Universidad de las Ciencias Informáticas

“Facultad 6”

Título: “Algoritmos para el Cálculo de Descriptores Atómicos y Moleculares”

Trabajo de diploma para optar por el título de

Ingeniero Informático

Autores: Orlando Arencibia Benítez
José Gilberto Rodríguez Hernández

Tutor: Ing. Alexis René Rodríguez León

Co-Tutores: Dr. Ramón Carrasco Velar
MSc. Aurelio Antelo Collado

Junio, 2008

“Año 50 de la Revolución”

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los 28 días del mes de Junio del año 2008.

Orlando Arencibia Benítez

_____

Firma del autor

José Gilberto Rodríguez Hernández

_____

Firma del autor

Ing. Alexis René Rodríguez León

_____

Firma del Tutor

Dr. Ramón Carrasco Velar

_____

Firma del Co-Tutor

MSc. Aurelio Antelo Collado

_____

Firma del Co-Tutor

"Si avanzo, seguidme; si me detengo, empujadme; si retrocedo, matadme"

DATOS DE CONTACTOS

Tutor:

Ing. Alexis René Rodríguez León
Universidad de las Ciencias Informáticas, Cuidad de La Habana, Cuba.
Telef : (07) 835-8815
Email: arodriguezl@uci.cu

Co-Tutores:

Dr. Ramón Carrasco Velar.
Centro de Química Farmacéutica, Cuidad de La Habana, Cuba.
Email: rcarrasca@cqf.sld.cu

MSc. Aurelio Antelo Collado.
Universidad de las Ciencias Informáticas, Cuidad de La Habana, Cuba.
Telef : (07) 835-8815
Email: aantelo@uci.cu

AGRADECIMIENTOS

A Alexis René, nuestro tutor estelar, por el tiempo dedicado, sus consejos y por brindarnos su ayuda siempre que lo necesitamos.

A Carrasco por su explicación acerca de tan complicadas cuestiones químicas, que sin su ayuda no hubiéramos podido entender.

A Aurelio por las dudas aclaradas.

A nuestros compañeros de proyecto, en especial a Dairon Domínguez y Adrian Quintero, por su responsabilidad y ayuda infinita.

A todos los profes que hicieron posible nuestra formación profesional, especialmente a Yoisell Rodríguez por sus enseñanzas.

A nuestros amigos por la ayuda incondicional y estar presentes en todos los momentos.

A nuestros compañeros de grupo por creer siempre en nosotros.

A Fidel por hacer realidad nuestros sueños de formarnos como profesionales en una escuela de excelencia.

A todos aquellos que de una forma u otra se preocuparon por el desarrollo de este trabajo.

A todos muchas gracias.

Orlando

A mi familia por ser mi razón de ser.

A mi madre por estar siempre, en las buenas y en las malas, por impulsarme a ser quien soy. A ti mami gracias.

A mi papá por los buenos consejos, por inculcarme tantas cosas.

A Lauri, por el cariño inigualable y por creer siempre en mi.

A Luisa, por su infinito amor y comprensión, por estar cuando la necesite, por darme el placer de ser padre.

A mi hermano Yanosky, por su confianza, su ayuda y por siempre estar ahí.

A papi Lucio, por ser mi padre, mi abuelo y mi amigo, por los consejos y por tu infinita confianza.

A mi tía Iris, por dejarme ser su hijo, por darme todos los gustos y por ser única

A Rodovaldo, por el apoyo y la ayuda incondicional.

A mis tíos, por la preocupación.
A mis primos Frank, Jorgito, Karel, Laurent, Eli y el López, por el apoyo.
A Orly, por sus consejos y su ayuda.
A mis amigos, el Yoh@, Osmar, el Liso, Isko, Ario, el Kino, por ser incomparables, son lo mejor.
A Yurima, Maritsa y Dania por ayudarme cuando las necesite.
A todos, de corazón, ***Gracias!!!***

José

A mis padres, por educarme de una forma tan maravillosa, de la que estaré eternamente agradecido, por confiar en mí y por apoyarme siempre que los he necesitado. Los quiero mucho.
A mis hermanos Alejando y Yordi, por estar siempre presentes en mi y por saber que me quieren.
A toda mi familia, por su preocupación en cada momento y por poder contar siempre con ellos, especialmente a mis tíos Pedro y Rafael y mi primo Pepe, por brindarme en estos cinco año siempre todo lo mejor.
A mi novia Dunia, por brindarme todo su cariño y ayudarme tanto con la tesis al punto de hacerla casi suya.
A mis amigos de Palma, de los cuales siempre recibí todo el apoyo que necesité.

DEDICATORIA

“A nuestros padres”

RESUMEN

Para obtener los principios activos responsables de los efectos curativos de un medicamento es importante el uso de descriptores o índices que permitan establecer relaciones entre la estructura químico-físicas de los compuestos y su actividad biológica. En el presente trabajo, que forma parte del proyecto de investigación conjunta entre el Centro de Química Farmacéutica y la Facultad de Bioinformática de la Universidad de las Ciencias Informáticas, denominado: "Plataforma inteligente para la predicción de actividad biológica de compuestos orgánicos", se implementaron los algoritmos de varios descriptores atómicos y moleculares, los cuales se basan en la teoría de grafo químico. Los algoritmos implementados se ejecutaron de forma distribuida, permitiendo obtener los resultados con un menor costo de tiempo. Dichos algoritmos fueron probados de manera satisfactoria.

ÍNDICE

INTRODUCCIÓN

Desde los orígenes de la humanidad el hombre siempre ha sufrido diversas enfermedades y con el propósito de combatirlas y mejorar su calidad de vida, acudió a la naturaleza en busca de sustancias que le ayudaran a calmar sus dolencias. Hoy se hace indispensable la búsqueda de nuevos tratamientos o fármacos para las nuevas y diversas enfermedades a las que el mundo se enfrenta. La obtención, diseño y desarrollo de los mismos, se logra y se consolida gracias al conocimiento conjunto en las áreas de la Química, la Biología, la Medicina, la Ingeniería, la Biotecnología y la Bioinformática, entre otras, las que están integradas por el trabajo de diferentes grupos multidisciplinarios, para llevar el principio activo desde su origen hasta la presentación farmacéutica que se conoce como medicamento.

Los fármacos en un inicio se obtuvieron a partir de fuentes naturales de donde se extrajeron los principios activos responsables de sus efectos curativos atribuidos empíricamente. Posteriormente, no sólo se logró identificar a los compuestos responsables de la actividad farmacológica, sino que además se obtuvieron derivados químicos sintetizados que de cierto modo imitaran la acción del producto natural. [1]

En la actualidad, se utilizan herramientas que mejoran los procedimientos de extracción y síntesis de fármacos. La química computacional desarrolla la simulación de los requerimientos químico-físicos de estos para lograr especificidad sobre receptores o enzimas que correspondan a los sitios de acción, y la química combinatoria a su vez, permite generar mayor número de derivados en corto tiempo, a diferencia de las síntesis realizadas en laboratorios convencionales.[2]

En todo este largo proceso de prueba y error a lo largo del ciclo de desarrollo de un medicamento, tanto en la fase de descubrimiento, como en la fase de desarrollo se incurre en gastos significativos que llegan a ser millonarios por cada nuevo medicamento que sale al mercado. Sólo una de cada 10 000 moléculas ensayadas pasa a la fase de desarrollo, una de cada 100 000 supera los ensayos clínicos y logra registrarse, y sólo 3 de cada 10 nuevos medicamentos registrados recupera su inversión inicial. La pérdida de una molécula en las etapas finales supone una gran frustración y pérdida de recursos. [2]

En los últimos años, la industria farmacéutica ha reorientado sus investigaciones hacia aquellos métodos que permitan la selección racional o el diseño de nuevos compuestos. La efectividad de estos métodos depende en gran medida de los descriptores atómicos y moleculares seleccionados para caracterizar la estructura química, con el fin de predecir el comportamiento de estas. Se utilizan diversos tipos de descriptores en el diseño de fármacos: químico-cuánticos, topológicos, topográficos,

químico-físicos, híbridos, entre otros.

El Centro de Química Farmacéutica (CQF) y la Facultad de Bioinformática de la Universidad de las Ciencias Informáticas (UCI) están desarrollando un proyecto de investigación conjunta para el tamizaje virtual de compuestos orgánicos basado en técnicas de teoría de grafo e inteligencia artificial denominado: “Plataforma inteligente para la predicción de actividad biológica de compuestos orgánicos”.

Para poder establecer relaciones entre las estructuras químicas de los compuestos y su actividad biológica, es necesario contar con dos tipos de información, la estructura química descrita de alguna manera y los valores de su actividad biológica. Para describir la estructura química y asociarla a su actividad es necesario calcular descriptores de los compuestos existentes en la base de datos del proyecto, los cuales ascienden a 183 000. Debido a ese gran volumen de datos estos cálculos demorarían significativamente. En la actualidad, el cálculo de descriptores se puede lograr con diferentes programas, [Ver epígrafe 1.4] pero ninguno de estos se ajusta a los requerimientos del proyecto.

Ante tal situación se plantea el siguiente **problema científico:** ¿Cómo realizar el cálculo de descriptores atómicos y moleculares con un menor costo de tiempo?

El problema planteado se enmarca en el **objeto de estudio:** Computación de alto rendimiento aplicado a la Química Computacional.

El objeto de estudio delimita el **campo de acción:** Computación de alto rendimiento aplicado al cálculo de descriptores.

En aras de solucionar el problema planteado se define como **objetivo general:** Probar que los algoritmos implementados para calcular Descriptores Atómicos y Moleculares de forma distribuida reducen el tiempo de cálculo.

Para cumplir este objetivo se fijan los siguientes **objetivos específicos:**

- ✓ Implementar los algoritmos para realizar el cálculo de Descriptores Atómicos y Moleculares de forma distribuida.
- ✓ Validar los algoritmos implementados.
- ✓ Verificar que los algoritmos implementados reducen el tiempo de ejecución de los cálculos.

Para dar cumplimiento a los objetivos trazados se desarrollarán las siguientes **tareas:**

- ✓ Actualización del estado del arte sobre técnicas de programación de alto rendimiento.
- ✓ Implementación de una funcionalidad capaz de asignar la hibridación molecular a los átomos de la molécula.
- ✓ Implementación de una funcionalidad capaz de realizar la fragmentación molecular.
- ✓ Implementación de una funcionalidad capaz de realizar el cálculo de descriptores átomicos y moleculares de forma distribuida.
- ✓ Validación del correcto funcionamiento de los algoritmos implementados.
- ✓ Comprobación de la eficacia de los cálculos realizados.

El documento consta de un Resumen, Introducción, tres capítulos, Conclusiones Generales, Recomendaciones, Referencias Bibliográficas, Bibliografía y Anexos.

En el **Capítulo 1 Reseña Bibliográfica:** Se hace un estudio de todo lo relacionado al cálculo de descriptores, su definición, aplicación y programas existentes a nivel mundial que realizan estos cálculos.

En el **Capítulo 2 Materiales y Métodos:** Se exponen las principales tendencias, técnicas y tecnologías usadas para el desarrollo de la solución propuesta.

En el **Capítulo 3 Resultados y Discusión:** Se explica cómo se desarrollaron las funcionalidades de la solución propuesta, el pseudocódigo de dos de los principales algoritmos utilizados y los resultados obtenidos

CAPÍTULO 1

RESEÑA BIBLIOGRÁFICA

1.1 Introducción a la modelación molecular y al diseño de fármacos

En el campo de la modelación molecular y el diseño de fármacos es necesario describir la estructura química de manera cuantitativa para poder establecer los modelos que la correlacionen con los valores experimentales de la actividad biológica, estos valores cuantitativos son los llamados descriptores. El cálculo de estos constituye una forma de describir el comportamiento y la estructura de compuestos orgánicos con un alto nivel de precisión.

Las relaciones cuantitativas entre parámetros químico-fisicos y la actividad biológica las plantearon en los años 60 Hansch y Fujita mediante técnicas estadísticas, comenzando así la era de los estudios QSAR (*Quantitative Structure Activity Relationship*) como un caso particular y más complejo de las relaciones estructura-propiedad.**[3]** Los métodos QSAR y QSPR (*Quantitative Structure Property Relationship*) han demostrado que las relaciones entre la estructura molecular y las propiedades químico-físicas o actividades biológicas de los compuestos se pueden cuantificar matemáticamente a partir de parámetros estructurales simples.[4] Más adelante, en la década de los 70, los métodos de cristalización de proteínas y resolución de su estructura mediante difracción de rayos X, así como los de resonancia magnética nuclear, empezaron a aportar suficientes estructuras de macromoléculas como para pensar en un diseño de ligandos a partir del conocimiento tridimensional de su diana farmacológica.

Durante las décadas de los 70 y 80 se produjo un gran desarrollo teórico de metodologías de modelación molecular y la aparición de los ordenadores personales permitió popularizar su aplicación.

Ya en los 90 se desarrolló explosivamente la informática, que permitió el procesamiento de enormes volúmenes de datos y la aplicación de la química cuántica, que si bien era conocida desde hace años, su aplicación práctica era prácticamente inabordable con los medios de cómputo existentes. [3]

La evolución de la química teórica y la computación (tanto el *hardware* como el *software*) han propiciado darle un nuevo enfoque al diseño de fármacos. Dentro de este campo se encuentran los métodos asistidos por computadoras que relacionan la estructura química con la actividad biológica, los que se dividen en dos grandes categorías:

- ✓ Los métodos SAR (*Structure Activity-Relationships*) consideran las propiedades de las moléculas en tres dimensiones y son importantes en ellos aspectos como el análisis conformacional, la mecánica-cuántica, los campos de fuerzas y los gráficos moleculares interactivos. Estos últimos permiten la representación y la manipulación de la molécula en tres dimensiones, lo que proporciona una información espacial que es esencial para comparar moléculas y para estudiar la interacción entre ligandos y receptores macromoleculares.

- ✓ Los métodos QSAR (*Quantitative Structure Activity-Relationships*) están basados en el empleo de técnicas de correlación entre la estructura química y la actividad biológica. En este tipo de técnica, la actividad biológica se interpreta como función de diferentes aspectos de la estructura química. [2]

1.2 Métodos QSAR

Las técnicas QSAR están basadas en el tratamiento estadístico de principios básicos de la Química, que relacionan las propiedades químicas asociadas a la estructura de la molécula con el comportamiento biológico. Asumen que existe una correlación implícita entre las propiedades y la estructura molecular y tratan de establecer relaciones matemáticas simples para describir y luego extrapolar una o varias de esas propiedades. Dichas relaciones pueden determinarse a través de métodos matemáticos, ya sean, regresiones multilineales o métodos no lineales. [4]
Se ha demostrado que la actividad biológica de un conjunto de compuestos que actúan con el mismo mecanismo de acción puede modelarse matemáticamente a partir de parámetros estructurales simples. De ahí que en la última década, los modelos QSAR encontraron un auge como una alternativa viable para predecir actividades biológicas en compuestos orgánicos.

Requerimientos de los modelos QSAR:

- ✓ Disponer de descriptores moleculares capaces de caracterizar satisfactoriamente diferentes conjuntos de compuestos químicos.

- ✓ Desarrollar modelos matemáticos capaces de generar un modelo predictivo.

En la actualidad existen diferentes técnicas por las que se puede desarrollar un estudio QSAR y se pueden clasificar en:

- ✓ **QSAR tradicional**

  El método tradicional incluye el tratamiento estadístico de los datos por métodos multivariados, que incluyen el análisis de regresión, análisis clúster y análisis por componentes principales, entre otras técnicas estadísticas. En ellos se valora la actividad biológica como variable dependiente del conjunto de descriptores moleculares que constituyen las variables independientes. Como resultado de esto se obtienen modelos que describen la actividad biológica como una determinada función matemática de los descriptores moleculares, bien sean estos estructurales, químico-físicos, o ambos**.** [2]

- ✓ **QSAR por redes de neuronas**

  Los avances en neurofisiología, y las nuevas técnicas experimentales como la electroencefalografía, la tomografía axial computarizada, el tratamiento de imágenes por RMN, la tomografía de positrones o la de fotones, han incrementado la comprensión de la anatomía del cerebro y de los procesos químico-físicos que ocurren en él. A partir de este conocimiento, se diseñaron modelos matemáticos de adquisición de conocimientos del cerebro humano llamados Redes de neuronas artificiales (RNA), o simplemente, Redes de Neuronas (RN). Los modelos de redes neuronales se basan en la interacción colectiva y en paralelo de una gran cantidad de procesadores simples que simulan la conducta de las neuronas biológicas. [2]

- ✓ **QSAR tridimensional (3D-QSAR)**

  El QSAR tridimensional es una técnica moderna que combina los aspectos relacionados en el estudio QSAR tradicional con los de los estudios SAR o de modelado molecular. En los estudios SAR de análogos, la elección se limita a si el compuesto es activo o inactivo. En la práctica, sin embargo, los resultados se representan generalmente de forma numérica que refleja las diferencias cuantitativas en la unión de uno u otro ligando.

