimo.

- Optimización combinatoria

La *optimización combinatoria* es una rama de la *optimización* en matemáticas aplicadas y en ciencias de la computación, relacionada a la investigación de operaciones (IO), teoría de algoritmos y teoría de la complejidad computacional.

Un problema de optimización combinatoria radica en encontrar una solución en el espacio de soluciones, que cumpliendo con las restricciones maximice o minimice la función objetivo [32] [33] [34].

3.2.2 Definición formal del problema de optimización

El problema de optimización a resolver se puede establecer de la siguiente manera:

$$\min_{\varnothing \in \Phi} f(\varnothing) = \int_0^{\infty} 10^{-u} |\widehat{HICR}(u) - N_0 IICR(\varnothing, u/(2N_0))| du \quad (1)$$

Primero se considera un γ que es el *número de eventos demográficos* (entero no negativo) que será fijo. Luego el conjunto de los parámetros Φ se define como:

$$\Phi = \{\varnothing \in \mathbb{R} \times \mathbb{N} \times \mathbb{R}^{3(\gamma+1)} \mid \varnothing = (N_0, n, t_0, c_0, M_0, t_1, c_1, M_1, \dots, t_\gamma, c_\gamma, M_\gamma), \quad \text{s. t}$$

$$N_0 \in \mathbb{R}, \quad N_0 > 0; \quad (2)$$

$$n \in \mathbb{N}, \quad n \geq 2; \quad (3)$$

$$t_0 = 0; \quad (4)$$

$$t_i > 0, \quad i \in \{1, 2, \dots, \gamma\}; \quad (5)$$

$$t_i > t_j, \quad i > j; \quad (6)$$

$$c_i \in \mathbb{R}, \quad c_i > 0, \quad i \in \{0, 1, 2, \dots, \gamma\}; \quad (7)$$

$$M_i \in \mathbb{R}, \quad M_i > 0, \quad i \in \{0, 1, 2, \dots, \gamma\}; \quad (8)$$

donde:

(1) *Función objetivo*. Los valores de la curva \widehat{IICR} en cada $u \in \mathbb{R}$, $u \geq 0$ son conocidos, y la variable u indica el *tiempo en unidades de generaciones*. Estos valores se dan como una entrada que puede leerse desde el archivo de entrada *psmc* o *msmc*. La función \widehat{IICR} es la *IICR* inferida de datos genómicos.

La función *IICR* representa la *IICR teórica* de T_2 (los tiempos de coalescencia de dos genes). Este *IICR* teórico se computa bajo el modelo NSSC presentado en [13]. Para evaluar el *IICR* teórico en cualquier $t \in \mathbb{R}$, $t \geq 0$ se hace lo siguiente:

$$IICR(t) = \frac{\mathbb{P}(T_2 > t)}{f_{T_2}(t)} = \frac{1 - F_{T_2}(t)}{f_{T_2}(t)}$$

siendo F_{T_2} y f_{T_2} el *cdf* (función de distribución acumulada) y el *pdf* (función de densidad de probabilidad) de T_2 respectivamente.

(2) Representa el *factor de escala (scaling factor)*.

(3) Representa el *número de islas (number of demes)*.

(4), (5) y (6) Corresponden al *tiempo de cambio (time of change)* en el período i .

(7) Representa el *tamaño de la sub población (sub population)* en el período i .

(8) Representa la *tasa de migración (gene flow)* en el período i .

3.3 Cálculo de la distribución de los tiempos de coalescencia bajo el modelo NSSC

Partiendo de que ya existe el valor de γ (*número de eventos demográficos*) y el valor de \emptyset . Por cada $i \in \{0, \dots, \gamma\}$ tenemos la siguiente matriz de tasas (*rate matrix*):

$$Q_i = \begin{pmatrix} -c_i - M_i & -M_i & c_i \\ \frac{M_i}{n-1} & -\frac{M_i}{n-1} & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

A partir de los valores de las matrices Q_i , definimos la matriz P_t como:

$$\begin{aligned} P_t &= \left(\prod_{i=1}^{\eta} e^{(t_i - t_{i-1})Q_{i-1}} \right) e^{(t - t_{\eta})Q_{\eta}} \\ &= e^{(t_1 - t_0)Q_0} e^{(t_2 - t_1)Q_1} \dots e^{(t_{\eta} - t_{\eta-1})Q_{\eta-1}} e^{(t - t_{\eta})Q_{\eta}} \end{aligned}$$

donde:

$$\eta = \arg \max_{i \in \{0, \dots, \gamma\}} \{t_i \mid t_i < t\}$$

El *cdf* y el *pdf* de t_2 bajo el modelo NSSC pueden evaluarse utilizando la matriz P_t :

$$F_{T_2}(t) = P_t(1,3),$$

$$f_{T_2}(t) = \frac{d}{dt} F_{T_2}(t) = \frac{d}{dt} P_t(1,3)$$

3.4 Resolución del problema computacionalmente

3.4.1 Metaheurísticas

Una metaheurística se puede describir como un proceso iterativo maestro que guía y modifica las operaciones de heurísticas subordinadas para producir de forma eficiente soluciones de alta calidad. Las heurísticas subordinadas pueden ser procedimientos de alto o bajo nivel, un método de búsqueda local o simplemente un método constructivo. Aunque se suele considerar como metaheurística a toda aquella generalización de una determinada heurística para cualquier tipo de problema, en la práctica es necesario analizar detalladamente el problema a resolver para determinar cuál de ellas es la que se presupone puede tener un mayor éxito en la búsqueda de soluciones, aunque no se pueda determinar con total seguridad si se alcanzara dicho éxito [35].

- Metaheurísticas basadas en un punto

Los algoritmos metaheurísticos basados en un punto (S-Metaheurísticas) son técnicas que parten de un punto inicial y van actualizando la solución presente.

- Metaheurísticas poblacionales

Los métodos basados en población, conocidas como P-Metaheurísticas, se caracterizan por trabajar con un conjunto de soluciones (población) en cada iteración, a diferencia de los anteriores cuya referencia es un solo punto en el espacio de búsqueda. El resultado final proporcionado por este tipo de algoritmos depende principalmente de la forma en que manipula la población. El hecho de mantener varios puntos, da la posibilidad de emplear operadores que utilizan una o varias soluciones para generar los próximos puntos a visitar [36].

3.4.2 Selección de las metaheurísticas

Para dar solución al problema de optimización combinatoria planteado en el *epígrafe* 3.2, se probaron algunas de las metaheurísticas pertenecientes a los dos grupos anteriores:

- Basadas en un punto:

1. Escalador de colinas
2. Recocido simulado

- Poblacionales:

1. Evolución Diferencial
2. Optimización por enjambre de partículas

Según el Teorema *No Free Lunch* (NFL) [37] se conoce que es imposible encontrar algún algoritmo metaheurístico que sea el mejor en comportamiento para un tipo de problema en específico. Incluso dentro del mismo tipo de problema resulta difícil establecer la superioridad de una metaheurística. Cuando no se dispone de resultados teóricos que permitan comparar el comportamiento de los algoritmos, se debe prestar especial atención al análisis de los resultados empíricos⁶.

En el desarrollo de este proyecto se realizaron comparaciones experimentales entre los algoritmos metaheurísticos mencionados antes de asumir la superioridad de uno sobre otros. A cada uno de los algoritmos se le fueron variando sus parámetros (en dependencia del algoritmo) hasta ir obteniendo mejores soluciones. Aunque lo anterior fue de utilidad, es de suma importancia mencionar que los mejores

⁶ Que está basada en la experiencia y en la observación de los hechos

resultados fueron obtenidos al reducir el dominio de los parámetros de la función objetivo, lo cual influyó notablemente en la disminución de la distancia entre las funciones. Los algoritmos *Optimización por Enjambre de Partículas* y *Evolución Diferencial* (poblacionales) fueron los más *eficaces* (lograron obtener los resultados deseados) en el problema del ajuste de curvas tratado en el presente trabajo, siendo el último mencionado el mejor de ambos. El término *eficiencia* no fue de gran impacto pues la duración de ejecución máxima de cada metaheurística no excedió los dos minutos.