  El primer logro en este sentido lo alcanzaron Cramer y colaboradores en 1988 al desarrollar el análisis comparativo de campos moleculares (CoMFA). La premisa básica de este método, es que “el muestreo adecuado de los campos eléctricos y electrostáticos alrededor de un conjunto de moléculas o fármacos, puede proporcionar toda la información necesaria para comprender las actividades biológicas”. [2]

1.3 ¿Qué es un descriptor?

Se conocen como descriptores, los cuantificadores matemáticos que relacionan la estructura molecular de los compuestos a partir de parámetros estructurales simples, lo que posibilita interpretar las propiedades moleculares y describir el comportamiento de compuestos orgánicos, la calidad de los mismos condiciona la calidad de los modelos matemáticos que describan los fenómenos biológicos. [5]
En el campo de los descriptores se conjugan diferentes disciplinas como el álgebra, la teoría gráfica o de grafos, la teoría de la información, la química computacional, las teorías de la reactividad química y la química-física. Además, la programación, el sofisticado software y hardware son importantes herramientas. [6]
Dentro de un amplio grupo de descriptores se encuentran los constitucionales, geométricos, electrostáticos, químico-cuánticos, termodinámicos, híbridos, topográficos y topológicos, destacándose estos últimos, pues se basan en la estructura en dos dimensiones o topología de las moléculas, los cuales se derivan generalmente de las matrices de conectividad entre vértices del grafo molecular desprovisto de hidrógenos. Debido a esto, los índices topológicos constituyen cierta pérdida de información, ya que se representa un objeto tridimensional por un número simple, no obstante, estos índices contienen una sorprendente información relacionada con la forma molecular, el grado de ramificación, tamaño molecular y la flexibilidad estructural que los hace de gran utilidad en los fines prácticos de sus aplicaciones. [7] Entre los más conocidos se destacan los índices de conectividad molecular, propuestos por Randić y desarrollados en profundidad por Kier y Hall. Se distinguen también otros índices como los topográficos que se basan en la estructura en tres dimensiones, y se derivan, entre otras, de las matrices topográficas entre vértices ponderados por los órdenes de enlace. Los índices híbridos implementados en esta tesis se derivan de la estructura molecular y una propiedad químico-física particionada sobre los átomos. [8]
En el presente trabajo se implementan los algoritmos de varios descriptores moleculares, los cuales describen aspectos estructurales de la molécula con un valor único, de manera similar a propiedades químico-físicas como el peso molecular, los puntos de fusión y ebullición. Los descriptores atómicos describen aspectos particulares de un átomo dentro de una molécula dada. Estos dos tipos de descriptores pueden clasificarse en topológicos, híbridos y topográficos, y así se considerará a lo largo de todo el documento.

1.4 Programas vinculados con el cálculo de descriptores

Actualmente en el mundo se trabaja en el cálculo de múltiples índices teóricos, para los cuales se han desarrollado diferentes programas entre los se encuentran: Codessa, Dragon y Molconn-Z.

1.4.1 Codessa (*Comprehensive Descriptors for Structural and Statistical Analysis*) [9]

Este programa brinda al usuario la posibilidad de calcular más de 500 descriptores (topológicos, geométricos, constitucionales, termodinámicos, electrostáticos), un módulo para el establecimiento de modelos QSAR basado fundamentalmente en métodos estadísticos (análisis de regresión, heurísticos y análisis multivariados) y la integración completa con Gaussian 98/03. Es multiplataforma, pero es un programa propietario de elevado costo.

Tipo de licencia	Costo 1er Año	Soporte
Usuario Simple	$4200	$800
Múltiples Usuarios	$12600	$7600
Servidor/Clúster	$25000	$15000
Organización	$31500	$19000

Tabla 1.1 Precio Comercial del Codessa

1.4.2 Dragon

Es una aplicación para el cálculo de descriptores moleculares originalmente desarrollado por *Milano Chemometrics y QSAR Research Group*. Calcula un total de 1 664 descriptores de ellos 119 son descriptores topológicos, 33 índices de conectividad, 74 descriptores geométricos, entre otros.
Dragon está diseñado para funcionar en Windows y Linux. Hay dos versiones de Windows: Dragon Professional y el modo Dragon Plus. Para Linux existe una versión, llamada DragonX, que sólo funciona en segundo plano por una línea de comandos. Su diseño sólo permite realizar el cálculo de descriptores moleculares y no tiene la funcionalidad de realizar análisis QSAR.

Tipo de licencia	Costo 1er Año	Costo 2do Año
Una Estación de Trabajo	€2700	€1000
Múltiples Estaciones de Trabajo	€4000	€2000

Tabla 1.2 Precio Comercial del Dragon Plus

1.4.3 Molconn-Z [10]

El programa Molconn-Z está diseñado para realizar el cálculo de una amplia gama de índices topológicos de una estructura molecular entre los que se destacan: índices de conectividad, Índice del Estado Electrotopológico, índices de equivalencia topológica e índice topológico total y alguna información de los fragmentos del grafo, de los átomos, del tipo de enlace, etc.
Molconn-Z permite el trabajo con bases de datos comerciales y exporta a un fichero de salida estándar los índices calculados para poder ser utilizados en análisis estadísticos.

Tipo de licencia	Costo
Usuario Simple	$8500
Múltiples Usuarios	$850
Organización	$17000

Tabla 1.3 Precio Comercial del Molconn-Z 4.0 [11]

Como se señaló anteriormente, estos programas que realizan el cálculo de los descriptores presentan un elevado costo, lo cual limita las posibilidades de adquirirlos, además de no calcular descriptores topográficos, por lo que no se ajustan a las necesidades del proyecto.

1.5 Conclusiones

En el capítulo se expuso de forma general los métodos de modelación molecular, haciendo énfasis en los métodos que relacionan la estructura química con la actividad biológica, QSAR. Se hizo un análisis del uso de los descriptores atómicos y moleculares para describir la relación mencionada. Se investigó acerca de la existencia de algunos programas que realizan el cálculo de los descriptores.

CAPÍTULO 2
MATERIALES Y MÉTODOS

En este capítulo se describen las herramientas utilizadas para la implementación de los algoritmos de los descriptores y se analizan los métodos empleados para dar solución al problema planteado.

2.1 Materiales

2.1.1 Lenguaje de Modelado (UML) [12]

El estándar internacional para el modelado, UML es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software desde una perspectiva orientada a objetos. Presenta las siguientes características:

- ✓ Modela sistemas utilizando técnicas orientadas a objetos.
- ✓ Permite especificar todas las decisiones de análisis, diseño e implementación, construyéndose así modelos precisos, no ambiguos y completos.
- ✓ Permite documentar todos los artefactos de un proceso de desarrollo (requisitos, arquitectura, pruebas, versiones, etc.).
- ✓ Proporciona un vocabulario y unas reglas que se centran en la representación conceptual y física de un sistema y que indican cómo crear y leer modelos bien formados.
- ✓ Mezcla gráficos y texto, de hecho, detrás de cada símbolo en la notación UML hay una semántica bien definida, de manera que un desarrollador puede escribir un modelo en UML, y otro desarrollador, o incluso otra herramienta, puede interpretar ese modelo sin ambigüedad.
- ✓ Sus modelos pueden conectarse de forma directa con una gran variedad de lenguajes de programación, permitiendo ingeniería inversa y directa.
- ✓ Cubre toda la documentación de la arquitectura de un sistema y todos sus detalles. También

proporciona un lenguaje para expresar requisitos y pruebas del software.

- ✓ Modela las actividades de planificación de proyectos y gestión de versiones.
- ✓ Existe un equilibrio entre expresividad y simplicidad, pues no es difícil de aprender ni de utilizar.
- ✓ UML es independiente del proceso, aunque para utilizarlo óptimamente se debería usar en un proceso que fuese dirigido por los casos de uso, centrado en la arquitectura, iterativo e incremental.

2.1.2 Herramienta CASE para la modelación del sistema: Visual Paradigm [5]

Las herramientas CASE (*Computer Aided Software Engineering* – Ingeniería de Software Asistida por Computadora) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas herramientas ayudan en todos los aspectos del ciclo de vida de desarrollo del software, en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores, entre otras.

Visual Paradigm es una potente herramienta para visualizar y diseñar elementos de software, posee características graficas muy cómodas que facilitan la realización de los diagramas de modelación que sigue el estándar UML, proporciona a los desarrolladores una plataforma que les permita diseñar un producto con calidad de forma rápida, facilita la interoperabilidad con otras herramientas; tiene licencia gratuita, es multiplataforma y posibilita la integración con las siguientes herramientas Java: Eclipse/IBM WebSphere, JBuilder, NetBeans IDE, Oracle JDeveloper, BEA Weblogic. Está disponible en varias ediciones: Enterprise, Professional, Community, Standard, Modeler y Personal, además posibilita hacer ingeniería inversa para Java, .Net, XMLY Hibernate, permite la exportación de imágenes jpg, png y svg.

La herramienta está diseñada para una amplia gama de usuarios, incluyendo Ingenieros de Software, Analistas de Sistemas, Analistas de Negocios y Arquitectos de Sistemas que estén interesados en la creación de grandes sistemas de software de manera confiable a través del paradigma “Orientado a Objetos”.

2.1.3 Lenguaje de programación: Java [13]

Un lenguaje de programación permite crear la interfaz gráfica para el usuario y la funcionalidad entre ellas, para llevar a cabo las actividades de gestión del sistema.

Para desarrollar este trabajo se utilizó Java, el cual es un lenguaje de programación muy extendido en la actualidad y que cada día cobra más auge dentro del mundo de la programación por sus potencialidades y facilidad de aprendizaje. Dentro de algunas de sus características más relevantes se tienen que Java es:

- ✓ **Simple**

  Ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de éstos. C++ es un lenguaje que adolece de falta de seguridad, pero C y C++ son lenguajes más difundidos, por ello Java se diseñó para ser parecido a C++ y así facilitar un rápido y fácil aprendizaje. Reduce en un 50% los errores más comunes de programación en lenguajes como C y C++, al eliminar muchas de las características de éstos.

- ✓ **Orientado a Objetos**

  Implementa la tecnología básica de C++ con algunas mejoras y elimina algunas cosas para mantener el objetivo de la simplicidad del lenguaje. Trabaja con sus datos como objetos y con interfaces a esos objetos. Soporta las tres características propias del paradigma de la orientación a objetos: encapsulación, herencia y polimorfismo.

- ✓ **Robusto**

  Realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución. La comprobación de tipos en Java ayuda a detectar errores en el ciclo de desarrollo. Obliga a la declaración explícita de métodos, reduciendo así las posibilidades de error. Maneja la memoria para eliminar las preocupaciones por parte del programador de la liberación o corrupción de esta.

- ✓ **Seguro**

  El código Java pasa muchas pruebas antes de ejecutarse en una máquina. Este se pasa a través de un verificador de *byte-codes* que comprueba el formato de los fragmentos de código y aplica un probador de teoremas para detectar fragmentos de código ilegal. Si los *byte-codes* pasan la verificación sin generar ningún mensaje de error, entonces sabemos que:

  - El código no produce desbordamiento de operandos en la pila.

  - El tipo de los parámetros de todos los códigos de operación son conocidos y correctos.

- No ha ocurrido ninguna conversión ilegal de datos, tal como convertir enteros en punteros.
- El acceso a los campos de un objeto se sabe que es legal.
- No hay ningún intento de violar las reglas de acceso y seguridad establecidas.

✓ **Arquitectura neutral**

El compilador de Java compila su código a un fichero objeto de formato independiente de la arquitectura de la máquina en que se ejecutará. Cualquier máquina que tenga el sistema de ejecución (*run-time)* puede ejecutar ese código objeto, sin importar en modo alguno la máquina en que ha sido generado. Actualmente existen sistemas *run-time* para Solaris 2.x, SunOs 4.1.x, Windows, Linux, Irix, Aix, Mac, Apple.

✓ **Distribuido**

Java se ha construido con extensas capacidades de interconexión *TCP/IP*. Existen librerías de rutinas para acceder e interactuar con protocolos como *HTTP* y *FTP*. Esto permite a los programadores acceder a la información a través de la red con tanta facilidad como a los ficheros locales. La verdad es que Java en sí no es distribuido, sino que proporciona las librerías y herramientas para que los programas puedan serlos.

✓ **Multihilo (Multithreaded)**

Permite muchas actividades simultáneas en un programa. Los hilos de ejecución (*threads*, a veces llamados procesos ligeros), son básicamente pequeños procesos o piezas independientes de un gran proceso. El beneficio de ser multihilo consiste en un mejor rendimiento interactivo y mejor comportamiento en tiempo real.

2.1.4 Entorno de desarrollo: Eclipse 3.2 [14]

Dentro de los entornos de desarrollo integrados (IDE, *Integrated Development Environment*) se encuentra Eclipse, que, combinado con el JDT (*Java Development Tooling*), permite disponer de un IDE para Java de excelente calidad. Eclipse posee un editor muy visual con sintaxis coloreada, ofrece compilación incremental de código, un potente depurador (que permite establecer puntos de interrupción, modificar e inspeccionar valores de variables, etc. e incluso depurar código que resida en una máquina remota), un navegador de clases, un gestor de archivos y proyectos, pero no se limita

sólo a esto. La versión estándar de Eclipse proporciona también una biblioteca de refactorización de código y una lista de tareas, soporta la integración con JUnit, y suministra un *front-end* gráfico para Ant, la conocida herramienta de código abierto que forma parte del proyecto Jakarta de Apache.
También incluye una herramienta para completar código, otra característica de Eclipse es que es muy eficiente para reducir los tiempos de depuración y pruebas.

2.1.5 Sistemas Gestores de Bases de Datos (SGBD): MySQL [15]

Los sistemas de bases de datos surgieron con el objetivo de resolver los problemas que planteaban los sistemas de ficheros. El SGBD o DBMS (*Data-Base Management System*) es un conjunto de programas que permiten a los usuarios definir, crear y mantener una base de datos, además de proporcionar un acceso controlado a la misma.
Todos los accesos a la base de datos se realizan a través del SGBD, el que proporciona un lenguaje de definición de datos que permite a los usuarios definir la base de datos como tal, y un lenguaje de manejo de datos que permite a los usuarios la inserción, actualización, eliminación y consulta de datos. Proporciona además acceso controlado, seguridad, integridad, concurrencia y controla la recuperación ante fallos, además de proporcionar un mecanismo de vistas que permite mostrar sólo aquellos datos que interesan.
MySQL es un SGBD relacional, multi-hilo y multi-usuario con más de seis millones de instalaciones, licenciado bajo licencia GPL de la GNU. Existen varias APIs que permiten a las aplicaciones escritas en diversos lenguajes de programación, acceder a las bases de datos MySQL, incluyendo C, C++, C#, Pascal, Delphi, Eifel, Smalltalk, Java (con una implementación nativa del *driver* de Java), Lisp, Perl, PHP, Python, Ruby, REALbasic (MAC), FreeBASIC, y Tcl; cada uno de estos utiliza una API específica. También existe un interfaz ODBC, llamado MyODBC que permite a cualquier lenguaje de programación que la soporte, comunicarse con las bases de datos MySQL. Este gestor de bases de datos es, probablemente, el gestor más usado en el mundo del software libre, debido a su gran rapidez y facilidad de uso. [16]

Las principales características de este gestor de bases de datos son las siguientes:

- ✓ Soporte multi-plataforma.
- ✓ Procedimientos almacenados.
- ✓ Vistas actualizables.

- ✓ Motores de almacenamiento independientes (MyISAM para lecturas rápidas, InnoDB para transacciones e integridad referencial).
- ✓ Soporte para SSL.
- ✓ Soporte completo para Unicode.
- ✓ Uso de multihilos, mediante hilos del kernel.
- ✓ Completo soporte funciones de agrupación.
- ✓ Ofrece un sistema de contraseñas y privilegios seguro mediante verificación basada en el host.
- ✓ Múltiples motores de almacenamiento (MyISAM, Merge, InnoDB, BDB, Memory/heap, MySQL Cluster, Federated, Archive, CSV, Blackhole y Example en 5.x), permitiendo al usuario escoger la que sea más adecuada para cada tabla de la base de datos.
- ✓ Agrupación de transacciones, reuniendo múltiples transacciones de varias conexiones para incrementar el número de transacciones por segundo.
- ✓ Soporta gran cantidad de datos. MySQL Server tiene bases de datos de hasta 50 millones de registros.
- ✓ Permite hasta 64 índices por tabla.

2.1.6 Plataforma de cómputo distribuido [17]

Un sistema distribuido es un conjunto de computadoras autónomas que aparecen integradas ante los usuarios como una única máquina para resolver determinado problema.
Los componentes del sistema no son más que hardware y software que se comunican entre sí a través de una red, preferiblemente canales de alta velocidad. En el caso de un sistema de cómputo distribuido el problema a resolver, por lo general, requiere grandes cálculos y lo que se trata es de reducir el tiempo en obtener la respuesta haciendo uso del procesamiento en paralelo al distribuir los cómputos de forma transparente para el usuario entre varias computadoras.

2.1.7 Recursos computacionales utilizados

Cuando se utiliza un conjunto de computadoras para realizar grandes cálculos y que todas funcionen como una, se necesita saber el tamaño del problema que se desea resolver y los recursos de hardware disponibles.

En la ***Tabla 2.1*** se muestra de manera general los recursos computacionales de hardware y software utilizados. Para realizar este trabajo se emplearon siete ordenadores con las mismas especificaciones. Es necesario destacar que estos ordenadores están dedicados al trabajo diario del personal que se desempeña en el proyecto, debido a esto su capacidad de cómputo varía en dependencia de los procesos que se ejecutan en un momento determinado, por lo que afecta el rendimiento de los cálculos desarrollados.

Procesador (CPU)	Memoria (RAM)	Red Ethernet	Sistema Operativo	Cantidad
Intel (R) Pentium (R) 4 Velocidad 3.0 GHz	512 MB	Velocidad 100Mbs	Ubuntu 7.10	7

Tabla 2.1 Especificaciones del hardware de las estaciones de trabajo utilizadas

2.2 Métodos

2.2.1 Teoría de Grafos

La teoría de grafos es utilizada para estudiar los problemas que surgen en una amplia variedad de áreas de aplicación incluyendo la informática, la ingeniería eléctrica, la química, la investigación operativa, las ciencias políticas y la economía, sólo por nombrar unas pocas. Con el fin de usar la teoría de grafos es necesario representar la estructura física del problema como un grafo.

Un grafo no dirigido ***G = (V A),*** se compone de dos conjuntos finitos: el conjunto de nodos ***V*** = *(v1, v2,…)*, que contiene el conjunto de nodos o vértices de ***G***; y el conjunto de aristas ***A*** = *(a1,a2,…)*, que es el conjunto de pares no ordenados de nodos diferentes de ***G***. Cada elemento del conjunto de aristas *a = (u,v)* se conoce como una arista y se dice que une los nodos *u* y *v*. Si *A= (u,v)* es una arista de ***G*** entonces los nodos *u* y *v* son adyacentes. Además se dice que *a* es incidente en los nodos *u* y *v*. Dado que estamos tratando con pares no ordenados, *(u,v)* y *(v,u)* representan la misma arista.

Caminos

Sean *x*, *y* vértices del grafo, se dice que hay un camino en *G* de *x* a *y* si existe una sucesión finita no vacía de aristas $\{x,v_1\}$, $\{v_1,v_2\}$,..., $\{v_n,y\}$. En este caso *x* e *y* son los extremos del camino, el número de aristas del camino se llama longitud del camino u orden. Si los vértices no se repiten, se dice que el camino es propio o simple.

Clúster

Es una cadena finita donde el nodo inicial es el centro y los demás nodos (al menos 3) están conectados por aristas al nodo centro.

Clúster/Camino

Es la combinación entre los clústeres y los caminos del grafo.

Representación de grafos computacionalmente.

Las dos maneras más comunes de representar un grafo como una estructura de datos en un programa es mediante:

- ✓ **Matrices de adyacencia:** La forma más fácil de guardar la información de los nodos es mediante la utilización de un vector que los indexe, de manera que los arcos entre los nodos se pueden ver cómo relaciones entre los índices. Esta relación entre índices se puede guardar en una matriz de adyacencia.

- ✓ **Listas de adyacencia:** En las listas de adyacencia lo que se hace es guardar por cada nodo, además de la información que pueda contener el propio nodo, una lista dinámica con los nodos a los que se puede acceder desde él. La información de los nodos se puede guardar en un vector, al igual que antes, o en otra lista dinámica.

2.2.2 Grafo Químico

La teoría de grafos es una herramienta matemática útil para los métodos QSAR/QSPR/SAR y está basada en la asociación de las ecuaciones químicas a los grafos matemáticos, a partir de la cual se pueden definir diversos descriptores, a fin de codificar la información estructural.

En la teoría del grafo químico se define este como un grafo conexo y no dirigido, en el que los nodos

son los átomos y las aristas son los enlaces entre estos. La necesidad de representar la estructura molecular por un número simple se origina a partir del hecho de que las propiedades moleculares son registradas como tales y por lo tanto, los modelos QSAR/QSPR/SAR se reducen a una correlación entre, al menos, dos conjuntos de números, vía una expresión algebraica (un conjunto de números representan las propiedades y los otros representan las estructuras moleculares que se estudian).
En la ***Figura 2.1*** se muestra el grafo químico desprovisto de hidrógenos de la Alanina.

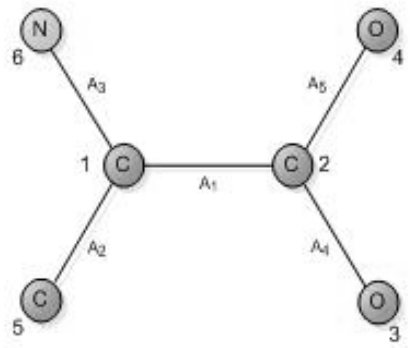


Figura 2.1 Grafo molecular de la Alanina

Ejemplo de fragmentos del grafo químico. [Ver Anexo 1]

- ✓ **Caminos**

  Caminos Orden 1

  1-2, 1-5, 1-6, 2-3, 2-4

  Caminos Orden 2

  1-2-3, 1-2-4, 2-1-5, 2-1-6

  Caminos Orden 3

  3-2-1-5, 3-2-1-6, 4-2-1-5,4-2-1-6

- ✓ **Clúster**

  Clúster Orden 3

  1-2-5-6, 2-1-3-4

- ✓ **Clúster/Camino**

  Clúster/Camino Orden 3/1

  1-2-5-6/2-3, 1-2-5-6/2-4, 2-1-3-4/1-5, 2-1-3-4/1-6

2.2.3 Estructuras de datos utilizadas

Para representar un grafo en la memoria de la computadora se utilizan comúnmente las listas y matrices de adyacencias. En este epígrafe se muestra un ejemplo de cada estructura y las variantes de las matrices utilizadas.

- ✓ Representación de la estructura de datos lista de adyacencia a partir del grafo químico representado en la ***Figura 2.1***

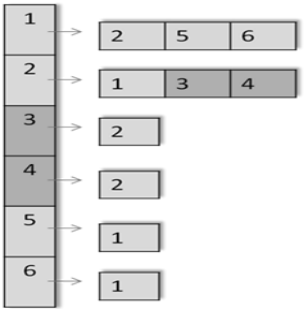


Figura 2.2 Lista de Adyacencia

- ✓ Representación de la estructura de datos matriz topológica de conectividad entre vértices (matriz de adyacencia de vértices) a partir del grafo químico representado en la ***Figura 2.1***

	1	2	3	4	5	6
1	0	1	0	0	1	1
2	1	0	1	1	0	0
3	0	1	0	0	0	0
4	0	1	0	0	0	0
5	1	0	0	0	0	0
6	1	0	0	0	0	0

Figura 2.3 Matriz topológica de conectividad entre vértices

- ✓ Representación de la estructura de datos matriz topográfica de conectividad entre vértices ponderados por los órdenes de enlace entre los vértices, a partir del grafo químico

representado en la ***Figura 2.1***. Donde d_{ij} representa la distancia desde el vértice **i** hasta el vértice **j.**

	1	2	3	4	5	6
1	0	d_{12}	0	0	d_{15}	d_{16}
2	d_{21}	0	d_{23}	d_{24}	0	0
3	0	d_{32}	0	0	0	0
4	0	d_{42}	0	0	0	0
5	d_{51}	0	0	0	0	0
6	d_{61}	0	0	0	0	0

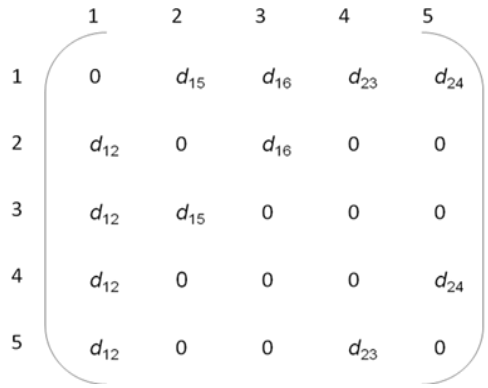
Figura 2.4 Matriz topográfica de conectividad entre vértices ponderados por los órdenes de enlace

- ✓ Representación de la estructura de datos matriz topológica de conectividad entre aristas a partir del grafo químico representado en la ***Figura 2.1***

	1	2	3	4	5
1	0	1	1	1	1
2	1	0	1	0	0
3	1	1	0	0	0
4	1	0	0	0	1
5	1	0	0	1	0

Figura 2.5 Matriz topológica de conectividad entre aristas

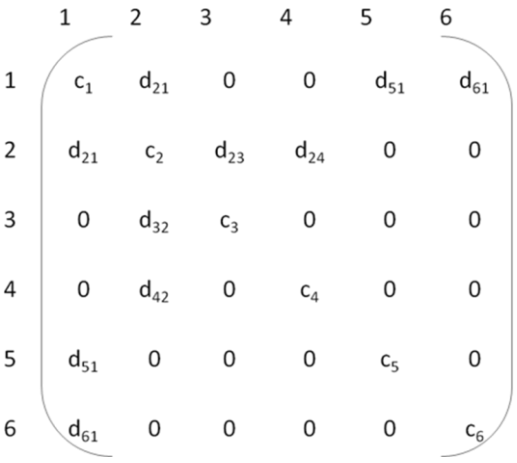
- ✓ Representación de la estructura de datos matriz topográfica de conectividad entre aristas ponderados por los órdenes de enlace a partir del grafo químico representado en la ***Figura 2.1*** Donde d_{ij} representa la distancia desde la arista **i** hasta la arista **j.**



	1	2	3	4	5
1	0	d_{15}	d_{16}	d_{23}	d_{24}
2	d_{12}	0	d_{16}	0	0
3	d_{12}	d_{15}	0	0	0
4	d_{12}	0	0	0	d_{24}
5	d_{12}	0	0	d_{23}	0

Figura 2.6 Matriz topográfica de conectividad entre aristas

- ✓ Representación de la estructura de datos matriz topográfica de conectividad entre vértices ponderados por la densidad de la carga atómica a partir del grafo químico representado en la ***Figura 2.1.*** La densidad de carga atómica asociada a cada átomo, se ubica en la diagonal principal de la matriz.



	1	2	3	4	5	6
1	c_1	d_{21}	0	0	d_{51}	d_{61}
2	d_{21}	c_2	d_{23}	d_{24}	0	0
3	0	d_{32}	c_3	0	0	0
4	0	d_{42}	0	c_4	0	0
5	d_{51}	0	0	0	c_5	0
6	d_{61}	0	0	0	0	c_6

Figura 2.7 Matriz topográfica de conectividad entre vértices ponderados por la densidad de la carga atómica

2.2.4 Pseudocódigo [18]

El pseudocódigo se utiliza para expresar algoritmos en una forma que sea independiente de cualquier lenguaje de programación. La razón para usar seudocódigo es que permite transmitir en términos generales las ideas básicas de un algoritmo. Una vez que los programadores entienden el algoritmo

que está siendo expresado por el pseudocódigo, pueden implementarlo en cualquier lenguaje de programación.

El pseudocódigo básicamente consiste en palabras reservadas y frases afines al idioma que se utilizan para indicar el flujo de control. Las palabras reservadas se escriben en letra minúscula negrita y los nombres de *TADs* en minúsculas. Para la escritura de un pseudocódigo se utilizan los siguientes convenios:

- ✓ El símbolo ► es usado para indicar que el resto de la línea debe ser usado como comentario. Si en una línea se usa más de una instrucción, se pone (;) para separarlas.
- ✓ Las instrucciones de asignación tienen la forma x ← e, que asigna el valor de *e* a la variable *x*; se pueden realizar asignaciones múltiples, ejemplo,

  x ← y ← e, asigna el valor de *e* a la variable *x* e *y*.

- ✓ Las construcciones iterativas pueden ser especificadas usando la siguiente forma:

  para instrucción de asignación **hasta** valor final
  hacer en pasos de incremento
  instrucciones del bucle

- ✓ Para evitar la proliferación de instrucciones en ciclos, construcciones condicionales y funciones, se utiliza **inicio** y **fin,** para indicar cuándo empiezan y terminan.
- ✓ Las construcciones condicionales usan las palabras reservadas **si-entonces-sino.**
- ✓ En el caso de las funciones, es necesario colocar una palabra como **regresar** o **devolver** para indicar cuál es la salida generada por el algoritmo.

2.2.5 Complejidad de algoritmos

La complejidad de un algoritmo considera tanto la dificultad de implementarlo como la eficacia del mismo. La eficiencia de un algoritmo puede estar dada por la cantidad de memoria que consume y/o la cantidad de tiempo de cómputo requerido, cuando el tamaño de la entrada del algoritmo crece. El tiempo requerido por un algoritmo se determina por el número de operaciones elementales que deben ser realizadas durante su ejecución, lo cual es también conocido por complejidad temporal del algoritmo y está dada por la función T(n), que puede ser una función polinomial, logarítmica,

exponencial o lineal con variable independiente siempre dependiendo de una variable n Real o natural, dependiendo del tipo de algoritmo analizado. [18]

Reglas generales para el cálculo de la complejidad de algoritmos [19]

La siguiente lista presenta un conjunto de reglas generales para el cálculo del número de operaciones elementales (OE), siempre considerando el peor caso. Estas reglas definen el número de OE de cada estructura básica del lenguaje, por lo que el número de OE de un algoritmo puede hacerse por inducción sobre ellas.

- ✓ Se considera que el tiempo de una OE es, por definición, de orden uno. La constante *c* toma valor uno.

- ✓ El tiempo de ejecución de una secuencia consecutiva de instrucciones se calcula sumando los tiempos de ejecución de cada una de las instrucciones.

- ✓ El tiempo de ejecución de la sentencia "***case*** C ***of*** v_1:S_1|v_2:S_2|...|v_n:S_n ***end***;" es $T = T(C) + \max\{T(S_1),T(S_2),...,T(S_n)\}$. $T(C)$ incluye el tiempo de comparación con v_1, v_2,..., v_n.

- ✓ El tiempo de ejecución de la sentencia "***if*** C ***then*** S_1 ***else*** S_2 ***end***;" es $T = T(C) + \max\{T(S_1),T(S_2)\}$.

- ✓ El tiempo de ejecución de un bucle de sentencias "***while*** C ***do*** S ***end***;" es $T = T(C)$ + (número de iteraciones)*($T(S) + T(C)$). Se puede observar que tanto $T(C)$ como $T(S)$ pueden variar en cada iteración, y por tanto hay que tenerlo en cuenta para su cálculo.

- ✓ Para calcular el tiempo de ejecución del resto de sentencias iterativas (***for***, ***repeat***, ***loop***) basta expresarlas como un bucle ***while***.

- ✓ El tiempo de ejecución de una llamada a un procedimiento o función $F(P_1, P_2,..., P_n)$ es uno (por la llamada), más el tiempo de evaluación de los parámetros P_1, P_2,..., P_n, más el tiempo que tarde en ejecutarse F, esto es, $T = 1 + T(P_1) + T(P_2) + ... + T(P_n) + T(F)$. No se contabiliza la copia de los argumentos a la pila de ejecución, salvo que se trate de estructuras complejas (registros o vectores) que se pasan por valor. En este caso se cuenta tantas OE como valores simples contenga la estructura. El paso de parámetros por referencia, no contabiliza.

- ✓ El tiempo de ejecución de las llamadas a procedimientos recursivos dan lugar a ecuaciones de recurrencia.

Para calcular el tiempo de ejecución *T* de un algoritmo, se clasifican dichas funciones de forma que se puedan comparar. Para ello, se definen clases de equivalencia, correspondientes a las funciones que crecen de la misma forma.