Las siguientes tablas muestran algunos de los resultados obtenidos a partir de un experimento realizado entre las curvas presentadas a continuación, con una distancia inicial de 247516600.1589 unidades:

1. Curva correspondiente al modelo PSMC extraída del fichero '*experiment_1.psmc*'
2. Curva correspondiente al modelo NSSC construida a partir del escenario *'{{'time': 0, 'n':15, 'M':1, 'c':1}, {'time':10, 'n':15, 'M':50, 'c':1}, {'time':50, 'n':15, 'M':1, 'c':1}}'*

NOTA IMPORTANTE: Por cada configuración de los parámetros reflejado en las pruebas, el algoritmo en estudio fue ejecutado entre 5 y 10 veces.

1. ALGORITMO: Escalador de colinas

Prueba	Parámetros				Distancia obtenida
	max_val	max_iters	restarts	curve	
1	8	50	0	False	2725702094.7835
2	5	30	1	True	5106789366.6128
3	10	100	2	False	1179027108.9410

Tabla 10. Prueba del algoritmo Escalador de colinas

CONCLUSIÓN: Ninguna de las pruebas mejoró el valor de la distancia inicial.

2. ALGORITMO: Recocido simulado

Prueba	Parámetros						Distancia obtenida
	max_val	max_iters	init_t	exp_c	min_t	max_att	
1	8	10 000	1.0	0.005	0.001	10	5352178894.9399
2	5	5 000	1.5	0.008	0.002	20	7781347648.1124
3	10	12 000	0.5	0.001	0.003	30	5351783701.1793

Tabla 11. Prueba del algoritmo Recocido simulado

CONCLUSIÓN: Ninguna de las pruebas mejoró el valor de la distancia inicial.

3. ALGORITMO: Evolución diferencial

Prueba	Parámetros				Distancia obtenida
	maxiter	popsiz	mutation	recombination	
1	1000	15	0.5	0.7	57626296.1468
2	500	20	0.8	0.5	60786459.7643
3	800	10	0.2	0.9	62756765.9006

Tabla 12. Prueba del algoritmo Evolución diferencial

CONCLUSIÓN: Las pruebas disminuyeron la distancia inicial en más de un 65%.

4. ALGORITMO: Optimización por enjambre de partículas

Prueba	Parámetros		Distancia obtenida
	maxiter	num_particles	
1	80	20	184589765.9987
2	120	15	190743579.7121
3	50	25	201156732.3568

Tabla 13. Prueba del algoritmo Optimización por enjambre de partículas

CONCLUSIÓN: Todas las pruebas disminuyeron la distancia inicial entre un 20% y un 25% aproximadamente.

3.5 Arquitectura

Los patrones arquitectónicos o patrones de arquitectura, también llamados arquetipos, ofrecen soluciones a problemas de arquitectura en ingeniería de software [38]. Dan una descripción de los elementos y el tipo de relación que tienen junto a un conjunto de restricciones sobre cómo pueden ser usados. Un patrón arquitectónico expresa un esquema de organización estructural esencial para un sistema de software, que consta de subsistemas, sus responsabilidades e interrelaciones. En comparación con los patrones de diseño, los patrones arquitectónicos tienen un nivel de abstracción mayor.

Aunque un patrón arquitectónico comunica una imagen de un sistema, no es una arquitectura como tal. Un patrón arquitectónico es más un concepto que captura elementos esenciales de una arquitectura de software. Muchas arquitecturas diferentes pueden implementar el mismo patrón y por lo tanto compartir las mismas características [38].

Uno de los aspectos más importantes de los patrones arquitectónicos es que encarnan diferentes atributos de calidad. Por ejemplo, algunos patrones representan soluciones a problemas de rendimiento y otros pueden ser utilizados con éxito en sistemas de alta disponibilidad.

- Patrón arquitectónico utilizado (Modelo - Vista - Controlador)

Modelo-vista-controlador (MVC) [38] es un patrón de arquitectura de software, que separa los datos y la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC

propone la construcción de tres componentes distintos que son: el modelo, la vista y el controlador. Por un lado define los componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

La siguiente figura muestra la arquitectura de un proyecto basado en el patrón MVC.

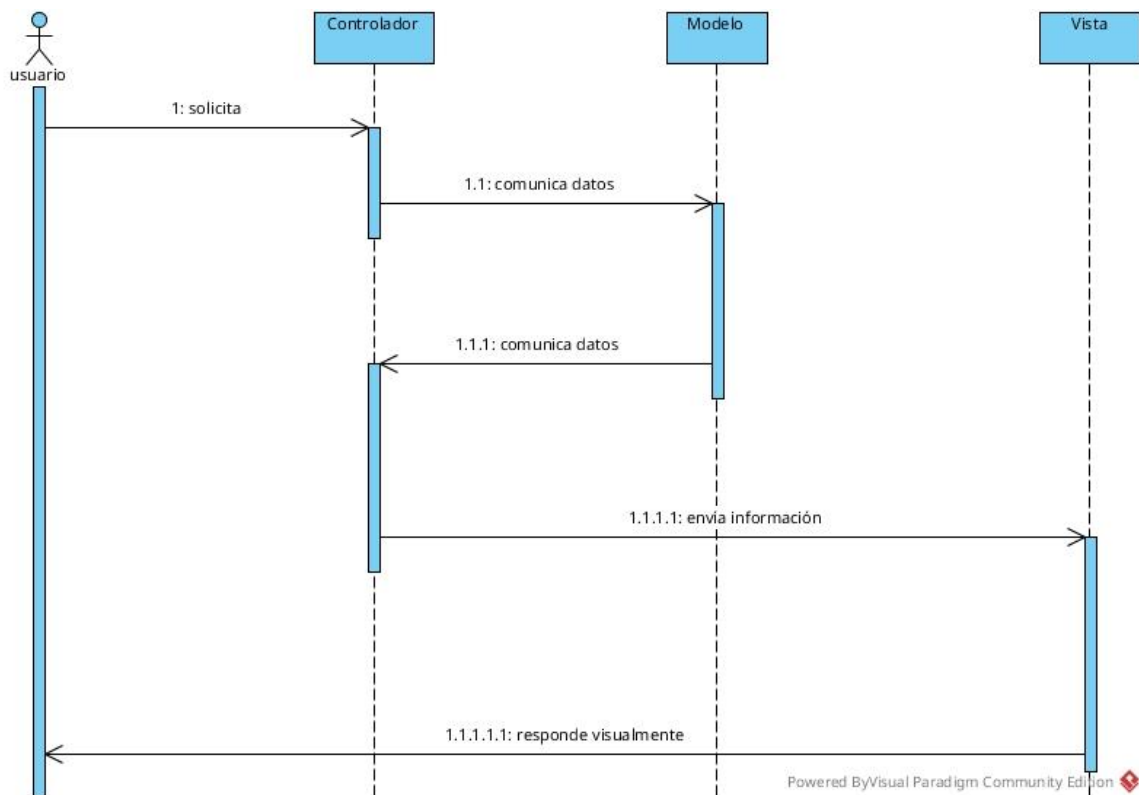


Figura 8. Arquitectura Modelo - Vista - Controlador

Para describir la arquitectura desde otro punto de vista se decidió utilizar también la estructuración en capas basada en responsabilidades:

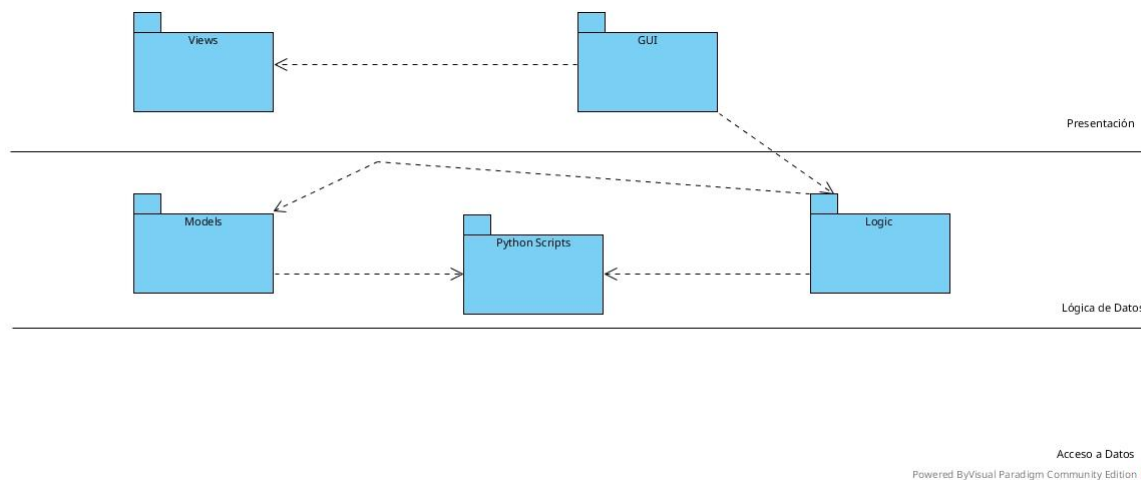


Figura 9. Estructuración en capas usando un enfoque de responsabilidades

Nótese que la capa de acceso a datos se encuentra vacía pues esta aplicación no accede a ninguna base de datos. La única información persistente existe en forma de ficheros y es la propia lógica quien la manipula.

3.6 Diagramas de Clases de Diseño

- Correspondiente a historias de usuario

Los diagramas de clases de diseño de las historias de usuario reflejadas en el capítulo anterior son relativamente similares, de forma que sólo se reflejará el diagrama correspondiente a *cargar los ficheros de los modelos PSMC, MSMC y NSSC*. Dicha historia de usuario es de suma importancia para la posterior manipulación de los datos.

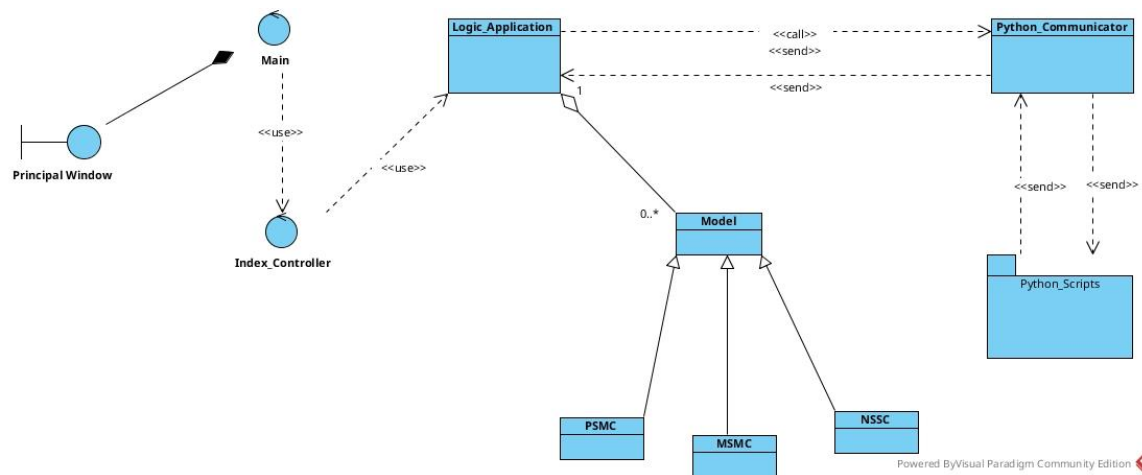


Figura 10. Diagrama de clases de diseño de cargar ficheros

- Correspondiente a la aplicación en general. Clases y sus relaciones

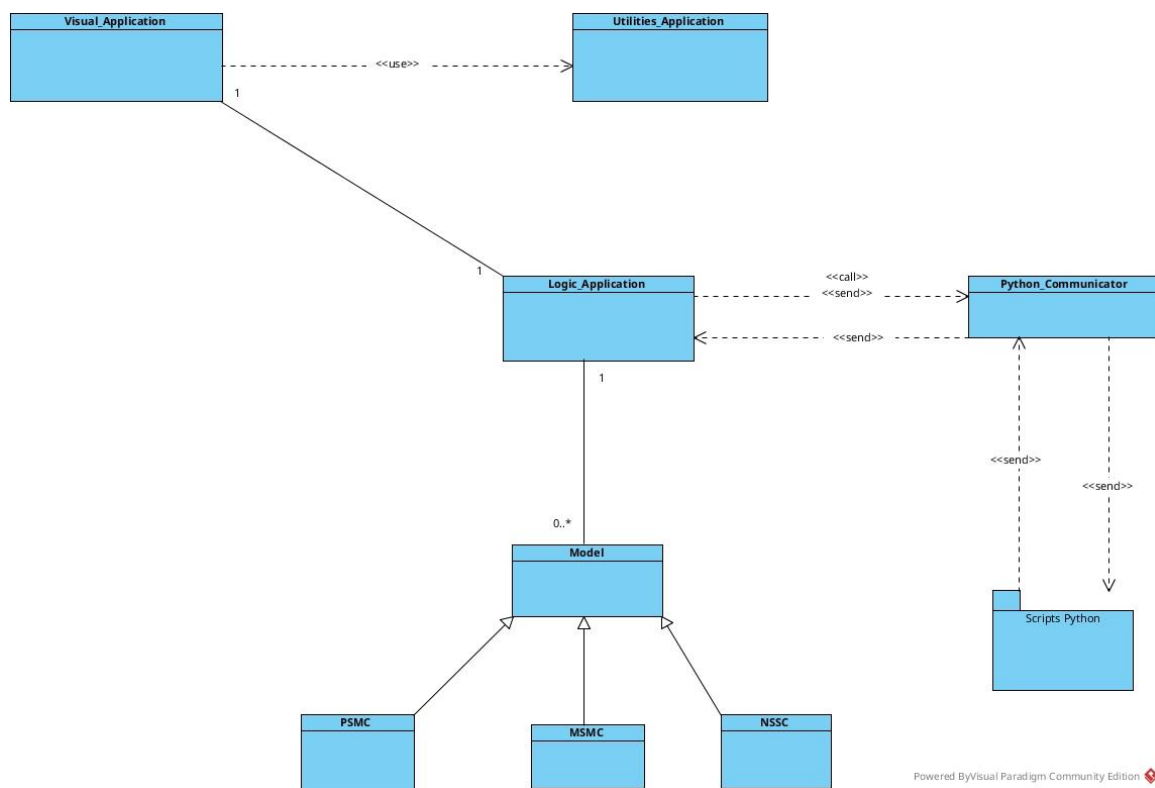


Figura 11. Diagrama de clases de la aplicación

Para una mejor comprensión del diagrama de clases anterior, se decidió mostrar a continuación cada clase independiente con sus atributos y métodos.