Definición de Cota Superior. Notación O [18]

Dada una función *f*, se quiere estudiar aquellas funciones *g* que a lo sumo crecen tan deprisa como *f*. Al conjunto de tales funciones se le llama cota superior de *f* y se denomina O(*f*). Conociendo la cota superior de un algoritmo se puede asegurar que, en ningún caso, el tiempo empleado será de un orden superior al de la cota.

2.2.6 Algoritmos de los descriptores calculados

2.2.6.1 Descriptores Atómicos

- ✓ **Índice del Estado Electrotopológico (Estate)** [8]

  Es un índice atómico topológico desarrollado por Kier y Hall. Depende de un valor intrínseco del átomo y de una función de la distancia topológica de su entorno, que lo modifica. A diferencia de otros índices topológicos, el índice del Estado Electrotopológico se computa como una invariante grafo teórica sobre cada átomo del grafo molecular, en lugar de hacerlo sobre la suma de todos los subgrafos del grafo molecular en su conjunto y utiliza el grafo desprovisto de hidrógeno. El índice se basa en el efecto de cada átomo sobre otros átomos en la molécula, modificado por la topología molecular. Cada átomo tiene asignado un valor intrínseco $\boldsymbol{I_i}$ calculado por la ecuación ***2.1*** en donde $\boldsymbol{N}$ es el número cuántico principal del átomo $\boldsymbol{i}$, $\boldsymbol{\delta^v}$ es el número de electrones de valencia en el esqueleto, y $\boldsymbol{\delta=\sigma-h}$ en donde $\boldsymbol{\sigma}$ es el número de electrones sigma y $\boldsymbol{h}$ el número de átomos de hidrógeno enlazados al átomo $\boldsymbol{i}$.

$$I_i = [(2/N)^2\,\delta^v + 1]/\delta \qquad \textbf{[2.1]}$$

El Estate ***S(Ai)*** para el átomo es el valor intrínseco modificado según la ecuación ***2.2***:

$$S(A_i) = I_i + \Delta I_i \qquad [2.2]$$

Donde ΔI_i cuantifica el efecto perturbativo sobre el valor intrínseco del átomo; esta perturbación se asume que es una función de la diferencia entre los valores intrínsecos I_i y I_j según la ecuación ***2.3***:

$$\Delta I_i = \sum (I_i - I_j)/R^2_{ij} \qquad i \neq j \qquad [2.3]$$

Donde r_{ij} es el número de átomos en el camino más corto entre los átomos ***i*** y ***j*** incluyendo tanto a ***i*** como ***j***. La diferencia en valores intrínsecos ΔI_i para un par de átomos del esqueleto codifica atributos electrónicos y topológicos relacionados con las diferencias de electronegatividad y de conectividad en el esqueleto.

- ✓ **Índice del Estado Refractotopológico (R-state, $\mathcal{R}_i$)** [8]

Es un índice atómico híbrido que depende de un valor intrínseco del átomo que es una función de la distancia topológica a los átomos de su entorno. Se diferencia del Índice del Estado Electrotopológico en que su valor intrínseco es el valor de su refractividad atómica y la de los átomos de hidrógeno a él enlazados. Este valor se modifica por la refractividad atómica de todos los átomos de su entorno.

El índice del Estado Refractotopológico se desarrolla a partir de la teoría del grafo químico y de la partición de la refractividad atómica definida por Ghose y Crippen.

El **(R-state, $\mathcal{R}_i$),** para un átomo ***i*** se define por la ecuación ***2.4***:

$$R_i = AR_i + \Delta AR_i \qquad [2.4]$$

Donde ***ARi*** es el valor de refractividad intrínseco del átomo ***i*** y ΔARi es un término de perturbación definida por la ecuación ***2.5***:

$$\Delta AR_i = \sum (AR_i - AR_j)/R^2_{ij} \qquad [2.5]$$

Donde se suman todos los vértices ***j*** adyacentes en el grafo, ***ARi*** y ***ARj*** son los valores intrínsecos de la refractividad de los átomos ***i*** y ***j***, respectivamente, y R^2ij es el número de átomos del camino más corto entre los átomos ***i*** y ***j***, incluyendo tanto a ***i*** como a ***j***. Al igual que

en el Estate y en el Sstate (Índice del Estado Electrotopográfico), la distancia topológica cuadrática indica que debe haber una disminución de la interacción, con el aumento de la distancia de separación entre los átomos. A los valores de la refractividad de cada uno de los átomos pesados se le adicionan los valores de refractividad atómica de los átomos de hidrógeno a él enlazados.

- ✓ **Índice del Estado Electrotopográfico**

Este índice es similar que el Índice del Estado Electrotopológico pero utilizando la matriz de conectividad entre vértices ponderada por la distancia de enlace.

- ✓ **Índice del Estado Refractotopográfico**

Este índice es similar que el Índice del Estado Refractotopológico pero utilizando la matriz de conectividad entre vértices ponderada por la distancia de enlace.

2.2.6.2 Descriptores Moleculares

- ✓ **Índice de Randić** [8]

El índice de Conectividad de Randić ${}^{p}\chi$ es un índice topológico basado en la matriz de adyacencia entre los vértices del grafo molecular y se define por:

$$ {}^{p}\chi = \sum [\delta(V_i)\ \delta(V_j)]^{-1/2} \qquad [2.6] $$

Donde el grado del vértice $\delta(Vi)$ es el número de vértices con los cuales el átomo ***i*** está enlazado directamente, y la suma es sobre todo los vértices adyacentes en la molécula.

- ✓ **Randić Revisitado** [8]

Después de varios estudios y análisis experimentales realizados por Estrada, este sugirió hacer una sustitución en el grado de la ecuación para el cálculo del Índice de Randić de *-1/2* a *-1/3* para resolver el problema de la degeneración de los índices en el caso de los isómeros en familias de compuestos.

- ✓ **Índice de Conectividad de Valencia** [8][20]

El índice de conectividad de Valencia $^{p}\chi$ es una modificación hecha por Kier y Hall al índice de conectividad de Randić y se basa en asignar a cada átomo la cantidad de electrones de valencia presentes.

El descriptor de valencia esta dado por:

$$\delta^{v} = Z^{v} - h \qquad [2.7]$$

Donde Z^{v} es el número de electrones de valencia y h el número de átomos de hidrógeno enlazados al átomo analizado. Para los átomos de la 2da, 3ra y 4ta fila de la tabla periódica se obtiene la expresión general:

$$\delta^{v} = (Z^{v} - h)/(Z - Z^{v} - 1) \qquad [2.8]$$

Donde Z es el número total de todos los electrones.

El índice de conectividad de Valencia se calcula para cada orden y tipo de fragmento. El cálculo se realiza multiplicando el valor δ^{v} de cada átomo de los fragmentos de la molécula y su expresión está dada por:

$$^{p}\chi = \sum_{k=1}^{h+1} \prod \delta_{ik}^{-1/2} \qquad [2.9]$$

- ✓ **Índice de Partición de la Refractividad Molecular** [8]

Es un índice molecular híbrido, se basa en la aplicación del algoritmo de Randić a la matriz de conectividad de los vértices del grafo molecular ponderado por las refractividades atómicas, pero con la diferencia que parte de la matriz del grafo químico completo. Es decir, considera para el cálculo, los átomos de hidrógeno de la molécula y se define según la expresión:

$$^{p}MR_{x} = \sum[\delta^{MR}(V_i)\ \delta^{MR}(V_j)]^{-1/2} \quad i \neq j \qquad [2.10]$$

- ✓ **Indice de Conectividad de las Aristas** [8]

El índice de Conectividad de las Aristas ε es un índice topológico basado en la matriz de

adyacencia entre las aristas del grafo molecular y se define por:

$$\varepsilon = \sum [\delta(e_i)\delta(e_j)]^{-1/2} \qquad [2.12]$$

Donde el grado de la arista $\delta(e_i)$ es el número de aristas adyacentes a e_i, y la suma es sobre todas las aristas adyacentes en el grafo.

- ✓ **Momentos Espectrales** [7]

Los Momentos espectrales de una matriz M_k se definen como las trazas t_r de las diferentes potencias de dicha matriz. O sea el k_{esimo} momento espectral de M_k será la suma de los elementos diagonales de dicha matriz elevada a la potencia *k.*

$$M_k = t_r(E^k) \qquad [2.11]$$

- ✓ **Índice topográfico de conectividad de vértices**

Ese índice fue definido por Estrada. [7] Se utiliza la matriz de conectividad entre vértices ponderada por los órdenes de enlace. Se aplica la ecuación de Randić, donde el grado del vértice ***i*** es la sumatoria de los valores de los elementos de la fila (o columna) ***i.*** Al igual que en éste, se consideran los subgrafos de diferentes órdenes, y el total de términos de la expresión es el número de vértices que componen dicho fragmento o subgrafo.

- ✓ **Índice topográfico revisitado de conectividad de vértices**

Ese índice fue definido por Estrada. [7]Se utiliza la matriz de conectividad entre vértices ponderada por los órdenes de enlace. Se aplica la ecuación de Randić Revisitado, donde el grado del vértice ***i*** es la sumatoria de los valores de los elementos de la fila (o columna) ***i.*** Al igual que en este, se consideran los subgrafos de diferentes órdenes, y el total de términos de la expresión es el número de vértices que componen dicho fragmento o subgrafo.

- ✓ **Modificación al Índice topográfico de Valencia**

El Índice topográfico de Valencia se define como la suma de la matriz de conectividad de los vértices considerando los lazos más la diagonal de la matriz de las cargas netas calculadas por algún método químico-cuántico, según la expresión:

$$\Omega(q) = \sum_{r} [\, \delta^{q}(V_i)\, \delta^{q}(V_j)]_{r}^{-1/2} \qquad [2.13]$$

***r*:** Es el número de pares de vértices adyacentes en el grafo molecular.

Para obviar la detección de los lazos en el grafo químico se decidió ponderar la matriz con las densidades de carga calculadas también por un método semiempírico. La estimación de las cargas y sus densidades electrónicas por estos métodos llevan implícita la inclusión de los electrones de valencia y por ende los pares electrónicos libres que corresponden a los lazos. Esta modificación está sujeta a la validación del índice con estudios de correlación estructura propiedad y a la correlación con el índice original. Estos estudios definirán la diferencia en contenido de información entre ambos.

- ✓ **Índice topográfico de conectividad entre aristas**

  Este índice es similar al índice topológico de Conectividad de las Aristas pero utilizando la matriz de conectividad entre aristas ponderada por la distancia de enlace.

- ✓ **Momentos Espectrales topográficos**

  Este índice es similar a los Momentos Espectrales topológicos pero se utiliza la matriz de conectividad entre aristas ponderada por la distancia de enlace.

2.3 Conclusiones

Se mencionaron las herramientas y los métodos utilizados para implementar los algoritmos: UML como lenguaje para describir este proceso, Visual Paradigm como herramienta CASE, Java como lenguaje de programación y Eclipse como entorno de desarrollo. Se describen los recursos computacionales utilizados. Se expuso de forma general, la teoría de grafos y la representación computacional de estos, utilizada para la realización de este trabajo, así como las técnicas de escritura en pseudocódigo y el cálculo de la complejidad algorítmica. Se describieron los algoritmos basados en el cálculo de descriptores atómicos y moleculares.

CAPÍTULO 3
RESULTADOS Y DISCUSIÓN

3.1 Modelo conceptual del módulo implementado

¿Qué es un modelo conceptual?

Es una idea global sobre los individuos, los grupos, las situaciones y los acontecimientos relacionados a una disciplina. Los modelos conceptuales se construyen a partir de los conceptos que son palabras que describen imágenes mentales de los fenómenos y de las proposiciones que establecen las relaciones entre los conceptos. Por tanto un modelo conceptual es un grupo de conceptos necesarios para describir el entorno en que opera un sistema.

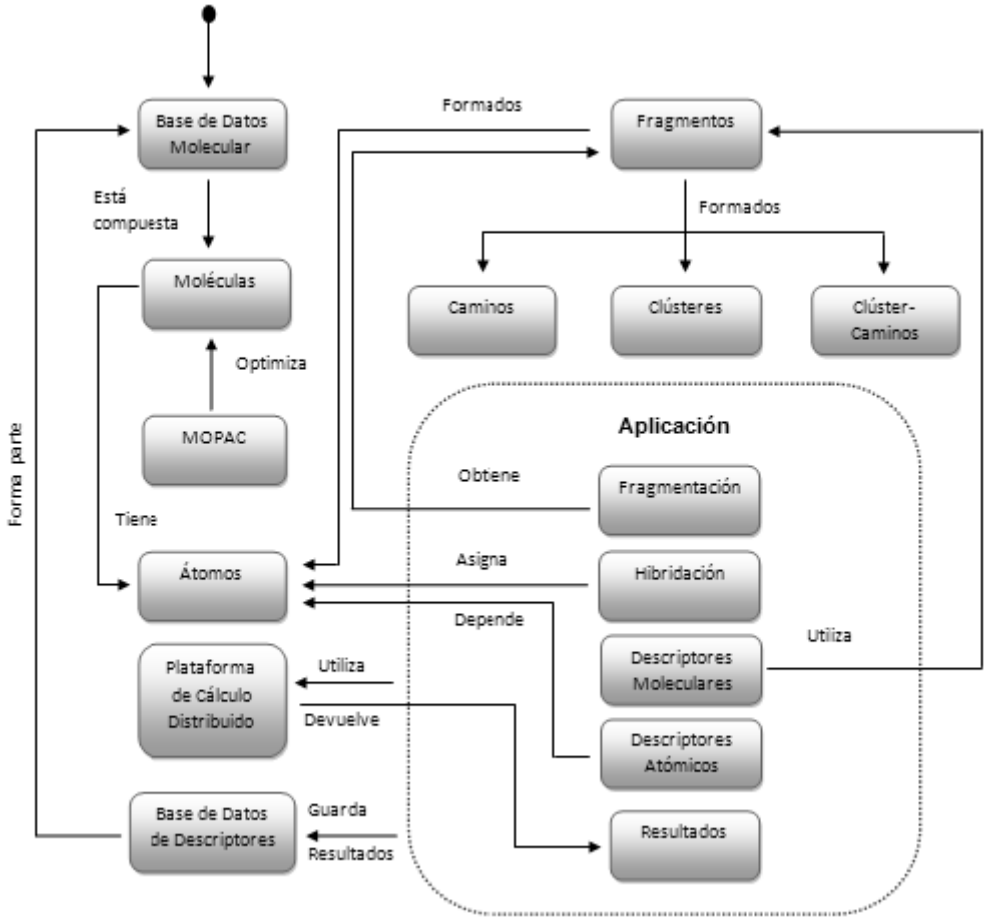


Figura 3.1 Modelo conceptual de las funciones realizadas por la aplicación

Para realizar el cálculo de descriptores [Ver figura 3.1] es necesario obtener las moléculas almacenadas en la base de datos molecular, las cuales son optimizadas por el **MOPAC**. Estas moléculas están compuestas por átomos, a los cuales se les asigna una hibridación, dependiendo de los mismos se realiza el cálculo de los descriptores atómicos y se calcula la fragmentación molecular de la cual se obtienen los caminos, clústeres y clúster-caminos del grafo químico, [Ver epígrafe 2.2.1, 3.3] información imprescindible para el cálculo de los descriptores moleculares. Todo este procedimiento se realiza utilizando la plataforma de cálculo distribuido. Los resultados obtenidos se guardan en la base de datos de descriptores la cual forma parte de la base de datos molecular. Estos son utilizados por otras aplicaciones para la realización de sus funcionalidades.

3.2 Diagrama de clases

En el presente epígrafe se muestra una representación gráfica de las clases y las relaciones que se implementaron para realizar el cálculo de los descriptores.