- Clases independientes con sus métodos y atributos

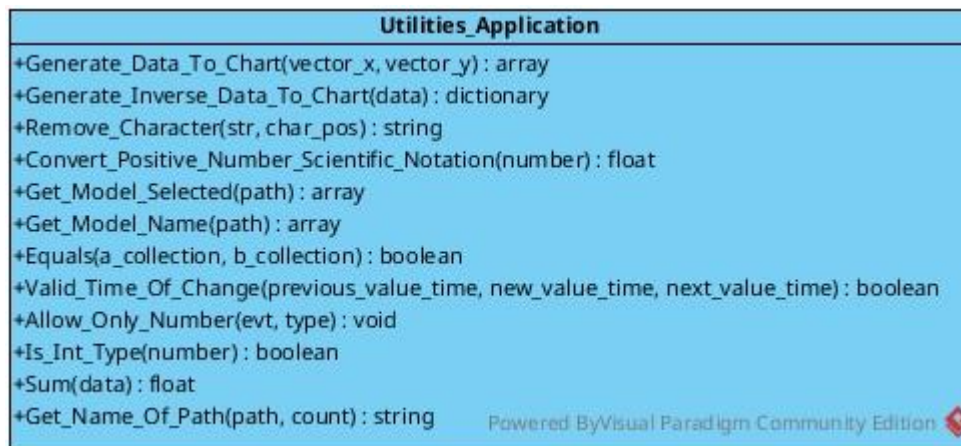


Figura 12. Clase *Utilities_Application*

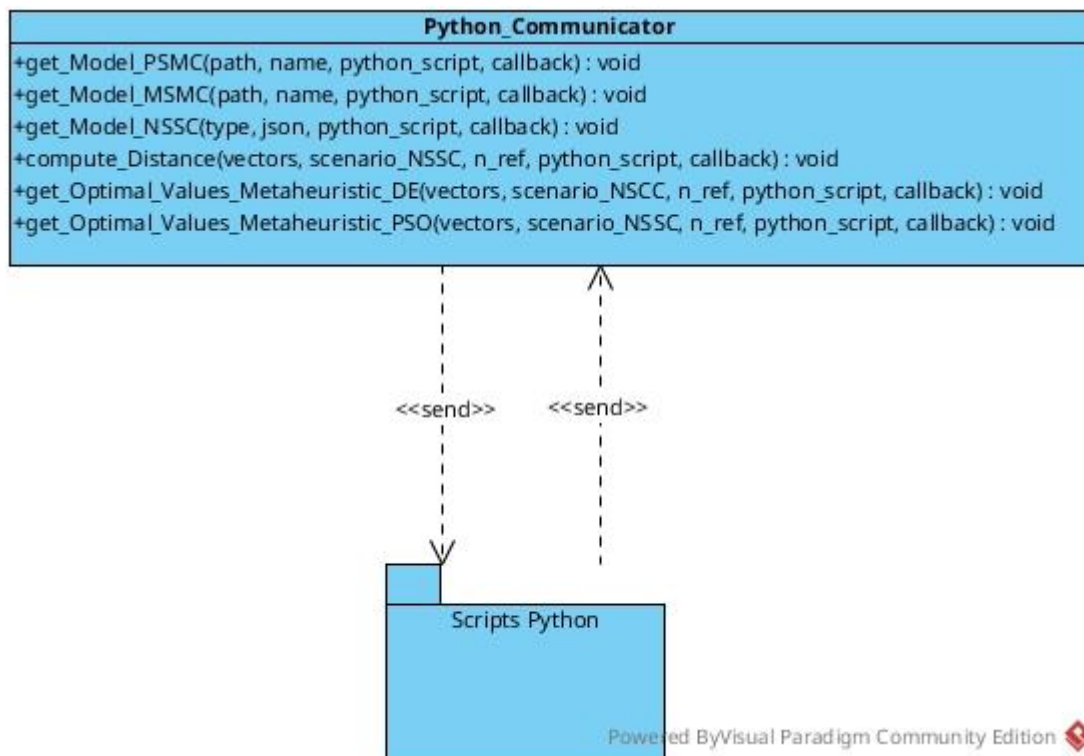


Figura 13. Clase *Python_Communicator* y paquete de scripts de python

Visual_Application
-canvas : html tag -chart : jquery plugin +Index_Of(name_graphic) : int +Get_Graphic(name_graphic) : Model (visual) +Get_Random_Color() : string +Select_Function(target, select_function, slider_mu) : void +Add_Model_Compute_Distance(control, name) : void +Visualize_Commune_Function(element_scale_by_default, psmc_msmc_nssc_model_control) : void +Visualize_PSMC() : void +Visualize_MSMC() : void +Visualize_NSSC() : void +Visualize_Application() : void +Visualize_Element_Of_List(name, model, color) : void +Update_NSSC(nssc_function) : void +Update_Colors(funcnt, color, legend_color) : void +Update_Scale_PSMC_MSMC(original_function, mu, s) : void +Update_Scale_NSSC(original_function, n_ref) : void +Reset_Scales(funcnt) : void +Reset_All_Scales() : void +Restart_NSSC_Options() : void +Get_Parameters(name) : array +Change_Axis_Scale(new_scale, axis) : void +Reset_Zoom() : void +Initialize_Information_Of_Functions() : void +Visualize_Information_Of_Functions(funcnt) : void +Show_Panel_Distance() : void +Hide_Panel_Distance() : void +Load_Principal_Window_Data(name, scenario, callback) : void +Show_Delete_Window(message, callback) : void +Show_Error_Window(message) : void +Show_Information_Window(message) : void +Delete_Function(name, target) : void +Show_Distance() : void +Show_Optimal_Values_Metaheuristics(callback) : void +Delete_Function_Metaheuristic_List(name) : boolean +Fill_Initial_Data_Vector(value, order) : array +Fill_Initial_Data_Matrix(name, order) : multi array +Initialize_Matrix(matrix, data) : void +Configuration_Vector() : void +Configuration_Matrix(matrix, order) : void +Add_Show_Time_Deme_Sizes(html, order, matrix_collection, id) : void +Build_Visual_Scenario(time_size, nssc_scenario, matrix_collection, deme_vector_collection, sampling_vector, order, type, number_of_events) : void +Fill_Deme_Vector(vector, count_demes, value) : void +Fill_Deme_Matrix(matrix, count_demes, value) : void +Add_Deme(count_demes, order, deme_vector_collection, sampling_vector, matrix_collection, type) : void +Delete_Deme(count, order, deme_vector_collection, sampling_vector, matrix_collection, type) : void +Build_Visual_Scenario_With_Sliders(nssc_scenario, matrix_collection, deme_vector_collection, sampling_vector, order, type, number_of_events) : void +Configuration_Sliders(type, matrix_collection, deme_vector_collection, sampling_vector, count) : void +Update_Slider(value, type, slider, input) : void +Reset_Slider(type, slider) : void +Hide_Corner_Jexcel() : void +Restart_Edit_Container() : void
Powered ByVisual Paradigm Community Edition 