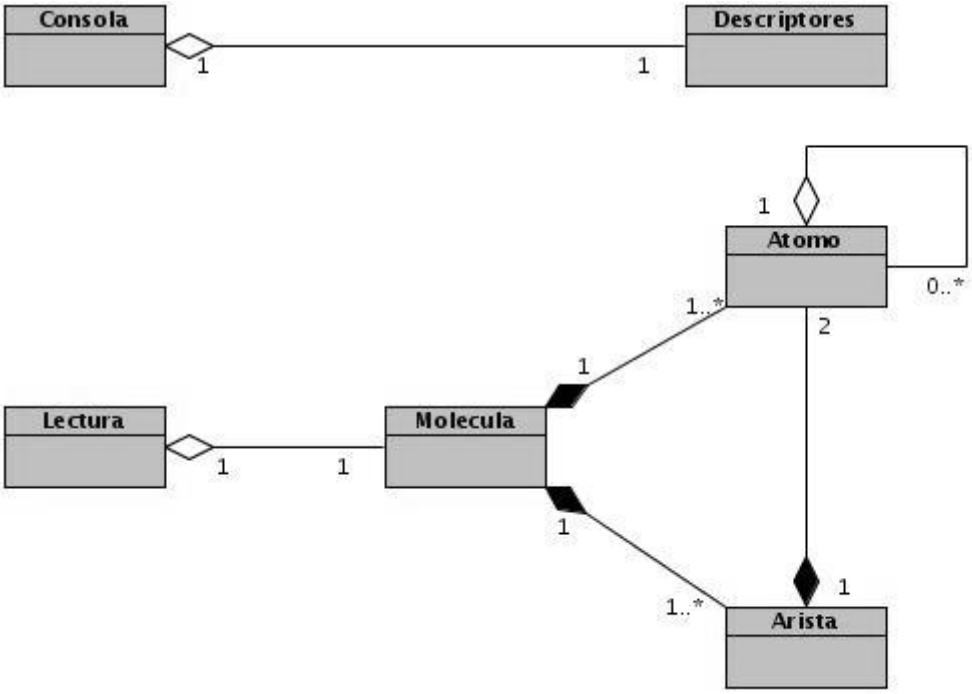


Figura 3.2 Diagrama de clases implementadas

3.2.1 Breve descripción de las clases implementadas

- ✓ ***Consola*:** Es la clase que se encarga de guardar en un fichero texto ***.dsc***, la hibridación de cada uno de los átomos, los fragmentos y los resultados de los descriptores del grafo molecular.

- ✓ ***Descriptores:*** Contiene la implementación de los algoritmos de los descriptores. [Ver epígrafe 2.2.6]

- ✓ ***Molecula:*** Contiene todas las funcionalidades que se realizan sobre el grafo químico, los átomos y las aristas que lo conforman, así como el nombre de la molécula que representa.

- ✓ ***Atomo*:** Esta formada por los atributos que representan al átomo, como el tipo de elemento, el identificador, la lista de átomos adyacentes a él, número atómico, cantidad de electrones de valencia, coordenadas del átomo x, y, z, la hibridación asignada, el valor de la refractividad atómica y el valor de la carga atómica.

- ✓ ***Arista*:** Está formada dos átomos (inicial y final), el tipo de enlace que la caracteriza, la longitud de la arista desde el átomo inicial al átomo final y su identificador.

- ✓ **Lectura:** Contiene las funcionalidades para realizar la lectura del fichero de la molécula (***.mol***) y crear el grafo molecular.

3.3 Principales algoritmos implementados

Descripción de los algoritmos

La base de datos molecular contiene las moléculas representadas en ficheros de texto ***.mol***. Estos contienen información de la molécula, como los átomos, junto con sus coordenadas en tres dimensiones y la densidad de carga atómica, además de las aristas con sus respectivos átomos. La lectura de estos ficheros se realizó para desarrollar la implementación de los descriptores. Para obtener la información brindada por los ficheros ***.mol*** se implementó la clase ***Lectura,*** la que se encarga de crear la lista de átomos y aristas para formar el grafo molecular, es necesario destacar que a cada átomo se le hace corresponder una lista de adyacencia formada por los demás átomos que están enlazados con él, dato significativo para asignarle la ***hibridación*.** Esta función es de gran importancia para la implementación de los algoritmos de los descriptores atómicos y moleculares, y se basa en reconocer cada átomo y su entorno inmediato, o sea, la posición que ocupa dentro de la molécula, la cantidad de elementos enlazados a él y cuáles son esos elementos, para de esta manera

darle una clasificación o parametrización propia, en el **Anexo 2** se observan dichas clasificaciones. Otra función necesaria para lograr el objetivo planteado en este trabajo es la ***fragmentación*** del grafo molecular**,** la cual es responsabilidad de la clase ***Molecula***.

3.3.1 Análisis del algoritmo utilizado para realizar el cálculo del descriptor molecular "Índice de conectividad molecular (Randić)"

El índice de conectividad molecular de ***Randić***, es uno de los índices más empleados en los estudios QSAR debido a su capacidad de describir las moléculas. También se emplea como punto de partida para la definición de nuevos índices, tanto topológicos como topográficos.

¿Cómo se implementó?

Para implementar el algoritmo, [Ver ecuación 2.6] el grafo molecular desprovisto de hidrógeno se fragmenta en subgrafos o fragmentos de diferentes órdenes. Primeramente, se toma un fragmento de un orden y a cada uno de los átomos que componen el fragmento se le halla su grado, que en este caso corresponde con la cantidad de elementos enlazados al átomo que sean distintos de hidrógenos. Los valores correspondientes al grado de los átomos o vértices de un mismo fragmento se multiplican; al resultado se le calcula la potencia -1/2 y este valor se suma con los resultados obtenidos al aplicar las mismas operaciones a los restantes fragmentos de un mismo orden. El resultado final del algoritmo empleado es una colección de valores por cada orden de los fragmentos de la molécula y por cada tipo de estos.

Pseudocódigo y análisis del tiempo de ejecución del algoritmo implementado

función *CalcularIndRandicCaminos* (Molecula *t**)***

x ← 0	([1] 1OE)
► Se asigna a longitud el tamaño de la lista de vértices	
longitud ← ***función*** *size* ()	([2] cOE)
►el bucle se hace hasta 16 para hallar caminos hasta orden 15	
para p ← 1 ***hasta*** longitud ***&&*** *hasta16* ***hacer***	([3] 6OE)
tmp ← ***función*** *TodosLosCaminos* (p)	([4] 3OE + F_1)
val ← 0	([5] 1OE)
longitud1 ← ***función*** *size()* de tmp	([6] cOE)
para i ← 0 ***hasta*** longitud1 ***hacer***	([7] 4OE)
longitud2 ← ***función*** *size*() de tmp en la posición i	([8] cOE)

```
        para j ← 0 hasta longitud2 hacer                    ([9] 4OE)
          var ← 1                                           ([10] 1OE)
          lista ← función get(tmp[i].get(j))                ([11] cOE)
          longitud3 ← función size() de lista               ([12] cOE)
          para k ← 0 hasta longitud3 hacer                  ([13] 4OE)
            at ← función get(k)                             ([14] cOE)
            x ← función CantEnlNOH()                        ([15] (n+2)OE)
            var ← var * x                                   ([16] 2OE)
          fin para
          val ← val + var^(-0,5)                            ([17] 3OE)
        fin para
      fin para
      randic ← función add (val) ► se adicionan los resultados   ([18] (n+3)OE)
    fin para
    devolver randic                                         ([19] 1OE)
fin función
```

Descripción general de las funciones utilizadas

- ✓ *size*: Devuelve la longitud de la lista.
- ✓ *TodosLosCaminos***:** Recibe un parámetro *p* y devuelve un vector con todos los caminos de orden *p***.**
- ✓ *get*: Recibe una posición y devuelve el elemento que está en esa posición.
- ✓ *CantEnlNOH*: Devuelve la cantidad de enlaces de un átomo que sean distintos de hidrógenos.
- ✓ *add*: Recibe un valor y lo adiciona al final de una lista.

Para determinar la complejidad en tiempo del algoritmo, se calcula primero el número de operaciones elementales (OE) que se realizan:
En la línea (1) se ejecuta 1 OE (una asignación).
En la línea (2) se efectúa c OE (una asignación y una llamada a un método, la cual es constante).
En la línea (3) se efectúa la condición del bucle *FOR* con 6 OE (dos asignaciones, dos comparaciones, un incremento y el *AND*).

La línea (4) tiene 3 OE más el tiempo de F_1 OE (una asignación, uno por la llamada y uno por el parámetro mas el tiempo del método F_1).
La línea (5) tiene una asignación (1 OE).
La línea (6) tiene una asignación y una llamada a un método constante(c OE).
En la línea (7) se efectúa la condición del bucle *FOR* con 4 OE (dos asignaciones, una comparación, un incremento).
La línea (8) tiene c OE (una asignación, un acceso al vector, el tiempo de la función (c OE)).
La línea (9) se efectúa la condición del bucle *FOR* con 4 OE (dos asignaciones, una comparación, un incremento).
La línea (10) tiene una asignación (1 OE).
La línea (11) tiene c OE (una asignación, un acceso al vector y el tiempo de la función (c OE))
La línea (12) tiene c OE (una asignación y el tiempo de la función (c OE)).
En la línea (13) se efectúa la condición del bucle *FOR* con 4 OE (dos asignaciones, una comparación, un incremento).
La línea (14) tiene c OE (una asignación y el tiempo de la función (c OE)).
La línea (15) tiene (On+2) OE (una asignación (1 OE) y uno por la llamada + el tiempo de la función (n OE)).
La línea (16) tiene una asignación y una multiplicación (2 OE).
La línea (17) tiene una asignación, una suma y una potencia (3 OE).
La línea (18) tiene (n + 3) OE (uno por la llamada y una por el parámetro el tiempo de la función *ADD* (n OE)).
La línea (19) contiene un *RETURN* (1 OE).
c: es un número natural mayor que 0.

En el peor de los casos, se ejecutan las líneas (1) y (2), el *FOR* de la línea (3) se repite hasta 15 veces, se ejecutan las líneas (4, 5, 6); el *FOR* de la línea (7) y nuevamente se ejecutará n veces, la línea (8) y el *FOR* de la línea (9) y nuevamente se ejecutará n veces, se efectúan las líneas (10, 11, 12), se hace el *FOR* de la línea (13) y se ejecutan las líneas (14, 15, 16), al terminar el *FOR* de la línea (13), se ejecuta la línea (17), al terminar el *FOR* de la línea (7) se ejecuta la línea (18) y por último se termina el *FOR* de la línea (3) y la función termina en la línea (19).

$T(n) = T(C) + 15 * (T(S) + 3 + F_1 + T(C)) + 1$ **[I]**
$T(C) = 6$

$T(S) = T(C_1) + n*(T(S_1) + n + c + T(C_1))$ **[II]**

$T(C_1) = 4$

$T(S_1) = T(C_2) + n*(T(S_2) + c + T(C_2))$ **[III]**

$T(C_2) = 4$

$T(S_2) = T(C_3) + n*(T(S_3) + T(C_3))$ **[IV]**

$T(C_3) = 4$

$T(S_3) = n+c$

Sustituyendo en IV:

$T(S_2) = 4 + n*(n+c + 4)$

$= n^2 + cn + 4n + 4$

$= n^2 + cn + 4$

Sustituyendo en III:

$T(S_1) = 4 + n*(n^2 + cn + 4 + c + 4)$

$= n^3 + cn^2 + cn + 8n + 4$

$= n^3 + cn^2 + cn + 4$

Sustituyendo en II:

$T(S) = 4+ n*(n^3 + cn^2 + cn + 4 + n + c + 4)$

$= n^4 + cn^3 + cn^2 + 4n + n^2 + cn + 4n + 4$

$= n^4 + cn^3 + cn^2 + cn + 4$

Sustituyendo en I:

$T(n) = 6 + 15 *(n^4 + cn^3 + cn^2 + cn + 4 +3 + F_1 + 6) +c$

$= 15n^4 + cn^3 + cn^2 + cn + 15F_1 + 201 + c$

$= 15n^4 + cn^3 + cn^2 + cn + 15F_1 + c$

Sustituyendo $\mathbf{F_1}$:

$T(n) = 15n^4 + cn^3 + cn^2 + cn + 15F_1 + c$

$= 15n^4 + cn^3 + cn^2 + cn + 15(3n^4 + 10n^3 \log n + cn^3 + cn^2 + cn + c) + c$

$= \mathbf{50n^4 + 150n^3 \log n + cn^3 + cn^2 + cn + c}$

La cota superior del algoritmo es: $O(n^4)$.

3.3.2 Análisis del algoritmo utilizado para realizar el cálculo del descriptor atómico "Índice del Estado Refractotopográfico (Rstate)"

Este es un índice atómico, topográfico e híbrido definido por Carrasco [8] y Padrón [8], el cual se implementa por primera vez de manera profesional. Este índice incluye información topográfica al sustituirse en la expresión, el número de vértices en el subgrafo por la distancia euclidiana calculada por un método semiempírico. Los valores intrínsecos de cada átomo se mantienen inalterados, modificándose solamente la distancia.

¿Cómo se implementó?

Para implementar las ecuaciones descritas, **[Ver ecuaciones 2.4, 2.5]** debe partirse del grafo químico desprovisto de hidrógenos. Primeramente se toma un átomo y se calcula su valor de refractividad atómica, este valor parte de la hibridación del átomo **[Ver Anexo 2]** y a este se le suma el valor de refractividad de todos los hidrógenos enlazados al átomo tratado, es importante aclarar que cuando se habla del grafo químico desprovisto de hidrógenos es porque sólo se tienen en cuenta los demás átomos para realizar el algoritmo, pero los hidrógenos existentes en el grafo sí son necesarios para realizar otras operaciones secundarias y esto es validado cuando se realiza la asignación de la ***hibridación***. El valor obtenido en la anterior operación es sumado con el efecto perturbativo que genera el átomo sobre todos los demás que componen la molécula. El efecto perturbativo no es más que la diferencia entre la refractividad atómica del átomo tratado y la del resto de los átomos, dividido por la distancia topográfica mínima elevada al cuadrado, entre el átomo tratado y el resto. Todo este proceso se realiza para cada átomo de la molécula. El resultado final del algoritmo es una colección de valores que corresponden con el número de átomos de la molécula distintos de hidrógeno, más un último valor que corresponde a la suma de todos los índices. Así, si se tienen 20 átomos en la molécula, entonces se obtienen 20 índices más un total.

Pseudocódigo y análisis del tiempo de ejecución del algoritmo empleado.

```
función Calcular_Estad_Refractopográfico(Molecula t)
    elem ← función CrearVector()                                  ([1]  2OE)
    tmp ← función GetAtomos()                                     ([2]  2OE)
    atomos ← función  CrearVector<Atomo>()                        ([3] 2OE)
    longitud ← función size()                                     ([4] cOE)
    para i ← 0 hasta longitud hacer                               ([5] 4OE)
        si función get (i)  ≠  "H"                                ([6] cOE)
```

```
        atomos ← función add(get(i))                                    ([7] n+c OE)
       fin si
    fin para
    longitud1 ←  función size()                                         ([8] cOE)
    para i ← 0 hasta longitud1 hacer                                    ([9] 4OE)
      valorRefractividadIntrinseco ← función ValorRefracIntrinseco(función get(i))   ([10] cn + cOE)
      perturbación ← función CalcularEfectoTopografico(función get(i), atomos, t)
      ([11] 2n^2 + cn + cOE)
      índice ← valorRefractividadIntríseco + perturbación               ([12] 2OE)
      elem ← función (función get(i))                                   ([13] n+cOE)
      elem ← función add (indice)                                       ([14] n+2OE)
      indice ← 0                                                        ([15] 1OE)
      estRefractopográfico función add (i, función clone())             ([16] n+cOE)
      ► est_refractopográfico guarda por cada posición un arreglo;
        que contiene en la primera posición el  átomo y en la segunda;
        su el índice calculado
      elem función clear()                                              ([17] cOE)
    fin para
  devolver  est_refractopográfico                                       ([18] 1OE)
  fin función
```

Descripción general de algunas funciones utilizadas

- ✓ *size*: Devuelve la longitud de la lista.
- ✓ *get*: Recibe una posición y devuelve el elemento que está en esa posición.
- ✓ *add*: Recibe un valor y lo adiciona al final de una lista.
- ✓ *ValorRefracIntrinseco*: Recibe un átomo y le calcula su valor intrínseco de refractividad atómica.
- ✓ *CalcularEfectoTopografico*: Recibe átomo que se encuentra en la posición i, la lista de átomos y una instancia del grafo t.
- ✓ *CrearVector*: Crear un vector para guardar una colección de elementos.

- ✓ *GetAtomos*: Devuelve una referencia a la lista de átomos.
- ✓ *clone*: Hace una copia de la lista.

Para determinar la complejidad en tiempo, se calcula primero el número de operaciones elementales (OE) que se realizan:
En la línea (1) se ejecutan 2 OE (una asignación y la llamada al método).
En la línea (2) se ejecutan 2 OE (una asignación y la llamada al método).
En la línea (3) se ejecutan 2 OE (una asignación y la llamada al método).
En la línea (4) se ejecutan c OE (una asignación y la llamada al método constante).
En la línea (5) se efectúa la condición del bucle *FOR* con 4 OE (dos asignaciones, una comparación, un incremento).
La línea (6) está formada por una condición, una comparación, la llamada a un método constante y dos parámetros (c OE).
La línea (7) está compuesta por dos llamadas a métodos y un parámetro (3 OE) por los métodos (uno constante (c OE)) y el otro (cn +c OE) y finalmente (cn + c OE).
La línea (8) tiene c OE (una asignación y la llamada + el tiempo de la función (c OE)).
La línea (9) se efectúa la condición del bucle *FOR* con 4 OE (dos asignaciones, una comparación, un incremento).
La línea (10) tiene una asignación (1 OE), dos llamada a funciones (tiene un método constante (c OE) y la otra función (n + c OE)) y un parámetro (1 OE), se tiene (n + c OE).
La línea (11) tiene una asignación (1 OE), dos llamada a funciones (tiene un método constante (c OE) y la otra función ($2n^2$ + cn + c OE)) y un parámetro (1 OE) se tiene ($2n^2$ + cn + c).
La línea (12) tiene una asignación y una suma (2 OE)).
La línea (13) tiene dos llamadas a métodos (uno constante (c OE) y otro(n OE)) más un parámetro (1 OE), se tiene (n + c OE).
La línea (14) tiene la llamada a una función y su parámetro (n + 2 OE).
La línea (15) tiene una asignación.
La línea (16) está compuesta por dos llamadas a métodos y un parámetro (3 OE) por los métodos (uno constante (c OE)) y el otro (n OE) y finalmente (n + c OE).
La línea (17) tiene la llamada a una función constante(c OE).
La línea (18) contiene un RETURN (1 OE).

En el peor de los casos: comienza ejecutando las líneas (1, 2, 3, 4), se hace el *FOR* de la línea (5) y se

ejecuta n veces, se evalúa la condición de la línea (6), en caso de cumplirse se hace la línea 7, se continua hasta que no se cumpla la condición y sale del ciclo, se ejecuta la línea (8), y ejecuta el *FOR* de la línea (9) y se ejecuta n veces haciendo antes las líneas (10, 11, 12, 13, 14, 15, 16, 17) y finaliza ejecutando la línea (18).

Complejidad:

$T(n) = T(C) + n*(T(S) + T(C)) + T(C_1) + n*(T(S_1) + T(C_1)) + 7 + 2c$ **[I]**

$T(C) = 4$

$T(C_1) = 4$

$T(S) = n + c$

$T(S_1) = cn + c + 2n^2 + cn + c + 2(n + c) + n + 2 + 3 = 2n^2 + 2cn + n + 4c = 2n^2 + cn + n + c$

Sustituyendo en **I**:

$T(n) = T(C) + n*(T(S) + T(C)) + T(C_1) + n*(T(S_1) + T(C_1)) + 5 + c$

$= 4 + n*(n + c + 4) + 4 + n*(2n^2 + cn + n + c + 4) + 5 + c$

$= n^2 + cn + 4n + 2n^3 + cn^2 + n^2 + cn + 4n + c + 13$

$= 2n^3 + 2n^2 + cn^2 + 2cn + 8n + c + 13$

$= \mathbf{2n^3 + cn^2 + cn + c}$

La cota superior del algoritmo es: $O(n^3)$

Complejidad de la función *CalcularEfectoTopografico*:

$T(n) = \mathbf{2n^2 + cn + c}$

Complejidad de la función *ValorRefracIntriseco:*

$T(n) = \mathbf{cn + c}$

3.4 Programación en el sistema distribuido [17]

Para ejecutar un cómputo distribuido, el programador necesita solamente extender dos clases en Java: ***DataManager*** y ***Algorithm***, que vienen programadas y predefinidas en el sistema. También se pueden usar bibliotecas adicionales de Java u otras clases definidas por el usuario. [Ver Anexo 3]

Clase DataManager

El propósito de ***DataManager*** es generar todas las unidades de trabajo que se enviarán a los clientes, procesar todos los resultados devueltos por los clientes, ajustar el particionamiento de las unidades del

trabajo, generar la información del estado del problema y terminar el cómputo distribuido. El desarrollador debe implementar una clase que herede de ***DataManager*** para dar solución a un problema particular. En la ***Figura 3.3*** se observa la declaración de la clase implementada.

```
public class ManagerMol extends DataManager {

    /*
     * Declaración de variables
     */

    public ManagerMol() throws Throwable {

    public void adjustGranularity(double arg0) throws Throwable {

    public void closeResources() throws Throwable {

    public Vector generateWorkUnit(ClientInfo arg0) throws Throwable {

    public String getStatus() throws Throwable {

    public boolean processResults(Long arg0, Vector results) throws Throwable {

}
```

Figura 3.3 Declaración de la clase *ManagerMol*

Métodos implementados de la clase *ManagerMol*

- ✓ ***ManagerMol():*** Es el constructor de la clase ***ManagerMol***, encargado de inicializar todas las variables y el responsable del manejo de los paquetes que se van a enviar a los clientes.

- ✓ ***closeResources():*** El propósito de este método es cerrar cualquier recurso que puede estar abierto al concluir el proceso.

- ✓ ***generateWorkUnit(ClientInfo arg0)***: Este método es llamado por la plataforma de cálculo distribuido cada vez que un cliente solicita una unidad del trabajo para procesar. El tipo de retorno del método es un Vector que representa a una colección de datos. El algoritmo que funciona en el cliente recibe este Vector como su unidad del trabajo para procesar.

- ✓ ***processResults(Long arg0, Vector results)***: Este método es llamado cada vez que un cliente devuelve un conjunto de resultados. Contiene dos parámetros, el primero es el identificador (ID) de la unidad del trabajo, que en este caso coincide con el nombre de una molécula y el segundo es el Vector que representa el conjunto de resultados que fueron devueltos por el algoritmo (*Algorithm*) que funciona en el cliente, es decir el resultado del cálculo de los descriptores. Si el cómputo distribuido termina entonces este método debe retornar *true*, si no *false*.

Clase Algorithm

La clase ***Algorithm*** tiene parte del código que corre en el cliente y procesa las unidades de trabajo generadas por el método ***generateWorkUnit*** de la clase ***DataManager***.
En la ***Figura 3.4*** se observa la declaración de la clase implementada que hereda de ***Algorithm***

```
public class ProcesMolAlgorithm extends Algorithm {
    |
    public Vector processUnit(Vector arg0) throws Throwable {

    public void initProcess() throws FileNotFoundException {

    public void optimizarMol(File fout, File fmol, String dirGuardar){
}
```

Figura 3.4 Declaración de la clase ProcessMolAlgorithm

Métodos implementados

- ✓ ***processUnit(Vector arg0):*** Encargado de procesar la unidad de trabajo enviada por el servidor y retornar un vector que represente el conjunto de resultados.
- ✓ ***initProcess()*:** Este método es invocado por la función ***proccesUnit*** para descargar el fichero ***jar,*** el cual contiene los algoritmos implementados para el cálculo de descriptores, así como el **MOPAC**.
- ✓ ***optimizarMol(File fout, File fmol, String dirGuardar):*** Este método es invocado por la función ***proccesUnit*** para optimizar las coordenadas de los átomos.

3.5 Resultados experimentales

3.5.1 Comparación de los resultados obtenidos

Para comprobar el correcto funcionamiento de los algoritmos implementados y la veracidad de los resultados obtenidos, se realizó una comparación entre estos, y los arrojados por el Dragon [Ver Epígrafe 1.4.2] al calcular varios descriptores en la molécula Alanina [Ver Anexo 4]. Es válido destacar que el Dragon no calcula descriptores topográficos, y sólo calcula fragmentos camino hasta orden cinco sin incluir clústeres ni las correspondientes combinaciones clústeres/camino.

Los resultados obtenidos al calcular los índices Momentos Espectrales, tanto topológico como topográfico se analizaron realizando un algoritmo de multiplicación de matrices con el programa matemático ***MatLab,*** y los datos obtenidos fueron similares.

Índice de Randić					
Fragmentos	**Caminos**			**Clúster**	**Clúster/Camino**
Orden	**1**	**2**	**3**	**3**	**3/1**
Aplicación	2.6427344	2.4880339	1.3333333	0.6666667	1.3333333
Dragon	2.643	2.488	1.333		
Índice de Valencia					
Fragmentos	**Caminos**			**Clúster**	**Clúster/Camino**
Orden	**1**	**2**	**3**	**3**	**3/1**
Aplicación	1.6270897	1.1269128	0.3895276	0.2193713	0.5930026
Dragon	1.627	1.127	0.39		
Índice de conectividad entre aristas					
Fragmentos	**Caminos**			**Clúster**	**Clúster/Camino**
Orden	**1**	**2**	**3**	**3**	**3/1**
Aplicación	3.3284271	2.4142136	1	0.5	0.3535534
Dragon	3.328	2.414			

Tabla 3.1 Comparación con el Dragon de algunos resultados obtenidos

3.5.2 Ejecutando la aplicación de forma distribuida

Como se explicó en el *Epígrafe 3.3,* para realizar el cálculo de los descriptores se parte de la lectura de un fichero texto ***.mol*** de la estructura optimizada por el **MOPAC**. Este programa lee como entrada el fichero ***.mop***, que se encuentra en la base de datos y devuelve un fichero ***.out***. Con lo obtenido del fichero de salida se crea un nuevo ***.mol***, que es el que recibe como entrada el fichero ***.jar*** que posee los algoritmos implementados para la realización de los cálculos de descriptores. Es importante destacar que la estructura molecular puede alterar la rapidez de los cálculos a desarrollar, los cuales son directamente proporcionales al tamaño de la molécula, aunque éste no es el único aspecto a considerar para establecer las causas de un mayor tiempo de cómputo, cómo se podría pensar. La aplicación implementada realiza simultáneamente el cálculo de descriptores topológicos y topográficos. Esto constituye una novedad técnica pues entre los software reportados en la literatura especializada no se encuentra ninguno con estas características. Esta combinación de posibilidades da lugar, por supuesto, a una aplicación más pesada y que para la ejecución del cálculo de los topográficos, sea

más lenta. Este aparente inconveniente es superable pues se le ofrece al investigador una nueva potencialidad de cálculo con la que anteriormente no contaba. A esto se le puede agregar el hecho de que al estar implementada de forma distribuida, lleva implícita la economía en los tiempos de ejecución del programa.

En este epígrafe se demuestra cuánto influye en la rapidez de los cálculos la utilización de los algoritmos implementados de forma distribuida y la complejidad de la estructura del grafo químico. Para conocer las relaciones antes expuestas, se hizo uso de la estadística. Se realizó un estudio preliminar de los resultados obtenidos mediante técnicas de análisis de regresión lineal simple y múltiple, para lo que se utilizó el programa **StatGraphics 3.1**. En correspondencia con esto se realizaron los siguientes experimentos:

Experimento 1:

Se calcularon los descriptores a dos moléculas con diferentes estructuras y similar cantidad de átomos. La primera molécula, presenta una estructura básicamente lineal, la segunda, contiene una estructura más ramificada. En la ***Figura 3.5*** se observa el tiempo de cálculo de cada molécula.

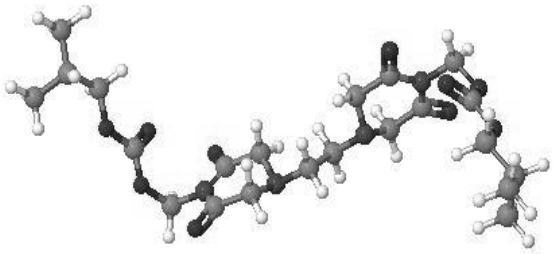
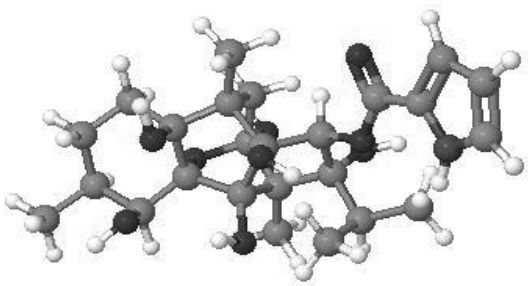
	
Cantidad de átomos - 70	**Cantidad de átomos - 70**
Cantidad de átomos ≠ hidrógeno - 36	**Cantidad de átomos ≠ hidrógeno - 35**
Cantidad de clústeres - 12	**Cantidad de clústeres - 14**
Tiempo de cálculo - $2_{min}16_{seg}$	**Tiempo de cálculo - $2_{min}47_{seg}$**

Figura 3.5 Tiempo de los cálculos realizados a dos moléculas con diferentes estructuras

Para explicar en cuánto está determinado o influyen en la predicción del tiempo (seg) de cálculo, la cantidad de átomos, órdenes de los caminos y números de clústeres, se tomó una pequeña muestra de 30 moléculas, a las cuales se le calcularon todos los algoritmos implementados, en un único ordenador.

Con los resultados que se obtuvieron, se realizó un análisis para tratar de explicar en cuánto influye cuantitativamente la estructura molecular en el comportamiento del tiempo de cómputo, Para ello se realizó un análisis de regresión lineal múltiple. Se obtuvo la ecuación *I*, que muestra la dependencia del tiempo con respecto a las variables: cantidad de átomos, orden de los caminos y cantidad de clústeres.

$$\textbf{\textit{tiempo}} = -27.99(\pm 13.21) + 6.25(\pm 1.79)\textbf{\textit{clústeres}} + 1.53(\pm 2.20)\textbf{\textit{caminos}} + 0.44(\pm 0.46)\textbf{\textit{átomos}} \quad [\mathrm{I}]$$

$$\textbf{\textit{R}}^2 = 67{,}10 \qquad \textbf{\textit{n}} = 30 \qquad \textbf{\textit{s}} = 22.92 \qquad \textbf{\textit{F}} = 17.00$$

En la ecuación cada coeficiente representa el efecto que causa en el tiempo un aumento unitario de las variables. La eficiencia del modelo puede valorarse por los valores de la ***F*** de ***Fischer*** y el valor de la desviación estándar ***s*** del modelo. El coeficiente de determinación múltiple $\textbf{\textit{R}}^2$ representa el porciento de variabilidad del tiempo en función de las demás variables y se utiliza para medir la tendencia de comportamiento de la variable dependiente con respecto a la variación de las independientes. Para este tipo de experimento, este coeficiente se considera aceptable a partir del 30% [21] y tiende a aumentar según la cantidad de variables que intervengan en el modelo. Debido a esto, las principales dificultades para ajustar un modelo de regresión múltiple surgen cuando es necesario identificar entre el conjunto de variables disponibles aquellas que están relacionadas con la respuesta y que la predicen de la mejor forma posible. En el modelo obtenido, el nivel de significación de dos de las variables (**P_Value**) es alto, lo que determina la hipótesis de que estas no tienen relación significativa con el tiempo. Para plantear lo contrario, el nivel de significación de las variables debe ser menor que 0.05.

Variables	P_Value
Constante	0.042
Átomos	0.3406
Caminos	0.4930
Clústeres	0.018

Tabla 3.2 P_Value de las variables contenidas en la ecuación I

Como se observa en la ***Tabla 3.2***, sólo los clústeres presentan un valor importante en cuanto a la variabilidad del tiempo, lo que forzó la búsqueda de un mejor modelo.

A través de experimentos realizados con el programa **StartGraphics**, se obtuvo como mejor modelo el que está dado por la ecuación ***II*:**

$$\textbf{tiempo} = -23.24(\pm 11.20) + 7.05(\pm 1.37)\textbf{clústeres} + 0.69(\pm 0.29)\textbf{átomos} \qquad [II]$$

$R^2 = 66.46$ $n=30$ $s=22.69$ $F= 25.77$

En la ecuación *II* disminuye el número de variables, así como el valor de R^2. No obstante, se incrementa apreciablemente el valor de la ***F*** de ***Fischer*** y disminuye ligeramente el valor de la desviación estándar. La reducción del número de variables a dos, permite obtener un modelo en el que todas ellas son significativas como se puede deducir de los valores del error de cada variable.

Variables	P_Value
Constante	0.04
Átomos	0.02
Clústeres	0.00

Tabla 3.3 P_Value de las variables contenidas en el modelo II

El modelo obtenido sólo debe ser usado con valores que estén en el rango de los obtenidos en la muestra, de no ser así, se incurre en una extrapolación de valores que afectaría la precisión de la estimación. [21]

Los resultados de estas estimaciones sugieren que la presencia de un mayor número de clústeres es el principal factor que afecta la cantidad de tiempo de ejecución. Esto coincide con observado en la ***Figura 3.5*** en el que la principal diferencia entre ambas moléculas está en este aspecto. Con la realización de este experimento se demostró cuánto influye la estructura molecular en la velocidad de los cálculos.