Figura 14. Clase Visual_Application

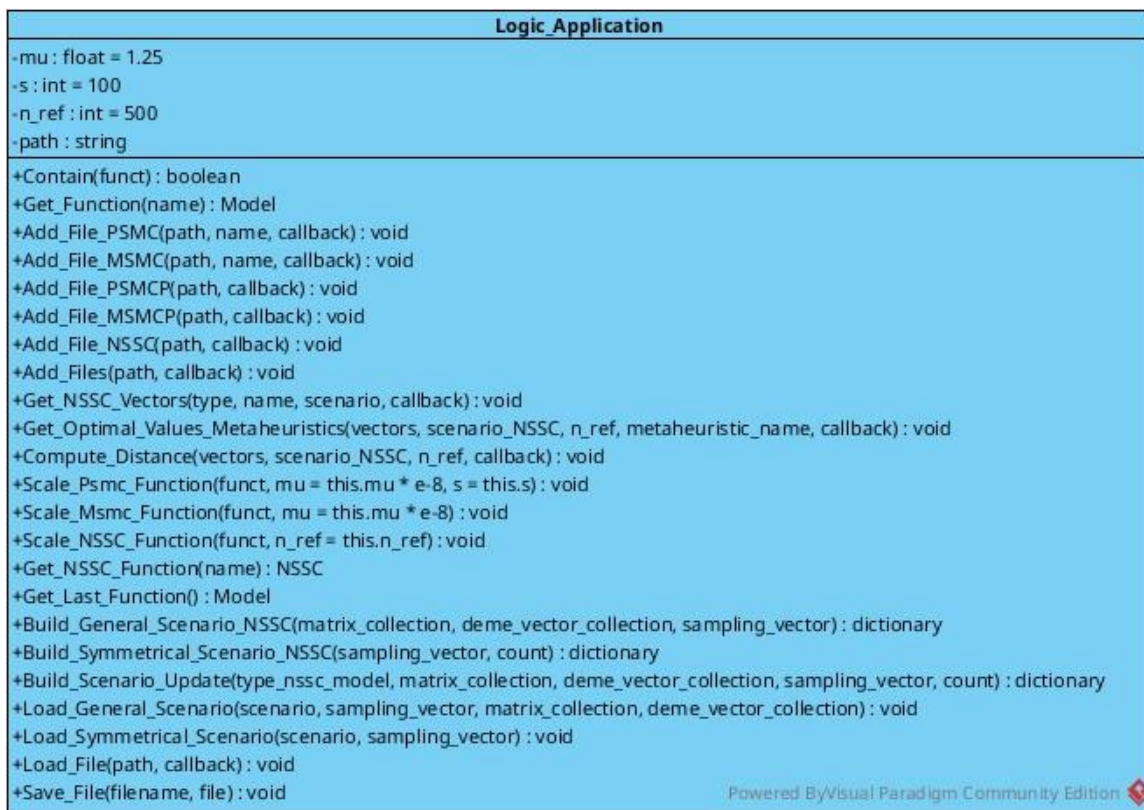


Figura 15. Clase Logic_Application

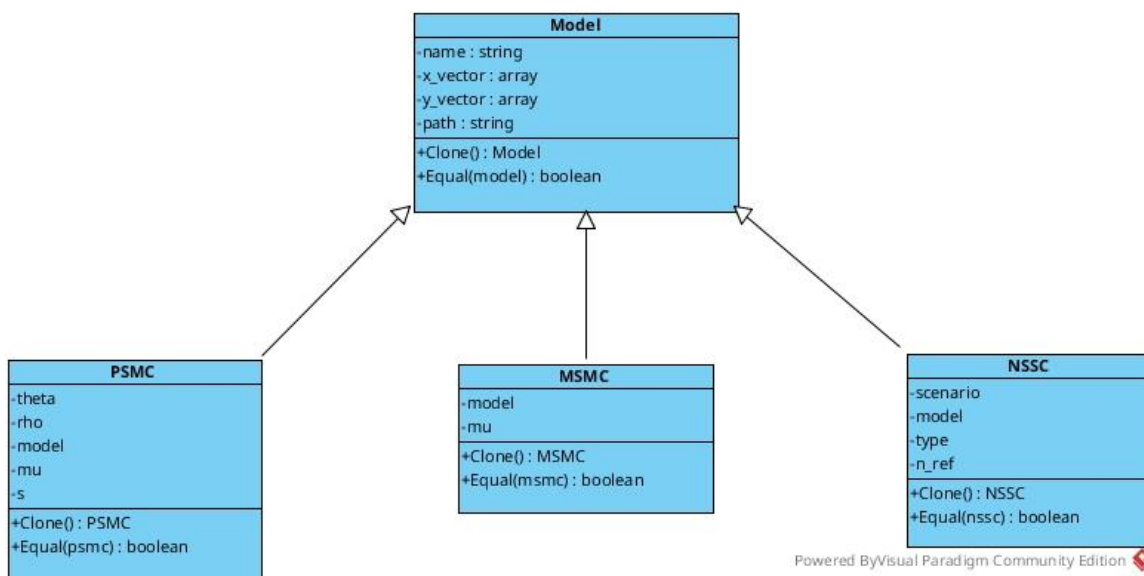


Figura 16. Clase Model y sus respectivas herencias

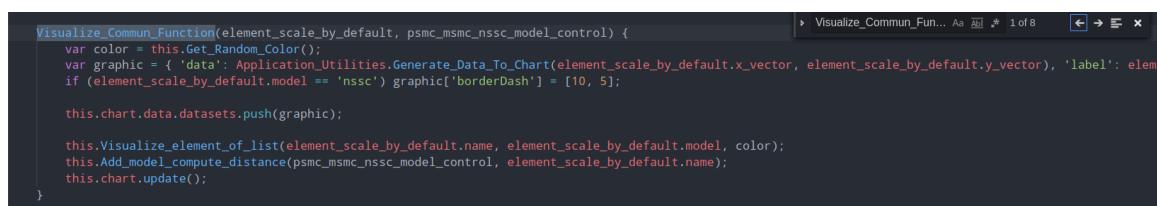
Véase como se incluyen dos clases adicionales “*Visual_Application()* y *Application_Uilities()*”, las cuales son imprescindibles para el correcto funcionamiento de TODAS las historias de usuario.

3.7 Principios de diseño

Para ayudar a desarrollar programas robustos, mantenibles y que se puedan modificar, existen varios principios de diseño que contribuyen a desarrollar software de este tipo. En la aplicación presentada en este trabajo, se tuvieron en cuenta algunos de estos principios de acuerdo a objetivos concretos y específicos.

- DRY (*del inglés Don't repeat yourself*)

DRY [39] es un principio básico y muy importante a tener en cuenta en el desarrollo de un software. Además, es muy simple de entender. Se basa en la NO repetición de código. El código duplicado es propenso a generar errores y muy difícil de mantener. La solución en sentido general es la de extraer código repetido para encapsularlo y así tenerlo localizado en una sola función. Por ejemplo:



```

Visualize_Communications_Function(element_scale_by_default, psmc_msmc_nssc_model_control) {
    var color = this.Get_Random_Color();
    var graphic = { 'data': Application_Uilities.Generate_Data_To_Chart(element_scale_by_default.x_vector, element_scale_by_default.y_vector), 'label': element_scale_by_default.name };
    if (element_scale_by_default.model == 'nssc') graphic['borderDash'] = [10, 5];

    this.chart.data.datasets.push(graphic);

    this.Visualize_element_of_list(element_scale_by_default.name, element_scale_by_default.model, color);
    this.Add_model_compute_distance(psmc_msmc_nssc_model_control, element_scale_by_default.name);
    this.chart.update();
}

```

Figura 17. Función *Visualize_Communications_Function()*

A continuación se observa como el método *Visualize_Communications_Function()* (Fig. 16), se utiliza en otras funciones. De esta manera se pone en práctica el principio DRY descrito anteriormente:

```

Visualize_PSMC() {
  if ((this.logic_application.functions_collection.length > this.chart.data.datasets.length)) {
    var last_element_add = this.logic_application.Get_Last_Function();
    var element_scale_by_default = last_element_add.Clone();

    this.logic_application.Scale_Psmc_Function(element_scale_by_default, element_scale_by_default.Mu * 1e-8, element_scale_by_default.S);
    // last_element_add.Mu = element_scale_by_default.Mu / 1e-8;

    this.Visualize_Communs_Function(element_scale_by_default, $('#psmc-msmc-model'));
  }
}

```

Figura 18. Función Visualize_PSMC()

```

Visualize_MSMC() {
  if ((this.logic_application.functions_collection.length > this.chart.data.datasets.length)) {
    var last_element_add = this.logic_application.Get_Last_Function();
    var element_scale_by_default = last_element_add.Clone();

    this.logic_application.Scale_Msmc_Function(element_scale_by_default, element_scale_by_default.Mu * 1e-8);
    // last_element_add.Mu = element_scale_by_default.Mu / 1e-8;

    this.Visualize_Communs_Function(element_scale_by_default, $('#psmc-msmc-model'));
  }
}

```

Figura 19. Función Visualize_MSMC()

```

Visualize_NSSC() {
  if ((this.logic_application.functions_collection.length > this.chart.data.datasets.length)) {
    var last_element_add = this.logic_application.Get_Last_Function();
    var element_scale_by_default = last_element_add.Clone();

    this.logic_application.Scale_NSSC_Function(element_scale_by_default, element_scale_by_default.N_ref);

    this.Visualize_Communs_Function(element_scale_by_default, $('#nssc-model'));
  }
}

```

Figura 20. Función Visualize_NSSC()

```

Visualize_Application() {
  if (this.chart.data.datasets.length > 0) {
    for (const element of this.chart.data.datasets) {
      this.Delete_Function_Metaheuristic_List(element.label);
    }
    this.chart.data.datasets = [];
    $('#btn-delete').parents('.listview_item').remove();
  }

  for (const element of this.logic_application.functions_collection) {
    var funct_clone = element.Clone();

    if (element.model == 'psmc' || element.model == 'msmc') {
      if (element.model == 'psmc') this.logic_application.Scale_Psmc_Function(funct_clone, funct_clone.Mu * 1e-8, funct_clone.S);
      else this.logic_application.Scale_Msmc_Function(funct_clone, funct_clone.Mu * 1e-8);

      this.Visualize_Communs_Function(funct_clone, $('#psmc-msmc-model'));
    }
    else {
      this.logic_application.Scale_NSSC_Function(funct_clone, funct_clone.N_ref);
      this.Visualize_Communs_Function(funct_clone, $('#nssc-model'));
    }
  }
}

```

Figura 21. Función Visualize_Application()

- KISS (*del inglés Keep It Simple Stupid*)

Es necesario lograr que el diseño de un programa sea lo más sencillo posible [39]. En la aplicación desarrollada se tuvo en cuenta este principio, tratando de evitar la

complejidad innecesaria como norma general.

Los epígrafes 3.4 y 3.5 muestran la simplicidad del diseño de la aplicación enfocada en la arquitectura del software y el diseño de las clases que incluye.

- Principio SoC (del inglés separation of concerns)

El principio SoC [40] aboga por la separación de los diferentes aspectos de una aplicación, por ejemplo, la capa de negocio, la capa de presentación, etc. De lo anterior podemos concluir que un buen ejemplo lo constituye la estructuración en capas basada en responsabilidades puesto en práctica en el presente proyecto.

- Principio de Hollywood “No nos llames, nosotros te llamaremos”

En el diseño del software en cuestión se tuvo en cuenta este principio. Por ejemplo, existe una clase llamada Python_Communicator implementada en JavaScript, la cual contiene los métodos encargados de comunicarse con las funciones de Python, y en ningún caso el código Python puede interactuar con dicha clase. En resumen; *no nos llames (código Python), nosotros te llamaremos (clase de JavaScript)*.

3.7.1 Interfaz de usuario

El aspecto visual de una aplicación, es un tema más a tener en cuenta para ofrecer a los usuarios una imagen de seriedad y generar en ellos una sensación de confianza. Existe una clara vinculación entre la credibilidad y el grado de confianza percibidos por los usuarios de un software. Es muy importante cuidar la primera impresión, la estética de una aplicación transmite unos valores que son percibidos por el usuario en un instante. Por tanto, se debe mostrar un diseño fuerte y convincente para ganar en los aspectos antes mencionados. La aplicación desarrollada en este trabajo es visualmente atractiva para el usuario, ya que se busca limpieza en las pantallas y

simplicidad. Esto se logra usando colores homogéneos en todo momento, alineando todos los componentes y tratando de no cargar la vista:



Figura 22. Ventana principal de la aplicación

3.8 Tratamiento de errores

La aplicación evita que se produzcan errores desde la interfaz no permitiendo al usuario realizar acciones cuando no tienen sentido. También implementa la validación de datos teniendo en cuenta el tipo y el rango, entre otros. De esta manera impide la entrada de valores erróneos.

Otros posibles errores fueron tratados a través de notificaciones. Estos son mostrados al usuario en forma de mensajes de advertencia o error, explicando lo más detallado posible la causa de origen.

Las clases *Application_Uilities()* y *Visual_Application()* intervienen de manera crucial en el proceso de validación del software.

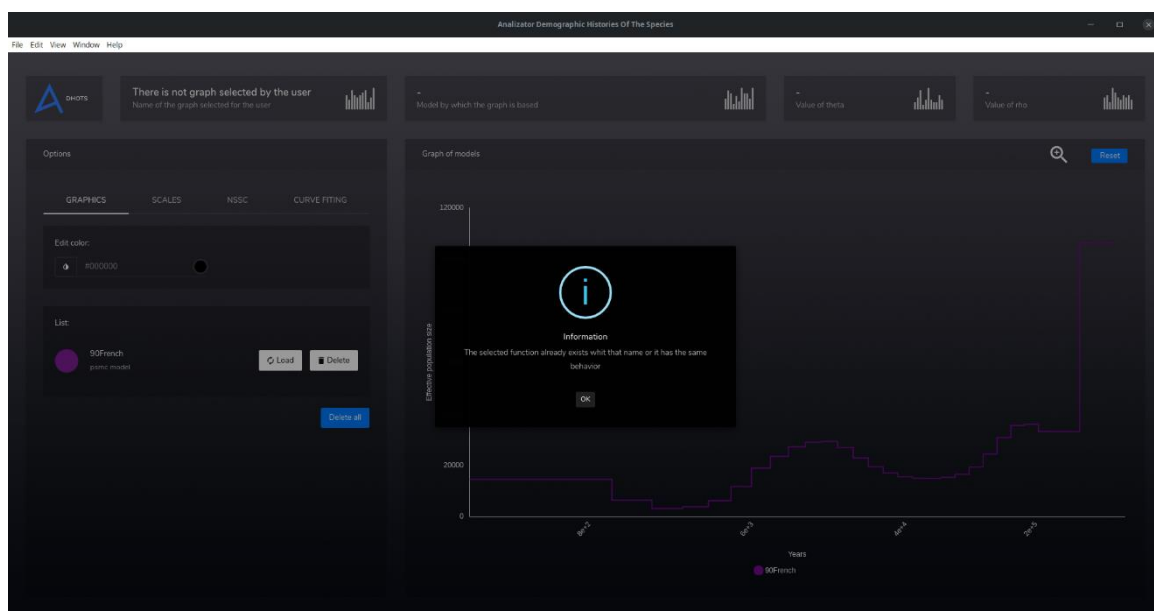


Figura 23. Información lanzada al intentar cargar un archivo existente

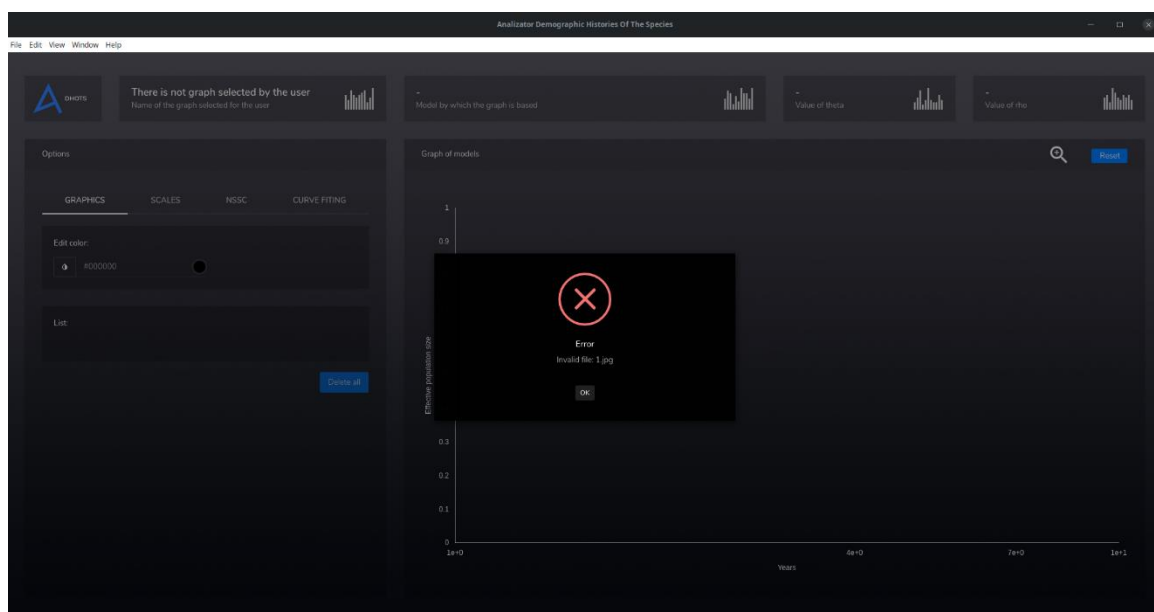


Figura 24. Error lanzado al intentar cargar un archivo no válido

3.9 Análisis de costos y beneficios de la solución propuesta

La técnica de *Costo / Beneficio* tiene como objetivo fundamental proporcionar una medida de rentabilidad sobre un proyecto. Esto mediante el análisis de los costos previstos con los beneficios esperados en la realización del mismo.

3.9.1 Costos

El costo de la aplicación desarrollada gira principalmente alrededor del factor tiempo y algunos derivados de la curva de aprendizaje. Otros como los de carácter financiero; dígame gastos por adquisición de hardware y software, gastos de instalaciones, de mantenimiento, consultoría, salarios, entre otros, son prácticamente nulos.

El autor de este trabajo realizó el software sin retribución alguna desde el punto de vista económico. No obstante, suponiendo que una empresa nacional hubiese asumido el proyecto, el costo del mismo podría verse del siguiente modo. Suponiendo que el salario básico de un ingeniero informático es de \$450 al mes y el proyecto tuvo un tiempo de desarrollo aproximado de 5 meses, se puede concluir entonces que el costo ascendería alrededor de los \$2250.

3.9.2 Beneficios

Los beneficios de esta herramienta se cuantifican principalmente de manera *intangibile*. La aplicación constituye un aporte invaluable para la ciencia en el campo de la biología, en particular, en genética de poblaciones. Entre algunos de los principales beneficios esperados se encuentran:

1. La detección más precisa de eventos de reducción de tamaño efectivo (*bottlenecks*).
2. Una mejor inferencia del tiempo de coalescencia a lo largo de todo el ADN.
3. Más precisión en la localización de sitios con alta probabilidad de recombinación (*recombination hotspots*).

4. Obtener con más certeza conocimientos sobre la evolución de la población en términos de diversidad genética.

De lo anterior visto, podemos concluir que resulta factible el desarrollo del software, fundamentalmente porque el costo es despreciable en comparación al alto beneficio que se espera obtener. Esto sobresale aún más cuando se trata de aportes a la ciencia moderna.

3.10 Conclusiones parciales del capítulo

La descripción formal del problema de optimización tratado en el capítulo anterior, ayudó al autor de este proyecto a tener una visión más clara de lo que debía realizar para enfrentar la automatización del ajuste de curvas.

Los algoritmos metaheurísticos fueron la solución al problema planteado. Durante su estudio e implementación, destacaron lo importante que es realizar experimentos que puedan demostrar la superioridad de un método sobre otros. En el problema del ajuste de curvas, se concluyó con elementos concretos, la superioridad de los algoritmos poblacionales sobre los basados en un punto.

La arquitectura del software y los diagramas de clases de diseño, aportaron una visión general sobre algunos aspectos de importancia. Las funcionalidades de las clases presentadas tienen una alta relación entre ellas, lo que mantiene el enfoque a su único propósito. Por otra parte, el bajo acoplamiento evita la fuerte dependencia que pudiese existir. La legibilidad, mantenibilidad, reusabilidad y alta capacidad de pruebas son algunos de los beneficios que desencadena el principio de “alta cohesión y bajo acoplamiento” en la programación orientada a objetos.

La manera en que se manipuló el tratamiento de errores asegura un mejor funcionamiento del software y minimiza el mal uso de este para usuarios de poca experiencia.

En sentido general, el desarrollo del proyecto fue totalmente factible, aspecto que quedó bien demostrado. Los costos de cualquier índole quedaron muy disminuidos ante el aporte a la ciencia, en especial, a la genética de poblaciones.

Conclusiones

El *proceso actual* de reconstrucción de las historias demográficas de las distintas especies es objeto de estudio para muchos investigadores en la actualidad. Dicho proceso forma parte de una investigación que trata de corregir algunos defectos en los modelos teóricos utilizados hasta el momento. La inclusión del modelo NSSC como posible solución a los errores encontrados, trajo como consecuencia la problemática que dio origen al software presentado en este trabajo.

Dicha aplicación, implementa las funcionalidades que posibilitan poner en práctica las teorías demostradas y aceptadas por los investigadores del Instituto de Matemáticas Aplicadas de Toulouse, Francia. De esta forma, el cumplimiento del objetivo general expuesto en el primer capítulo queda evidenciado, considerando además que la propuesta ya está siendo usada y probada por los principales interesados. Por tanto, finalmente puede concluirse que:

- Para resolver las deficiencias, los métodos y herramientas más efectivos fueron los seleccionados para el desarrollo, demostrándose que Scrum es una metodología adecuada para proyectos con un alcance y envergadura similar al descrito en este trabajo.
- La continua entrega de un producto funcional al Dueño del Producto, trajo como consecuencia la obtención de funcionalidades realmente ajustadas a las necesidades de los interesados en un tiempo considerablemente corto.
- Las pruebas realizadas favorecieron definitivamente la confiabilidad de la aplicación, así como también influirán posteriormente al mantenimiento de la misma.

- La implementación y selección de los algoritmos metaheurísticos que mejores se ajustaron al problema del ajuste de curvas, evidenciaron en el caso de este problema en particular, la importancia del análisis de los resultados empíricos.
- Los diagramas de clases de diseño contribuyeron a aumentar el grado de comprensión de la estructura del proyecto. Estos junto a los diagramas de arquitectura, dieron la visión necesaria para la corrección de errores y ayudarán al posterior desarrollo y mantenimiento de la aplicación.
- Por último, se demostró la factibilidad de la herramienta desarrollada y se espera que sea de gran utilidad. Hoy en día ya ha recibido críticas positivas en países de Europa como Francia y Alemania, donde ha tenido una gran aceptación entre los expertos en el área de la genética de poblaciones.

Recomendaciones

Para versiones posteriores del sistema se recomienda especialmente:

- Un manual de usuario para la manipulación del software. Es recomendable que sea realizado en idioma inglés o francés puesto que, en el futuro cercano, la aplicación se estará probando fundamentalmente en los laboratorios de Evolución y Diversidad Biológica de la Universidad de Paul Sabatier, Francia.
- Continuar con el estudio de las metaheurísticas enfocadas en el problema del ajuste de curvas. Esto permitirá encontrar otros algoritmos que quizás mejoren los resultados obtenidos.

Referencias bibliográficas

- [1] Richard Halliburton. *Introduction to Population Genetics*. Prentice Hall, 2004.
- [2] Antonio Barbadilla. *La Genética de Poblaciones*. Departamento de Genética y Microbiología. Universidad Autónoma de Barcelona, 2011.
- [3] Massimo Livi Bacci, Ariel Historia. *Introducción a la demografía*. ISBN 978-84-344-6573-9, 2007.
- [4] Magnus Nordborg. *Coalescent Theory*. Department of Genetics, Lund University, March 24, 2000.
- [5] Hudson, R. R. *Estimating the recombination parameter of a finite population model without selection*. Genet. Res. 1987.
- [6] Green, E.D., Watson, J.D. & Collins, F.S. *Human Genome Project: Twenty-five years of big biology*. Nature. 2015.
- [7] Ansorge, W.J. *Next-generation DNA sequencing techniques*. New Biotechnology. 2009.
- [8] Heng Li and Richard Durbin. *Inference of human population history from individual whole - genome sequences*. Macmillan Publishers, 2011.
- [9] Stephan Schiffels and Richard Durbin. *Inferring human population size and separation history from multiple genome sequences*. May 21, 2014.
- [10] O Mazet, Willy Rodríguez, S Grusea, S Boitard and L Chikhi. *On the importance of being structured: instantaneous coalescence rates and human evolution-lessons for ancestral population size inference?* Macmillan Publishers, 2015.
- [11] Willy Rodríguez, *Estimación del tamaño efectivo de la población a partir del ADN de un único individuo. El modelo PSMC*. Institut de Mathématiques de Toulouse, Université de Toulouse & CNRS, France, 2013.

- [12] Olivier Mazet, Willy Rodríguez, Lounes Chikhi. *Demographic inference using genetic data from a single individual: Separating population size variation from population structure*. Macmillan Publishers. 2015.
- [13] Willy Rodríguez, Olivier Mazet, Simona Grusea, Armando Arredondo, Josue Corujo, Simon Boitard and Lounes Chikhi. *The IICR and the non-stationary structured coalescent: towards demographic inference with arbitrary changes in population structure*. September, 2018.
- [14] Willy Rodríguez, Olivier Mazet, Simona Grusea, Armando Arredondo, Josue Corujo, Simon Boitard and Lounes Chikhi. **Supplementary information for:** *The IICR and the non-stationary structured coalescent: towards demographic inference with arbitrary changes in population structure*. September, 2018.
- [15] JavaScript Language. [En línea]. Disponible en:
<http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf>
- [16] Eich, Brendan. *ECMAScript Harmony*. (August 2016)
- [17] David Goodger. *Code Like a Pythonista: Idiomatic Python*. January 9. 2012.
- [18] Sawicki, Kevin. *Atom Shell is now Electron*. April 23. 2015.
- [19] J. Leoliger, *Version Control with Git, First ed.* O'Reilly. 2009.
- [20] *The Alliance, The Agile Manifesto*. [En línea]. Disponible en:
<http://www.agilealliance.org/the-alliance/the-agile-manifesto>
- [21] K. Schwaber y J. Sutherland. *Scrum Guide*. 2013.
- [22] M. Cohn. *User Stories Applied: For Agile Software Development*. Addison Wesley, 2004.
- [23] BonitaSoft's. *Bonita Open Solution 5.2: An Essential Toolkit BPM*. (13/05/2017)
[En línea]. Disponible en:
<http://eweek.com/enterprise-apps/bonitasoft-s-bonita-open-solution-5.2-an-essential-toolkit-for-bpm>

- [24] Jeremy LIPP. *Be Efficient: Bonitasoft Introduces New Bonita BPM 6 Platform*. 05/06/2013.
- [25] Ishikawa, Kaoru. *Introducción al control de la calidad*. Ediciones Díaz de Santos. (28/10/1994)
- [26] MathWord. *Fibonacci number*. [En línea]. Disponible en:
<http://mathworld.wolfram.com/FibonacciNumber.html>
- [27] Roy Osherove. *The Art of Unit Testing, Second Edition*. 2014 by Manning Publications Co. All rights reserved.
- [28] Unit testing framework. [En línea]. Disponible en:
<https://docs.python.org/3/library/unittest.html>
- [29] Mocha. [En línea]. Disponible en:
<https://mochajs.org/>
- [30] L. C, UML y Patrones. *Introducción al análisis y diseño orientado a objetos*. ed. Pearson, 1999.
- [31] A. a. W.-S. L. Antoniou. *Practical Optimization: Algorithms an Engineering Applications*. Springer Science+Business Media, LLC., 2007.
- [32] B. P. J. A. e. a. Melián, *Metaheurísticas: una visión global*. Revista Iberoamericana de Inteligencia Artificial, no 19, pp. 7 -28, 2003.
- [33] A. Duarte, M. Gallego y F. Gortázar. *Optimización en sistemas de dirección*. GAVAB (www.gavab.es), 2010.
- [34] R. Martí. *Algoritmos Heurísticos en Optimización Combinatoria*. Revista Iberoamericana de Inteligencia Artificial, pp. 123 -130, 2003.
- [35] C. Blum. *Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison*. ACM Computing Surveys, 2003.

- [36] A. Rosete. *Una solución flexible y eficiente para el trazado de grafos basada en el Escalador de Colinas Estocástico*. La Habana: FACULTAD DE INGENIERÍA INDUSTRIAL, CEIS, ISPJAE, 2000.
- [37] Wolpert, D.H., Macready, W.G. *No free lunch theorems for optimization*. IEEE Transactions on Evolutionary Computation. Vol. 1. No. 1. pp. 67–82. 1997.
- [38] Jonathan Kaplan, William Crawford. *J2EE Design Patterns*. September 2003.
- [39] Ludwin Barbin. *Software design principles*. August 11. 2013.
- [40] Robert C. Martin. *Agile Software Development, Principles, Patterns, and Practices*. October 1. 2002.

Glosario de términos y siglas

Genética de poblaciones: La genética de poblaciones es la rama de la genética cuyo objetivo es describir la variación y distribución de la frecuencia alélica para explicar los fenómenos evolutivos. Así es sentada definitivamente dentro del campo de biología evolutiva.

Demografía: La demografía es la ciencia que estudia las poblaciones humanas, su dimensión, estructura, evolución y características generales. Estadísticamente estudia la estructura y la dinámica de las poblaciones, así como los procesos concretos que determinan su formación, conservación y desaparición. Tales procesos son los de fecundidad, mortalidad y migración: emigración e inmigración.

Modelo oculto de Markov: Un modelo oculto de Markov o HMM (por sus siglas del inglés, Hidden Markov Model) es un modelo estadístico en el que se asume que el sistema a modelar es un proceso de Markov de parámetros desconocidos.

Teoría de la coalescencia: La teoría de la coalescencia es una teoría dentro de la genética de poblaciones que propone que, en una población dotada de una cierta variabilidad genética, y para genes neutros que no afecten a la viabilidad de un individuo, las diferentes secuencias génicas presentes en ella deberían haber tenido un solo ancestro común, es decir, provenir de una única secuencia (gen) original (*ancestro común más reciente* o MRCA por sus siglas en inglés).

Flujo de genes: También conocido como migración, es la transferencia de las formas alternativas que puede tener un gen de una población a otra.

Población panmítica (panmixia): Población basada en cruzamientos aleatorios.

Bioinformática: Es la aplicación de tecnologías computacionales a la gestión y análisis de datos biológicos. Los términos bioinformática, biología computacional y, en ocasiones, biocomputación, son utilizados en muchas situaciones como sinónimos, y hacen referencia a campos de estudios interdisciplinarios muy vinculados que requieren el uso o el desarrollo de diferentes técnicas estudiadas en la Ingeniería Informática como ciencia aplicada.