Experimento 2:

Se seleccionan 69 grupos de moléculas diferentes. Cada grupo contiene una cantidad variable de moléculas. Se grafican los tiempos de ejecución de la aplicación para cada grupo, contra el número de moléculas que contiene dicho grupo. [Ver Figura 3.6] Todos los cálculos se realizaron primeramente en un sólo ordenador. Posteriormente se repitió empleando la plataforma de cálculo distribuido.

El método para la selección del tamaño de la muestra fue el siguiente: Se fijó un intervalo entre 20 y 250 para seleccionar la cantidad de grupos de moléculas (muestras) lo que da un total de 69 grupos. En cada una de las decenas se seleccionaron tres números aleatorios que corresponderán a la cantidad de moléculas del grupo. Esto hace que en la medida que se avanza en las decenas se incrementa el tamaño de cada grupo. Esta distribución puede observarse en la ***Tabla 3.4***, donde las columnas **CM** representan la cantidad de moléculas y las columnas **T1** y **T2** el tiempo de cómputo en un ordenador y en la plataforma de cálculo distribuido respectivamente.

CM	T1	T2	CM	T1	T2	CM	T1	T2
23	4.1	1	99	17.7	4	178	45.9	6
24	4.3	1	104	16.3	3	179	42.8	7
27	7.3	3	108	19.7	3	180	39.3	6
35	5.7	1	109	25.9	4	185	43.6	6
37	6.3	2	112	20.6	4	189	47.9	6
38	8.2	2	114	27.1	4	192	40.4	5
40	10.4	2	117	20.1	4	194	42.8	7
45	9.2	1	124	32.6	5	197	38.9	5
46	8.4	1	125	34.4	5	200	40.3	6
54	8.2	1	127	39.5	4	205	41.7	5
57	12.9	3	132	37.8	4	208	41.1	7
59	10.2	2	135	46.2	5	212	48.9	7
60	9.1	3	138	39.6	6	214	51.2	6
67	11.8	3	141	35.7	5	218	46.4	8
68	14.4	3	147	45.2	5	220	46.9	7
71	14.2	3	149	38.8	4	225	47.5	6
73	17.5	3	150	39.8	6	229	53.4	7
79	15.2	2	157	40.1	6	230	48.3	6
81	16.1	2	159	37.1	5	232	53.9	7
82	22.5	3	161	42.4	5	236	49.1	6
89	27.5	5	162	39.9	6	245	51.3	6
90	16.6	3	167	37.2	5	246	58.3	7
95	18.3	3	173	40.3	4	249	49.1	6

Tabla 3.4 Representación de la muestra analizada y el tiempo de cálculo

En la ***Figura 3.6*** se muestra la gráfica de dispersión relacionada con el tiempo de cómputo que se necesitó para calcular las muestras en un ordenador y de forma distribuida. Destacar que la plataforma de cálculo distribuido devuelve los resultados en minutos, despreciando los segundos.

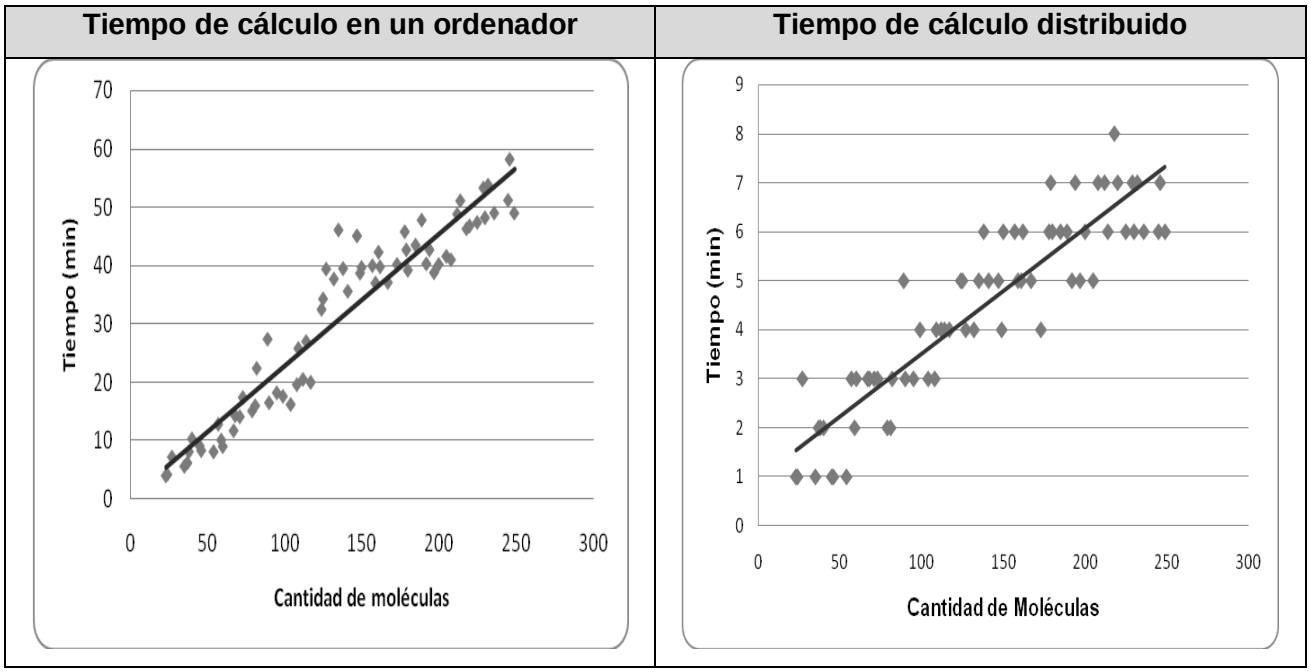


Figura 3.6 Gráfica de dispersión del tiempo de cálculo

Para obtener el grado de asociación existente entre el tiempo de cómputo y la cantidad de moléculas procesadas al aumentar la carga, y poder predecir el comportamiento de este para un número de moléculas dado, se realizó un estudio de los datos obtenidos utilizando regresión lineal simple. Se obtuvo un modelo de regresión dado por la ecuación *III*.

$$\textbf{\textit{tiempo}} = 0.125335(\pm 1.30) + 0.226861\textbf{\textit{cantidadMoléculas}} \qquad [\text{III}]$$

$\boldsymbol{R^2}$=91.04 $\boldsymbol{n}$=69 $\boldsymbol{s}$=4.69 $\boldsymbol{F}$=681.02

Los resultados de este estudio arrojaron un coeficiente de determinación lineal de un 91.04%, el cual se considera muy bueno ya que éste determina el porciento de la variabilidad del tiempo explicado por la recta de regresión. [22]

Este modelo presenta un nivel de significación de 0.00, lo que lo convierte en una potente herramienta para la predicción del tiempo en función de la cantidad de moléculas.

Realizando una sustitución por la cantidad de moléculas que se quiere calcular, entonces se

demoraría un tiempo de: **28 días** y **19 horas** de cálculo ininterrumpido en un sólo ordenador, algo computacionalmente costoso.

Estas mismas operaciones se realizaron con los valores obtenidos al procesar la misma cantidad de moléculas de forma distribuida en una red local de solamente siete máquinas. Con lo que se creó el modelo ***IV*:**

$$\textbf{tiempo} = 0.96256(\pm 0.21) + 0.0255109\,\textbf{\textit{cantidadMoléculas}} \qquad [\text{IV}]$$

$$\boldsymbol{R^2}=82.52 \qquad \boldsymbol{n}=69 \qquad \boldsymbol{s}=0.78 \qquad \boldsymbol{F}=316.44$$

Este modelo presenta un nivel de significación de 0.00 y un coeficiente de determinación lineal de 82.52%. Por lo que se consideró útil para realizar conocer cuánto demoraría el procesamiento de todas las moléculas. El resultado obtenido fue de **3 días** y **14 horas**.

Con estos resultados se demuestra que la realización del cálculo de los descriptores topológicos y topográficos de forma distribuida, optimiza el tiempo de cómputo considerablemente.

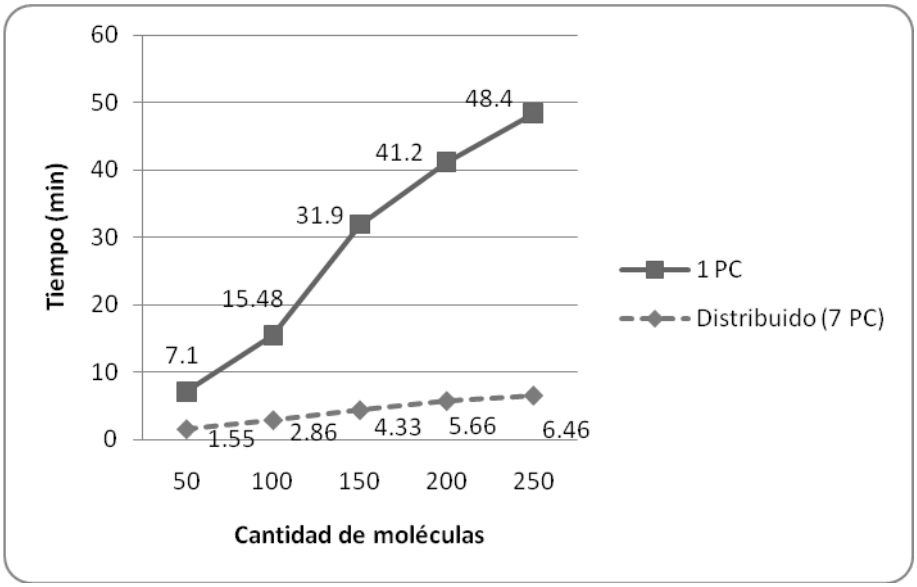


Figura 3.7 Comparación de la media de los tiempos necesitados para el cálculo

Experimento 3:

Que los cálculos desarrollados dependan de otra aplicación, contribuye a que estos no se ejecuten de la manera más rápida posible. Este experimento se realizó en un ordenador y se contabilizaron dos intervalos de tiempo.

- ✓ Desde que la aplicación obtiene el fichero ***.mop*** para la utilización del **MOPAC**, hasta se devuelve el ***.mol*** optimizado. (**T1**)

- ✓ Desde que el fichero ***.jar*** procesa el ***.mol*** hasta que devuelve los resultados en el fichero ***.dsc***. (**T2**)

El proceso descrito permitió conocer que la velocidad del resultado final depende en gran medida del **MOPAC**, lo cual es un resultado esperado. Como se observa en la ***Figura 3.8*** la gráfica descrita por la función ***T2*** tiene una menor pendiente que ***T1*** por lo que podemos afirmar que ***T1*** crece más rápido en función de la cantidad de moléculas que ***T2***.

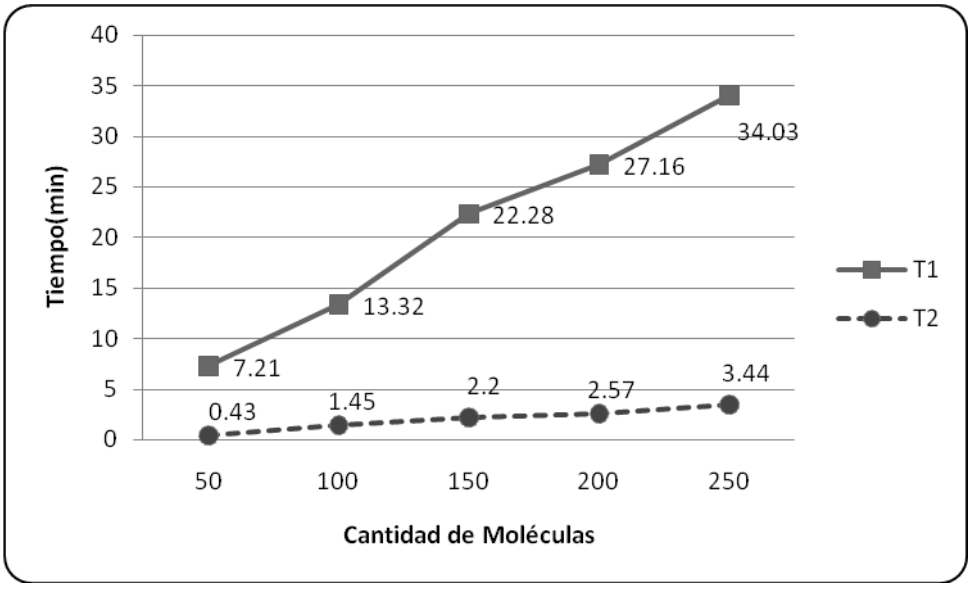


Figura 3.8 Tiempo de ejecución para T1 y T2

3.6 Conclusiones

En el capítulo se describió mediante la utilización de conceptos el flujo de acciones para desarrollar la implementación de los algoritmos. Se mostró el diagrama de clases implementadas que fueron necesarias para el desarrollo de los mismos, así como el pseudocódigo y la complejidad temporal de dos de estos. También se ejemplificó a grandes rasgos cómo se programó la aplicación utilizando la plataforma de cálculo distribuido. Por último, se describieron y discutieron los experimentos realizados para corroborar el correcto funcionamiento de los algoritmos implementados y demostrar que la realización de este trabajo disminuye en un tiempo considerable el cálculo de los descriptores.

CONCLUSIONES GENERALES

- ✓ Se realizó el análisis de los procesos necesarios para llevar a cabo el cálculo de descriptores.
- ✓ Se implementaron las funcionalidades capaces de asignar la hibridación a cada átomo, fragmentar el grafo molecular y realizar el cálculo de descriptores atómicos y moleculares.
- ✓ Se realizó la validación del correcto funcionamiento de los algoritmos implementados a partir de pruebas manuales, arrojando resultados favorables.
- ✓ Se realizó una comparación de los resultados obtenidos, con Dragon, la que resultó satisfactorio.
- ✓ Se comprobó la eficacia de los cálculos realizados por la aplicación a partir del tiempo que se demora en devolver los resultados.
- ✓ Se comprobó que realizar los cálculos de forma distribuida disminuye el tiempo de cómputo de los algoritmos implementados.
- ✓ Se demostró la factibilidad de realizar simultáneamente cálculos de índices topológicos y topográficos. Esto constituye una novedad con respecto a otras aplicaciones presentes en el mercado.

RECOMENDACIONES

- ✓ Incorporar nuevos descriptores atómicos y moleculares a la aplicación.
- ✓ Desarrollar una aplicación informática que contenga las funcionalidades implementadas y sea integrada mediante un plugin a la plataforma GRAph-TOol.

REFERENCIAS BIBLIOGRÁFICAS

1. Garduño Ramírez L. *"¿Qué sabes de los fármacos? Nuestras moléculas a prueba*". [Consultado el 23/11/2007]. Disponible en: http://www.uaem.mx/posgrado/doctos/QSD_Farmacos.pdf

2. J.C. Escalona, R. Carrasco y J. A. Padrón."*Introducción al diseño racional de fármacos*", Abril, 2008. [Consultado el 5/5/2008]. Disponible en: http://revistas.mes.edu.cu/elibro/libros/tecnologia/9789591606471.pdf/view

3. CASTAÑÓN, "*H. G. D. T. Modelización molecular de los receptores de adenosina y sus ligandos en el marco de diseño de fármacos asistido por ordenador*". [Consultado el 4/12/2007]. Disponible en: http://www.tdx.cat/TDX-0930104-091416

4. Espinosa Porragas G. "*Modelos QSPR/QSAR/QSTR basados en sistemas neuronales cognitivos*",Junio, 2002 [Consultado el 25/11/2007] Disponible en: http://biblioteca.universia.net/html_bura/ficha/params/id/20560.html

5. Costales Leyva L. y Guirola González A. "*Predicción de actividad anticancerígena de compuestos orgánicos partiendo de descriptores, utilizando programación genética*". Trabajo de Diploma de Ingeniería Informática, Universidad de las Ciencias Informáticas, 2007. [Consultado el 20/11/2007].

6. Todeschini R y Consonni V, "*Handbook of Molecular Descriptors*", 2000 [Consultado el 9/12/2007]. Disponible en: http://www.moleculardescriptors.eu

7. Estrada Roger E. "*Estudio sobre nuevos modelos grafo-teóricos para el diseño molecular en Química Orgánica*". Tesis de doctorado en Ciencia Químicas, Universidad Central de las Villas, 1996. [Consultado el 25/11/2007].

8. Carrasco Velar R. "*Nuevos descriptores atómicos y moleculares para estudios de estructura-actividad. Aplicaciones*". Tesis de doctorado en Ciencia Químicas. Centro de Química Farmacéutica, 2003 [Consultado el 23/11/2007]. Disponible en; http://revistas.mes.edu.cu/eduniv/02-Libros-por-ISBN/0601-0700/978-959-16-0646-4-Descriptores-Atomicos.pdf/at_download/file

9. J.S. Dewar Michael, Katritzky A., Karelson M "*Providers of Solutions for Computational Chemistry*", 1994. [Consultado el 19/5/2008]. Disponible en: http://www.semichem.com/codessa/default.php

10. Gracia S, Victoria E, "*Selección de nuevos antibacterianos por topología molecular*",1993 [Consultado el: 20/5/2008]. Disponible en: http://www.tesisenxarxa.net/TESIS_UV/AVAILABLE/TDX-0519105-115013//simon.pdf

11. Molconn-ZTM, 2006 "*Price List for North America and Europe*". [Consultado el: 19/5/2008]. Disponible en http://www.edusoft-lc.com/molconn/mconpric.html

12. Diseño *y Modelación de un Proyecto de Software Utilizando el lenguaje UML*. [Consultado el 9/12/2007]. Disponible en: http://www.monografias.com/trabajos28/proyecto-software/proyecto-software.shtml

13. Manual de Java [Consultado el 10/12/2007]. Disponible en: http://www.webtaller.com/manual-java/caracteristicas-java.php

14. ABIÁN., M. Á. "*El archipiélago ECLIPSE*", 2007. [Consultado el 10/12/2007]. Disponible en: http://www.javahispano.org/articles.article.action?id=81

15. Cruz Martinez K. "*Sistema de almacenamiento de la información del Sistema de Información de laboratorios",* Universidad de las Ciencias Informáticas, 2007. [Consultado el 15/4/2008].

16. Pérez Durán R. y Ortíz Tornín S. "*Lenguaje Descriptor de Estructuras Químicas*", Universidad de las Ciencias Informáticas, 2007. [Consultado el 15/4/2008].

17. Aguilera Mendoza L. "*Sistema de cómputo distribuido aplicado a la Bioinformática*". Tesis de maestría en Bioinformática, Universidad de las Ciencias Informáticas, 2008. [Consultado el 15/5/2008].

18. Heileman L. Gregory. *Estructuras de datos, algoritmos y programación orientada a objetos, 2003.*

19. Guerequeta R, Vallecillo A. "Técnicas de Diseño de Algoritmos. Capítulo 1: La complejidad de los algoritmos", 2000. [Consultado el 5/06/20008]. Disponible en http: //www.lcc.uma.es/~av/Libro/CAP1.pdf

20. Hall Lowell, Kellogg Glen, Haney David. "*Molecular Topology Analysis. Chapter 2 Methods Background: Topological Indices*", 2008 [Consultado el 4/06/2008]. Disponible en: http://www.edusoft-lc.com/molconn/manuals/400/chaptwo.html

21. Conferencia de Probabilidades y Estadísticas, "Regresión lineal multiple", Universidad de las Ciencias Informáticas. Curso 2007-2008

22. Conferencia de Probabilidades y Estadísticas, "Introducción al análisis de Correlación y Regresión", Universidad de las Ciencias Informáticas. Curso 2007-2008

BIBLIOGRAFÍA

1. Heileman L. Gregory. *Estructuras de datos, algoritmos y programación orientada a objetos,2003.*

2. Jacobson, I., Booch, G. y Rumbaugh, J. *El Proceso Unificado de Desarrollo de software,* 2000, Capítulos 1-5.

3. Carrasco Velar R. *Nuevos descriptores atómicos y moleculares para estudios de estructura-actividad. Aplicaciones*. Tesis de doctorado en Ciencia Químicas. Centro de Química Farmacéutica, 2003

4. Estrada Roger E. *Estudio sobre nuevos modelos grafo-teóricos para el diseño molecular en Química Orgánica*. Tesis de doctorado en Ciencia Químicas, Universidad Central de las Villas, 1996.

5. Johnsonbaugh R. *Matemáticas Discretas*, Volumen II, 1999, Capítulo 6: Teoría de Gráficas.

6. Consonni V, Mauri A, Pavan M. *Dragon*, 2006.

7. Englander R. *Developing Java Beans*, 1997.Capítulo 4.

8. Flanagan D. *Java en pocas palabras*, 1999. Capítulo 3.

10. Mitchell W D.*Java sin errores*, 2001. Parte II, Capítulo 6.

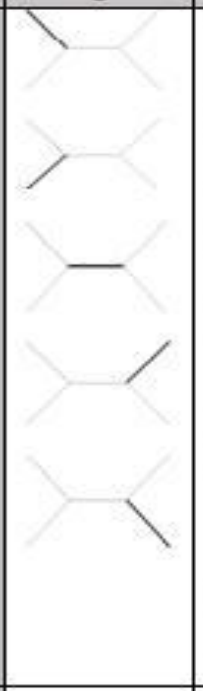
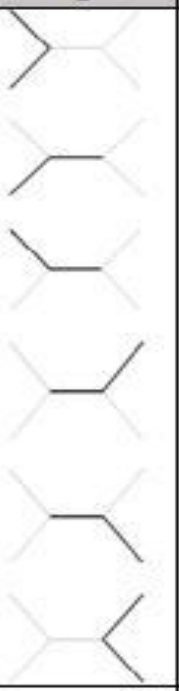
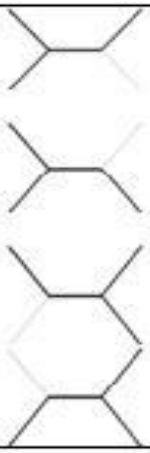
11. Zukowski J. *Programacion Java 2 J2SE 1.4*, 2003.

12. Conferencia de Programación IV, *Análisis de complejidad de algoritmos recursivos*, Universidad de las Ciencias Informáticas. Curso 2007-2008

13. Bouza Herrera C. Sistachis Vega V. “Estadística, teoría básica y ejercicios”.2004

ANEXOS

Anexo 1. Tabla de fragmentos de la molécula de Alanina

Fragmentos			Orden	
	1	2	3	4
Caminos				
Clúster				
Clúster- Caminos				

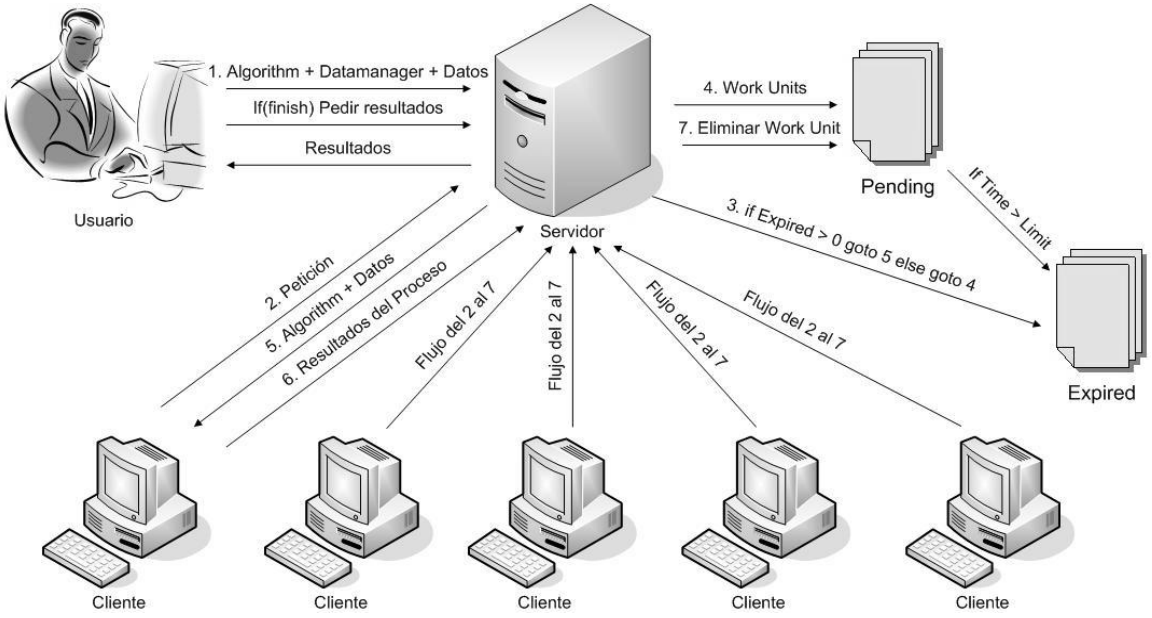
Anexo 2. Valores de refractividad atómica empleados

Tipo de Átomo	Refractividad Atómica	Tipo de Átomo	Refractividad Atómica
Csp_3	2.8158	N(Ar)	2.7662
Csp_2	3.8278	NO_2	3.5054
Csp	3.8974	Ar-N=X	3.8095
C(Ar)	3.509	F	1.0632
C=X	3.0887	Cl	5.6105
H	0.9155	Br	8.6782
-O-	1.6351	I	13.8741
=O	1.7956	Ssp_3	7.319
O=N	2.1407	Ssp_2	9.168
Nsp_3	3.01	R-SO-R	6.0762
Nsp_2,Nsp	3.2009	R-SO2-R	5.3321

Para el caso del fosforo (P), se realizó una parametrización empírica para asignarle un valor de refractividad atómica, los cuales necesitan ser probados por métodos químicos-físicos.

Sp_3 =13.50, **Sp_2 =** 15.0045, **P=X =** 19.0475

Anexo 3. Funcionamiento de una plataforma de cálculo distribuido



Anexo 4. Resultados obtenidos por la aplicación desarrollada al calcular los descriptores a la molécula Alanina

Descriptores Moleculares					
Índice de Randić					
Fragmentos	**Caminos**			**Clúster**	**Clúster/Camino**
Orden	**1**	**2**	**3**	**3**	**3/1**
Resultados	2.6427344	2.4880339	1.3333333	0.6666667	1.3333333
Índice de Valencia					
Fragmentos	**Caminos**			**Clúster**	**Clúster/Camino**
Orden	**1**	**2**	**3**	**3**	**3/1**
Resultados	1.6270897	1.1269128	0.3895276	0.2193713	0.5930026

Índice de Conectividad entre Aristas					
Fragmentos	**Caminos**			**Clúster**	**Clúster/Camino**
Orden	**1**	**2**	**3**	**3**	**3/1**
Resultados	3.3284271	2.4142136	1	0.5	0.3535534
Índice de Randić Revisitado					
Fragmentos	**Caminos**			**Clúster**	**Clúster/Camino**
Orden	**1**	**2**	**3**	**3**	**3/1**
Resultados	3.267904	3.3289454	1.9371353	0.9685677	1.9371353
Índice de la Refractividad Molecular					
Fragmentos	**Caminos**			**Clúster**	**Clúster/Camino**
Orden	**1**	**2**	**3**	**3**	**3/1**
Resultados	1.5302655	1.0286896	0.3551512	0.1944095	0.1988272
Momentos Espectrales					
Orden	**1**	**2**	**3**	**4**	**5**
Resultados	0	12	12	52	100
Orden	**6**	**7**	**8**	**9**	**10**
Resultados	300	700	1892	4692	12252
Orden	**11**	**12**	**13**	**14**	**15**
Resultados	31020	80020	204100	524172	1340572
Orden	**Total**				
Resultados	2199896				

Índice de Randić Topográfico					
Fragmentos	Caminos			Clúster	Clúster/Camino
Orden	1	2	3	3	3/1
Resultados	1.883519	1.5006813	0.6772784	0.0700729	0.0274708
Índice de Randić Revisitado Topográfico					
Fragmentos	Caminos			Clúster	Clúster/Camino
Orden	1	2	3	3	3/1
Resultados	2.6114101	2.3813497	1.2387052	0.218733	1.1136348
Índice de valencia Topográfico					
Fragmentos	Caminos			Clúster	Clúster/Camino
Orden	1	2	3	3	3/1
Resultados	1.9179298	1.5530132	0.7173865	0.1430331	0.0282194
	Índice de Conectividad entre Aristas Topográfico				
Fragmentos	Caminos			Clúster	Clúster/Camino
Orden	1	2	3	3	3/1
Resultados	2.7772814	1.6908910	0.5872318	0.2938127	0.1466097
Momentos Espectrales topográfico					
Orden	1	2	3	4	5
Resultados	0	24.4653451	34.9875993	220.8179501	603.4493779
Orden	6	7	8	9	10
Resultados	2629.358497	8733.222452	34101.7427	120806.9122	454172.7569
Orden	11	12	13	14	15
Resultados	1646427.272	6103862.967	22318070.87	82305009.35	301922034.5
Orden	Total				
Resultados	414916732.7				

Descriptores Atómicos						
		Índice del Estado Electrotopológico				
1C	2C	3O	4O	5C	6N	Total
1.8048929	0.1894894	1.6912465	1.6912465	0.1752383	1.7429125	6.5655707
		Índice del Estado Electrotopográfico				
1C	2C	3O	4O	5C	6N	Total
1.9716847	0.5905215	1.9755325	2.1025439	0.8355184	1.9418495	6.5655707

Índice del Estado Refractotopológico						
1C	**2C**	**3O**	**4O**	**5C**	**6N**	**Total**
5.9227484	3.0603291	2.8668465	2.1118465	2.4319802	5.1757493	21.5695
		Índice del Estado Refractotopográfico				
1C	**2C**	**3O**	**4O**	**4C**	**6N**	**Total**
6.0895402	2.659297	3.1511325	2.5231439	1.7717001	5.3746862	21.5695

GLOSARIO

- ✓ **Actividad Biológica:** Capacidad inherente de una sustancia, tal como un fármaco o una toxina, para alterar una o más funciones químicas o fisiológicas de una célula.

- ✓ **API**: *Applications Programming Interface -Interfaz de programación de aplicaciones*, es una serie de funciones que están disponibles para realizar programas para un cierto entorno.

- ✓ **Átomo:** Partícula más pequeña de un elemento químico que retiene las propiedades asociadas con ese elemento.

- ✓ **Bioinformática**: El uso de las matemáticas aplicadas, la estadística y la ciencia de la informática para estudiar sistemas biológicos.

- ✓ **Byte-code:** Es un código intermedio más abstracto que el código máquina. Habitualmente es tratado como un fichero binario que contiene un programa ejecutable similar a un módulo objeto, que es un fichero binario producido por el compilador cuyo contenido es el código objeto o código máquina.

- ✓ **FTP:** *File Transfer Protocol-Protocolo de Transfarencia de Fichero,* es un protocolo de transferencia de archivos entre sistemas conectados a una red TCP basado en la arquitectura cliente-servidor.

- ✓ **HTTP**: *HyperText Transfer Protoco- Protocolo de Transferencia de Hipertexto*, es el protocolo usado en cada transacción de la Web (www).

- ✓ **Ingenieria de Software**: Es una tecnología multicapa en la que se pueden identificar: los métodos (indican cómo construir técnicamente el software), el proceso (es el fundamento de la Ingeniería de Software, es la unión que mantiene juntas las capas de la tecnología) y las herramientas (soporte automático o semiautomático para el proceso y los métodos).

- ✓ **JAR:** Es un tipo de archivo que permite ejecutar aplicaciones escritas en lenguaje Java.

- ✓ **JDBC:** *Java Database Connectivity*, es un API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede.

- ✓ **Matlab:** Es la abreviatura de Matrix Laboratory (laboratorio de matrices). Es un programa de análisis numérico creado por The MathWorks en 1984. Está disponible para las plataformas Unix, Windows y Mac OS X.

- ✓ **Molécula**: Es la partícula de una sustancia que retiene toda las propiedades de la misma y está compuesta por uno o más átomos.

- ✓ **Principios Activos:** Sustancias químicas o biológicas a las que se les atribuye una actividad determinada para constituir un medicamento.

- ✓ **Química Computacional:** Es una rama de la química que utiliza computadores para ayudar a resolver problemas químicos. La química computacional es ampliamente utilizada en el diseño de nuevas drogas y materiales.

- ✓ **RMI**: ***Java Remote Method Invocation***, es un mecanismo ofrecido en Java para invocar un método remotamente.

- ✓ **SOAP:** Protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML.

- ✓ **StatGraphics:** Es un programa que está diseñado para facilitar el análisis estadístico de datos.

- ✓ **TCP/IP:** TCP (*Transfer Control Protocol - Protocolo de Control de Transmisión*) y el IP (*Internet Protocol- Protocolo de Internet*), es un conjunto de protocolos de red en los que se basa Internet.

- ✓ **Web Service**: Servicio web, es un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones.