Universidad de las Ciencias Informáticas

Facultad 6

Título: Módulo de búsqueda de fragmentos de estructuras químicas para la Plataforma GRAph TOol.

Trabajo de Diploma para optar por el título de
Ingeniero en Ciencias Informáticas

Autores

Yadir Martínez Vergara

Maikel Muñoz Roja

Tutores

Dr. Ramón Carrasco Velar

Msc. Aurelio Antelo Collado

Ing. Yania Molina Souto

La Habana, Julio de 2008

DECLARACIÓN DE AUTORÍA

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

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Yadir Martínez Vergara

_____

Firma del Autor

Maikel Muñoz Roja

_____

Firma del Autor

Ing. Yania Molina Souto

_____

Firma del Tutor

“En la Tierra hacen falta personas que trabajen más y critiquen menos, que construyan más y destruyan menos, que prometan menos y resuelvan más, que esperen recibir menos y dar más, que digan mejor ahora y no mañana”

Ernesto Che Guevara

DATOS DE CONTACTO

Tutores:
Dr. Ramón Carrasco Velar
Centro de Química Farmacéutica, Ciudad de la Habana, Cuba.
Email: ramon.carrasco@cqf.sld.cu

Msc. Aurelio Antelo Collado
Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba.
Email: aantelo@uci.cu

Ing. Yania Molina Souto
Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba.
Email: ymolinas@uci.cu

AGRADECIMIENTOS

Quisiéramos expresar nuestro más sincero y profundo agradecimiento a nuestros tutores, Ing. Yania Molina Souto, Msc. Aurelio Antelo Collado, y Dr. Ramón Carrasco Velar; quienes han prestado mucho más que su colaboración para la realización del presente trabajo, su tiempo, su dedicación, sus conocimientos, su experiencia, su paciencia, aportando además estímulo y apoyo. Sin el conjunto de todo ello no habría sido posible la realización de la presente Tesis de Grado.

A nuestro hermano y compañero José Leandro González, quien siempre nos ha mostrado su interés, estímulo y ayuda.

A todos los compañeros del laboratorio por el ánimo, el interés mostrado así como por haber compartido con nosotros muchos momentos buenos y algunos que no lo fueron tanto. Mención especial a Javier Alfonso Valdez, nuestro incansable programador, gracias por su paciencia.

A todos los que en algún momento nos ayudaron y apoyaron.

A TODOS. Gracias por hacer que sea así.

Gracias a los que dieron sin pedir nada a cambio. Gracias a los que pidieron algo, sin duda nos consideraron amigos.

Gracias a los que en alguna ocasión nos hicieron llorar, de nuevo fuimos niños.

Gracias a los que rieron con nosotros, sin duda fuimos felices.

Gracias a los que hablaron de nosotros en algún momento, bien o mal, poco o mucho, entonces nos hicieron presente.

Gracias a los que nunca nos echaron nada en cara, a pesar de nuestro carácter.

Gracias a los que alguna vez nos echaron algo en cara, nos hicieron ver nuestra imperfección.

Maikel: A la entrañable Nelvis, por su aliento, su estímulo, a veces su comprensión y paciencia, siempre su fuerza y su amor.

Yadir: A Yixander y Yoamel por su amistad y su paciencia. Por aceptarme con mis defectos y virtudes. Por ser cada día esos compañeros con quien comparto mi tiempo de ocio y laboral. Por criticar y dar consejos en los momentos que hacen falta. Por su desinterés.

Yadir: A Yeilen, por su estímulo, su interés en que todo me salga bien, por sus consejos, su comprensión y paciencia y por su amor.

A nuestros padres, todo lo que somos se lo debemos a ellos. Se esforzaron en nuestra educación y bienestar. Nos han dado lo que jamás ellos tuvieron. ¡Ojala todos los padres fueran como ustedes!

A nuestras hermanas, Yadiris y Yenia, excelentes hermanas, mejores personas, de las que hay que aprender y tomar ejemplo.

A nuestros hijos que están por venir, serán el mayor de nuestros éxitos.

DEDICATORIA

A nuestros padres.

RESUMEN

Esta investigación surge en el marco de trabajo del Proyecto "Plataforma Inteligente para la Predicción de Actividad Biológica de Compuestos Orgánicos" (GRAph TOol), que se desarrolla conjuntamente por el Centro de Química Farmacéutica y el Grupo de Bioinformática de la Universidad de las Ciencias Informáticas, para el cual se han implementado un conjunto de módulos que trabajan independientes. Se desea que uno de estos módulos tenga la funcionalidad de buscar fragmentos de estructuras químicas similares dentro de la base de datos de la Plataforma, para ello se creó el presente equipo de investigación.

La problemática de buscar similitud entre diferentes estructuras químicas no es una disyuntiva nueva. Desde el propio descubrimiento de nuevas moléculas de interés farmacéutico se ha planteado la necesidad de realizar búsquedas de información en grandes bases de datos para la determinación y estimación de estructuras, es por ello que el desarrollo de aplicaciones informáticas para extraer conocimiento de la información generada en los laboratorios se manifiesta como una de las líneas de investigación más relevantes de la Bioinformática.

Dentro de los objetivos de esta investigación se encuentra desarrollar algoritmos capaces de realizar búsquedas de estructuras químicas en 2 y 3 dimensiones dentro de la base de datos de la Plataforma GRAph TOol; por tanto el resultado de esta investigación será proveer a dicha plataforma de la capacidad de realizar búsquedas de similitud en la base de datos de compuestos orgánicos que posee.

ÍNDICE

INTRODUCCIÓN

En la actualidad la cantidad de datos disponibles y la complejidad que se han alcanzado en las interacciones biológicas han llevado a que su tratamiento manual ya no sea viable. La reducción del costo y el aumento de la potencia de las computadoras, así como el desarrollo de plataformas específicas para el análisis y la gestión de datos han ofrecido un medio para manipular esa información. La aplicación de técnicas de análisis de datos a gran escala por computadora se ha denominado Bioinformática.

La Bioinformática está ofreciendo nuevas posibilidades científicas y comerciales. La capacidad de combinar la Bioinformática con tecnologías de selección de alto rendimiento ofrece el potencial de reducir el tiempo necesario para la investigación, así como el tiempo que se necesita para convertir un descubrimiento en un producto comercial.

"*Los adelantos en la Bioinformática y la biología computacional han permitido construir equipos y programas informáticos para simular estas interacciones. Las computadoras permiten llevar a cabo rastreos virtuales de catálogos enormes con mucha más rapidez de la que se puede lograr con las técnicas bioquímicas más avanzadas. El producto es una molécula diseñada racionalmente que, una vez terminada, puede incorporarse al programa tradicional de fabricación de medicamentos.*" [1]

Paralelo a ello, los científicos se han planteado el problema de representar las moléculas en entornos visuales en dos y tres dimensiones (2D y 3D), de manera que estas representaciones ofrecen interesantes ventajas como la de hacerse una idea más concreta de la estructura en 3D de la molécula, la forma en que están dispuestos los enlaces, incluso hasta ver que determinadas geometrías no son posibles, todo ello sin contar que de esta forma se rompen las barreras idiomáticas y de formación de los diversos investigadores.

Igualmente, para estudiar mejor una molécula o macromolécula, a menudo es preferible fragmentarla en piezas menores y estudiar cada una de ellas por separado. Esto simplifica el trabajo a costa de perder quizás alguna información sobre las interacciones entre las partes de la molécula, sin embargo, a pesar de ello esta técnica cada día alcanza mayor popularidad.

Actualmente a nivel mundial se pueden encontrar referencias de aplicaciones que desarrollan esta técnica, tal es el caso por ejemplo del Spartan y el Trident. Este último es una versión ampliada del primero orientada a la química médica. Entre sus capacidades sobresalen que permite construir, importar, representar y consultar moléculas; realizar cálculos; análisis conformacional, análisis de similitud y adicionalmente cuenta con una gran base de datos incluida en el software con cientos de miles de estructuras y propiedades moleculares, compuestos, fármacos comunes, ligandos y estructuras cristalinas.

Sin embargo estos sistemas son privativos, lo que implica que no se puede disponer de su código fuente y mucho menos de los algoritmos que utilizaron para realizar cada una de las tareas. Igualmente sus precios son muy elevados, llegando alcanzar los 1500 dólares en el caso del Spartan sólo la licencia para una estación de trabajo.

Por otra parte estos sistemas no son compatibles con las características de la Plataforma GRAph TOol, por lo que no podrían integrarse a ella. Estos programas cuentan con bases de datos de compuestos propias y la Plataforma tiene su base de datos de estructuras con actividad biológica reportada, por lo que poca utilidad brindaría uno de estos programas si no pueden obtener datos a partir de los compuestos de la Plataforma.

En Cuba, donde se ha desarrollado un elevado potencial humano en el campo de las ciencias, se inicia la explotación de esta novedosa disciplina. Los centros de investigación más relevantes del país están creando condiciones para fomentar la computación paralela y realizar tareas de gran complejidad.

Estas circunstancias condujeron al nacimiento del proyecto conjunto entre el Centro de Química Farmacéutica y la Universidad de las Ciencias Informáticas (CQF-UCI) “Plataforma Inteligente para la Predicción de Actividad Biológica de Compuestos Orgánicos”, concebido como aplicación modular multiplataforma, con capacidades para realizar tamizaje virtual de moléculas orgánicas empleando técnicas de inteligencia artificial para la predicción de actividad biológica, con modelos desarrollados a partir de una base de datos de compuestos con actividad reportada.

Teniendo en cuenta lo expuesto previamente sobre el estudio de las moléculas fragmentadas y las ventajas que esto reporta a los científicos, así como la necesidad de que los investigadores cuenten con la posibilidad de realizar análisis de similitud y búsqueda de fragmentos de moléculas en

una base de datos, se plantea como **problema científico** de esta investigación: ¿Cómo encontrar fragmentos análogos en diferentes estructuras químicas dentro de la base de datos de la Plataforma GRAph TOol?

Los análisis de similitud y la búsqueda de fragmentos de proteínas en grandes bases de datos permiten hallar moléculas similares y cuantificar la similitud molecular en base a estructura o función química. La similitud puede cuantificarse usando diversos confórmeros y es posible también cuantificar la similitud de las moléculas a farmacóforos. Es por ello que en el presente trabajo se plantea la necesidad de buscar fragmentos de estructuras químicas análogas en la base de datos de la Plataforma.

Por lo que el **objeto de estudio** son los algoritmos de búsqueda de fragmentos de estructuras químicas teniendo como **campo de acción** la búsqueda de fragmentos de estructuras químicas para la Plataforma GRAph TOol.

El presente trabajo tiene como **objetivo general** desarrollar algoritmos capaces de identificar fragmentos análogos en estructuras químicas en 2 y 3 dimensiones almacenadas en la base de datos de la plataforma GRAph TOol.

Para cumplir el objetivo general del trabajo se trazaron los siguientes **objetivos específicos**:

1. Realizar un estudio de la bibliografía y las técnicas existentes para la búsqueda de estructuras químicas.
2. Proponer los algoritmos para la búsqueda de fragmentos de estructuras químicas en 2 y 3 dimensiones.
3. Desarrollar un software para comprobar los algoritmos de búsqueda.
4. Analizar los resultados de las pruebas.

Para alcanzar dichos objetivos se planteó desarrollar las siguientes **tareas investigativas**:

1. Revisión del estado del arte acerca de sistemas de búsquedas de fragmentos desarrollados en el mundo.
2. Revisión del estado del arte acerca de algoritmos de búsquedas de fragmentos desarrollados por otros investigadores.

3. Análisis del estado del arte que permita dejar definido la posición del investigador respecto al uso de los algoritmos encontrados.
4. Diseño de los algoritmos para la búsqueda de fragmentos de estructuras químicas en 2 y 3 dimensiones.
5. Implementación del software de prueba para comprobar los algoritmos.
6. Implementación de los algoritmos de búsqueda de fragmentos análogos en estructuras químicas.
7. Realización de pruebas para comprobar los algoritmos de búsqueda.
8. Análisis de los resultados obtenidos con las pruebas del software.

Este documento está compuesto por un resumen, introducción, 3 capítulos que constituyen el cuerpo fundamental del documento, conclusiones generales, bibliografía y referencias bibliográficas. Los capítulos son:

Capítulo 1: **Fundamentación Teórica**. En este capítulo se presentará un resumen de la investigación realizada sobre la búsqueda de fragmentos de estructuras químicas. Se aborda el surgimiento de estas técnicas en el mundo. Se señalan las tendencias actuales y el estado del arte a tener en cuenta. Igualmente se describen las aplicaciones informáticas más relevantes desarrolladas hasta el momento sobre este tema.

Capítulo 2: **Materiales y Métodos**. En este capítulo se realiza la descripción de la investigación como base para la continuación del estudio por otros investigadores. Se realiza la selección y justificación de los materiales y métodos.

Capítulo 3: **Resultados y Discusión**. En este capítulo se explica brevemente los aspectos esenciales que se tuvieron en cuenta para la construcción del software de prueba, así como la implementación de los algoritmos y el aprovechamiento de la capacidad de cómputo que brinda una plataforma de cálculo distribuido. Además se ilustran y analizan los resultados de la investigación y las pruebas realizadas. Se realiza una evaluación de las implicaciones, trascendencia y beneficios de estos resultados y las posibles aplicaciones del sistema.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

Los avances en la ciencia así como el vertiginoso desarrollo de las tecnologías de la información y las comunicaciones (TIC) están incrementando el papel de la información científica en las organizaciones. La necesidad de obtener la información necesaria en el momento adecuado otorga el derecho a evaluar y estudiar el valor de la información.

A raíz de estos cambios diversos investigadores han postulado una nueva revolución industrial donde se producirá un mayor énfasis en el valor de la información y el conocimiento que en las mercancías.

1.2 La Bioinformática y su relevancia en el desarrollo de nuevos fármacos.

Según la definición del Centro Nacional para la Información Biotecnológica "National Center for Biotechnology Information" (NCBI por sus siglas en Inglés, 2001): *"Bioinformática es un campo de la ciencia en el cual confluyen varias disciplinas tales como: biología, computación y tecnología de la información…"* [2] Una de las principales aplicaciones de la Bioinformática es la simulación, la minería de datos y el análisis de los datos obtenidos en el estudio de moléculas relevantes para la vida. La Bioinformática además se nutre de dos grandes áreas del conocimiento: las ciencias biológicas y las ciencias de la computación.

El auge que experimenta la Bioinformática es visible con el surgimiento de compañías orientadas a prestar servicios en el manejo de la información biológica. Los datos en sí mismos no son comerciables, pero la información implícita en ellos si lo es. A nivel de la Industria Farmacéutica, -donde los procesos de investigación y desarrollo además de largos son sumamente costosos-, la Bioinformática está llamada a jugar un papel preponderante en el descubrimiento de nuevos medicamentos.

El éxito en las tareas de investigación y desarrollo en el marco de la industria farmacéutica está dado en cómo se entienda el sistema genético y en cómo se sea capaz de extraer de dicho sistema la información necesaria para inhibir ciertos procesos. Realmente una computadora no puede reemplazar

un laboratorio y es por esto que los aportes tradicionales químicos y farmacológicos continuarán siendo importantes, sin embargo, es igualmente cierto que el uso de recursos informáticos minimiza tiempo en esta clase de procesos de investigación y desarrollo haciendo más atractiva la relación Tiempo-Costo-Beneficio.

"La Bioinformática tiene amplio uso en los procesos de investigación y desarrollo de nuevos medicamentos, y esta tendencia se incrementará en la medida en que las computadoras, los software y las facilidades para su interconexión continúen su vertiginoso desarrollo. Hasta el momento pueden identificarse siete áreas vinculadas con la investigación y desarrollo de medicamentos, donde la Bioinformática juega un rol importante. Estas son: administración de la información de las sustancias y moléculas, administración de los datos de las reacciones, búsquedas de subestructuras por analogías, planificación de la síntesis químicas, elucidación de estructuras desconocidas, relación entre estructuras cuantitativas y actividad de moléculas, y modelación de estructuras de tres dimensiones y sus interacciones." [3]

Específicamente en el proyecto GRAph TOol se trabaja en varias de estas direcciones, siendo particularmente el tema de esta investigación, dentro de los mencionados, la búsqueda de subestructuras por analogía o análisis de similitud.

1.3 La búsqueda de similitud molecular

"Las estructuras químicas comenzaron a ser procesadas y almacenadas por los diversos servicios de información pública especializadas o relacionados con la industria química. Igualmente grandes empresas farmacéuticas diseñaron sistemas de procesamiento y almacenamiento de esta información vital para su trabajo de investigación y desarrollo. Ambos sistemas encontraron amplia aceptación entre los usuarios especializados que en muchas ocasiones prefieren hacer búsquedas mediante la estructura en sí, y no mediante los nombres de un compuesto específico que puede procesarse con diferentes nombres en diferentes bases de datos. Iniciadores de este tipo de bases de datos y servicios fueron The Chemical Abstracts Service (CAS), Beilstein Online, y Derwent Publications." [3]

La búsqueda de similitud es aquella técnica de recuperación mediante la cual a partir de una estructura química definida por un especialista se identifican aquellas moléculas en una base de datos

que son más similares a la molécula usando medidas cuantitativas de similitud intermolecular. Esta técnica de recuperación es aplicable a las bases de datos 2D y 3D; y dos clases de medidas de similitud de estructuras químicas han sido desarrolladas, las medidas globales y locales. Las medidas de clase global son aquellas que evalúan un valor numérico para la completa similitud entre dos moléculas. Las medidas del tipo local son aquellas medidas que proveen información física como resultado de la alineación de una molécula con otra; por ejemplo estos sistemas producen un mapeo de las características de la molécula con aquellas características estructurales de las moléculas existentes en la base de datos, de forma que este proceso constituya una superposición de una molécula sobre otra. En el caso particular de esta investigación esta superposición se basa fundamentalmente en la similitud topológica o topográfica, según el caso, que logren alcanzar el fragmento a buscar y las moléculas que se encuentran dentro de la Base de Datos.

Es importante señalar que los sistemas de información en su mayoría han trabajado en aquellas búsquedas que abarcan un gran número de registros o estructuras químicas en las bases de datos, estando estas enmarcadas en las de clase Global. Las conocidas por locales conllevan procesamientos mas lentos y no pueden incluirse gran número de registros a procesar de acuerdo con el estado actual del hardware y software disponible; generalmente son sistemas desarrollados para trabajos específicos de similitud entre un grupo muy limitados de moléculas.

Por otro lado el cálculo de similitud entre moléculas debe ser efectivo, su aplicación debe dar resultados útiles al usuario; y debe ser eficiente, que se traduce en utilizar los requerimientos de computación necesarios para lograr búsquedas de similitud en bases de datos de tamaño considerable en tiempos razonables.

Tres pasos son necesarios para realizar una búsqueda de similitud topológica; el usuario da las especificaciones de la molécula o fragmento de molécula a ser identificada; el sistema de búsqueda compara la estructura topológica de cada una de las estructuras existentes en la base de datos; y finalmente se obtiene una lista en orden descendente, donde las estructuras evaluadas como las más similares a la estructura de entrada encabezan la lista y van descendiendo en la medida en que su similitud disminuye.

Los trabajos de búsquedas de similitud crean las bases para un mejor desarrollo de los proyectos de investigación a la hora de evaluar nuevas variantes de moléculas para ser utilizadas en sustitución

de otras; creando todo un campo de gestión de información dentro de las estructuras químicas con directa implicación en los proyectos de investigación y desarrollo.

"*En los próximos diez años las TIC mediante los sistemas de búsquedas de estructuras químicas, las bases de datos de secuencias, el proyecto GENOMA, las bases de datos de reacciones químicas, y todos los sistemas actuales y en desarrollo para acelerar el proceso de descubrimiento de nuevos medicamentos, jugará un papel determinante en la culminación exitosa y óptima de los proyectos de investigación y desarrollo de nuevos fármacos. La industria biotecnológica orientada al proceso de descubrimiento de nuevos medicamentos no puede estar divorciada de esta tecnología y su directa aplicación en los proyectos de investigación."* [3]

1.4 Sistemas existentes

1.4.1 Trident

"Trident es una herramienta de modelado molecular aplicado al área de química médica. Dicho software incluye herramientas de modelado necesarias para el proceso de desarrollo de nuevos fármacos: construir y representar moléculas, calcular y analizar las estructuras, energías y propiedades moleculares, determinar las formas moleculares y cuantificar las similitudes entre estructuras." [4]

Trident se diseñó para ofrecer a los investigadores en el área de química médica herramientas de modelado que les permita realizar búsquedas conformacionales, calcular estructuras y propiedades y cuantificar la similitud molecular en 3D.

Trident además determina diversas propiedades tales como: estructura y propiedad, forma molecular, análisis de similitud, propiedades QSAR y propiedades gráficas. Trident sólo está disponible para los sistemas operativos (SO) Windows 2000 y XP.

1.4.2 Spartan

La versión más reciente de este programa data del año 2006. Este dispone de una interfaz gráfica de usuario, de uso sencillo, que permite construir y manipular moléculas, lanzar cálculos de mecánica molecular y química cuántica y visualizar resultados. Este software es además compatible

con los formatos SYBYL MOL, MOL2, PBD, MACROMODEL, MDL, TGF, SMILES, ficheros CIF y XYZ.

Entre las tareas que puede realizar este programa encontramos: Cálculo de energía de reacción y de activación, determinación de geometría de equilibrio, determinación de geometría del estado de transición, cálculo de las frecuencias vibracionales y intensidades, identificación de la distribución energéticas de los confórmeros, alineamiento de moléculas, generación de secuencias de reacción, identificación de tautomerías, estudio de las coordenadas geométricas y energías, termoquímica y análisis de similitud.

Se puede añadir además que cuenta con una base de datos con 140 mil moléculas las cueles se pueden filtrar por diferentes criterios.

Spartan '06, la versión más reciente está disponible para los sistemas operativos Windows, Macintosh, Linux y Unix.

En la introducción de este documento ya se hizo referencia detallada de los inconvenientes de estos sistemas y a la dificultad que implican estos para la incorporación de estos programas a la Plataforma GRAph TOol.

1.5 Método de búsqueda tentativo – Tablas Hash

Una metodología de búsqueda radicalmente diferente a sus anteriores llamada Tablas Hash consiste en proceder, no por comparaciones entre valores claves, sino encontrando alguna función $h(k)$que dé directamente la localización de la clave k en la tabla de valores.

La primera pregunta que viene a la mente es si es fácil encontrar tales funciones $h(k)$, y aunque suene algo pesimista la respuesta es inicialmente NO.

“Las tablas Hash se suelen implementar sobre arreglos de una dimensión, aunque se pueden hacer implementaciones multidimensionales basadas en varias claves. Las tablas Hash almacenan la información en posiciones aleatorias, así que el acceso ordenado a su contenido es bastante lento.

Para la implementación de una de estas tablas se necesita una estructura de acceso directo (generalmente un arreglo), una estructura de datos con una clave, una función resumen $h(k)$ cuyo dominio sea el espacio de claves y su imagen los números naturales.

Para almacenar un elemento en la tabla Hash se ha de convertir su clave a un número. Esto se consigue aplicando la función $h(k)$ a la clave del elemento. El resultado de la función resumen ha de mapearse en el arreglo que se emplea como soporte. Tras este paso se obtiene un índice válido para la tabla.

El elemento se almacena en la posición de la tabla obtenida en el paso anterior. Si en la posición de la tabla ya había otro elemento, se ha producido una colisión. Este problema se puede solucionar asociando una lista a cada posición de la tabla, aplicando otra función o buscando el siguiente elemento libre. Estas posibilidades han de considerarse a la hora de recuperar los datos.

Para recuperar los datos, es necesario únicamente conocer la clave del elemento, a la cual se le aplica la función resumen $h(k)$. El valor obtenido se mapea al espacio de direcciones de la tabla. Si el elemento existente en la posición indicada en el paso anterior tiene la misma clave que la empleada en la búsqueda, entonces es el deseado. Si la clave es distinta, se ha de buscar el elemento según la técnica empleada para resolver el problema de las colisiones al almacenar el elemento.

En una función Hash ideal, el cambio de un simple bit en la llave debería cambiar la mitad de los bits del Hash, y este cambio debería ser independiente de los cambios provocados por otros bits de la llave.

Si dos llaves generan un Hash apuntando al mismo índice, los registros correspondientes no pueden ser almacenados en la misma posición. En estos casos, cuando una casilla ya está ocupada, se debe encontrar otra ubicación donde almacenar el nuevo registro, y hacerlo de tal manera que podamos encontrarlo cuando se requiera" [5]

Las tablas Hash tienen como principal ventaja que el acceso a los datos es relativamente rápido siempre y cuando se cumpla que la función resumen distribuya uniformemente las claves, disminuyendo así el número de colisiones, sin embargo cuentan con grandes inconvenientes como el alto costo cuando se trata de ampliar el espacio de la tabla, dificultad para recorrer un número elevado

de elementos, desaprovechamiento de memoria al tener más espacio reservado que el que realmente se necesita; a ello se suma las grandes dificultades a la hora de implementar un algoritmo basado en Hash y la determinación de las funciones $h(k)$.

1.6 Conclusiones del capítulo

A partir del análisis de los sistemas existentes en el mundo que realizan tareas de búsqueda de similitud molecular se arribó a la conclusión que ninguno cumple con las necesidades de la Plataforma GRAph TOol. Esto impulsa a los investigadores a diseñar sus propios algoritmos e implementar su propia herramienta de manera que puedan servir de utilidad dentro de la Plataforma.

Con el desarrollo de un algoritmo y posteriormente de una aplicación informática, el trabajo con la Plataforma GRAph TOol será mucho más sencillo. Esto permitirá hacer más breve el proceso de búsqueda de nuevas entidades químicas con posible actividad biológica, las cuales servirán para el desarrollo de nuevos fármacos y disminuirá los costos de investigación y desarrollo en el complicado proceso de diseño de medicamentos.

CAPÍTULO 2: MATERIALES Y MÉTODOS

2.1 Introducción

En el capítulo anterior se presentó un resumen del estado del arte del tema de investigación así como de algunos elementos esenciales para comprender en que consiste el análisis de similitud molecular, se señalaron tendencias actuales, se ilustraron ejemplos y se arribó a conclusiones parciales.

En este capítulo se presentarán los materiales utilizados para la construcción del software de prueba así como los métodos que se siguieron a la hora de utilizar dichos materiales.

2.2 Materiales

2.2.1 Java - Lenguaje de programación

Java es un lenguaje de programación con el que se puede realizar cualquier tipo de programa. En la actualidad está muy extendido y cada vez cobra más importancia tanto en el ámbito de Internet como en la informática en general. Es desarrollado por la compañía Sun Microsystems y siempre enfocado a cubrir las necesidades tecnológicas más punteras.

Una de las principales características por las que Java se ha hecho muy famoso es que es un lenguaje independiente de la plataforma. Eso quiere decir que si se realiza un programa en Java podrá funcionar en cualquier ordenador del mercado. Es una ventaja significativa para los desarrolladores de software, pues antes tenían que hacer un programa para cada sistema operativo. Esto se consigue porque se ha creado una Máquina Virtual de Java para cada sistema que hace de puente entre el sistema operativo y el programa de Java y posibilita que este último se entienda perfectamente.

La independencia de plataforma es una de las razones por las que Java es interesante para Internet, ya que muchas personas deben tener acceso con ordenadores distintos. Java además está diseñado para ser lo suficientemente simple para que los programadores puedan lograr fluidez con el lenguaje. Ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de lenguajes como C/C++.

Una de las características más importante de Java es que posee una arquitectura neutral, es decir el compilador 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.

Sin duda alguna la frescura, la fortaleza, la robustez y por supuesto la portabilidad de Java lo hacen un lenguaje de programación excelente para los propósitos que se han propuesto en la realización de esta aplicación.

2.2.2 Entornos de desarrollo

Cuando se utiliza el lenguaje de programación Java para el desarrollo de aplicaciones se encuentran diversos entornos de desarrollo integrado (Integrated Development Environment-IDE) tales como NetBeans, JBuilder, Eclipse entre otros, cada uno de ellos posee sus ventajas y desventajas, contribuyendo siempre al enriquecimiento del propio lenguaje.

NetBeans

El NetBeans es una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. Está escrito en Java pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender el IDE NetBeans, es un producto libre y gratuito sin restricciones de uso. Posee todas las potencialidades de los entornos de alto nivel. Sin embargo este IDE tiene una notable deficiencia que es la enorme memoria que ocupa cuando se está ejecutando, lo que produce que las máquinas respondan lentamente a las instrucciones de los usuarios, esto hace que su utilización en computadoras con pocos recursos sea sumamente engorrosa.

Por otra parte teniendo en cuenta que NetBeans es un entorno de desarrollo excelente para aplicaciones visuales, el cual facilita mucho el trabajo a los desarrolladores y que basa sus mayores bondades hacia esta dirección, queda finalmente descartado, ya que la aplicación que se implementará depende mayormente de codificación de algoritmos y no de interfaces visuales, y lo que se requiere son respuestas rápidas y buen completamiento de código para acelerar la codificación.

Eclipse

Eclipse se ha convertido en una poderosa herramienta que integra diferentes aplicaciones para convertirse en un competitivo entorno de desarrollo integrado. Aunque utiliza java como lenguaje de programación, permite a través de la incorporación de plug-ins, programar para otros lenguajes más; soporta además la programación orientada a objetos, la depuración y compilación de código. Dentro de él, la implementación de aplicaciones resulta mucho más sencilla que entornos para Java anteriores.

Entre sus beneficios se pueden contar que se trata de una herramienta de código abierto, soporta herramientas que manipulan diferentes tipos de lenguajes, como por ejemplo Java, C, C++; corre en una gran cantidad de sistemas operativos incluyendo Windows y Linux, provee a los desarrolladores de herramientas que facilitan la creación de plug-ins y otros componentes de software.

Su capacidad de ser soportado por múltiples arquitecturas, ser multiplataforma, poseer capacidad de control de versiones con subversion, resaltado de sintaxis, autocompletamiento, posee asistentes para la creación, exportación e importación de proyectos; generación de plantillas de código, permite la integración con la herramienta CASE Visual Paradigm, y todas las demás ventajas y aplicaciones lo hacen el entorno de desarrollo ideal para implementar la solución propuesta.

2.2.3 UML - Lenguaje de modelado

El Lenguaje de Modelado Unificado (UML - Unified Modeling Language) *"es un lenguaje para visualizar, modelar, construir y documentar los artefactos de un sistema"* [6]

UML no es un lenguaje de programación sino un lenguaje de propósito general para el modelado orientado a objetos y también puede considerarse como un lenguaje de modelado visual que permite una abstracción del sistema y sus componentes.

El lenguaje UML tiene una notación gráfica muy expresiva que permite representar en mayor o menor medida todas las fases de un proyecto informático: desde el análisis con los casos de uso, el diseño con los diagramas de clases, objetos, etc., hasta la implementación y configuración con los diagramas de despliegue. Una de sus mayores ventajas radica en que es independiente del lenguaje de implementación. Además por ser un método formal de modelado aporta las ventajas de mayor rigor en la especificación, verificación y validación del modelo realizado.

2.2.4 Herramientas CASE

Rational Rose

El Rational Rose es una de las más poderosas herramientas de modelado visual para el análisis y diseño de sistemas basados en objetos. Se utiliza para modelar un sistema antes de proceder a construirlo y entre sus más interesantes ventajas está que permite cubrir todo el ciclo de vida de un proyecto a través de modelos y diagramas.

Esta herramienta fue desarrollada por los creadores de UML, utilizando la notación estándar en la arquitectura de software. Además integra todos los elementos que propone la metodología RUP para cubrir el ciclo de vida de un proyecto, sin embargo presenta las serias limitantes de no poseer soporte multiplataforma y no ser un software libre.

Visual Paradigm

Esta herramienta al igual que Rational Rose está diseñada para una gran gama de usuarios que van desde ingenieros de software, analistas de sistemas y de negocios hasta arquitectos de proyectos.

Es completamente compatible con UML y entre sus ventajas más relevantes se encuentran que permite realizar ingeniería tanto directa como inversa, es una herramienta colaborativa, es decir, soporta múltiples usuarios trabajando sobre el mismo proyecto; genera documentación del proyecto automáticamente en varios formatos tales como web o pdf, y por si fuera poco permite control de versiones. Igualmente se puede destacar su robustez, usabilidad y portabilidad.

En definitiva, Visual Paradigm se ha convertido en una herramienta CASE a tener en cuenta a la hora de pensar un proyecto importante por esta razón se ha seleccionado como herramienta CASE para la realización del sistema.

2.2.5 MySQL – Gestor de base de datos

Existen múltiples gestores de bases de datos y el propósito es que en el futuro se desarrolle una aplicación con la capacidad de interactuar con cualquiera de ellos. Sin embargo la base de datos de la

Plataforma GRAph TOol esta montada sobre el gestor de base de datos MySql, por tanto constituyó una necesidad adaptarse a esta circunstancia.

MySql es un sistema de gestión de base de datos relacional, multihilo y multiusuario. Es propiedad y está patrocinado por una empresa privada, que posee el copyright de la mayor parte del código, esto lo hace en la actualidad estar catalogado como un software privativo y que muchos de sus clientes estén migrando hacia otros gestores de bases de datos.

Las bases de datos en MySql son muy veloces en la lectura cuando utiliza el motor no transaccional MySAM, pero pueden provocar problemas de integridad en entornos de alta concurrencia en la modificación.

MySQL 5 recientemente añadió nuevas características que lo hacen aún más atractivo. Las más interesantes son los procedimientos almacenados, los desencadenadores (triggers) y las vistas.

2.3 Métodos

2.3.1 Programación orientada a objetos

La programación orientada a objetos (POO) es una metodología de programación avanzada y bastante extendida, en la que los sistemas se modelan creando clases, que son un conjunto de datos y funcionalidades. Las clases son definiciones, a partir de las que se crean objetos. Los objetos son ejemplares de una clase determinada y como tal, disponen de los datos y funcionalidades definidos en la clase." [7]

Entre las ventajas de la POO sobresalen la abstracción que alcanzan los datos, la disciplina y organización a la hora de programar, reutilización del código, mantenimiento y extensión de las aplicaciones, facilidad de crear aplicaciones visuales, construcción de prototipos, además de nuevas funcionalidades que le dan mayor potencia que se expresan a través de sus tres pilares: la herencia, el polimorfismo y la genericidad.

Igualmente la POO tiene la gran ventaja de recrear en su gran mayoría los problemas a través de situaciones de la vida real, lo cual permite crear sistemas cada vez más complejos.

Como ya se ha mencionado anteriormente el lenguaje de programación que se propuso para esta aplicación es Java, el cual tuvo su origen en Sun Microsystems. Esta firma es quien ha desarrollado el lenguaje Java, en un intento de resolver simultáneamente todos los problemas que se le plantean a los desarrolladores de software por la proliferación de arquitecturas incompatibles, tanto entre las diferentes máquinas como entre los diversos sistemas operativos y sistemas de ventanas que funcionaban sobre una misma máquina, añadiendo la dificultad de crear aplicaciones distribuidas en una red como Internet.

2.3.2 Modelo matemático utilizado

En la sección 1.5 del capítulo 1 se analizaron las Tablas Hash como método tentativo para la implementación de los algoritmos de búsqueda. Sin embargo los algoritmos basados en Hash como ya se explicó presentan algunas dificultades tales como el alto costo para amplios espacios de búsqueda así como el desaprovechamiento de memoria al tener más espacio reservado que el que realmente se necesita. Otra dificultad de los algoritmos basados en Hash es que no están diseñados para descartar posibilidades. Si un algoritmo Hash no encuentra la solución en una combinación volverá sobre el mismo camino una y otra vez hasta que acabe de evaluar todas las combinaciones posibles por lo que en el peor de los casos (que muchas veces se verifica) evaluará un número n de combinaciones innecesarias que otros algoritmos pudieran haber descartado en combinaciones iniciales.

Representación del conocimiento – Coordenadas Internas

Las coordenadas internas o también conocidas como matriz-z constituyen una sencilla forma de representar una molécula. Estas definen las coordenadas de los átomos en función de otros átomos. El origen de coordenadas viene dado por el primer átomo. Después, se define el primer parámetro geométrico como la distancia al segundo átomo. Un ángulo de enlace se define como el ángulo entre 3 átomos.

Los ángulos diedros se empezarán a definir a partir del cuarto átomo con respecto a los otros tres. El ejemplo del ácido fórmico ilustra estas características:

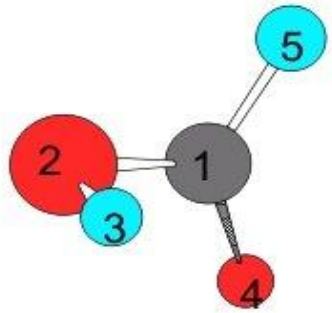


Figura 2. 1: Molécula de ácido fórmico.

Orden	Tipo	Ref	Distancia	Ref	Angulo	Ref	Diedro
1	C						
2	O	1	1,336				
3	H	2	0,972	1	106,10		
4	O	1	1,206	2	124,68	3	90,00
5	H	1	1,113	2	110,46	3	-90,00

Tabla 2. 1: Coordenadas internas de una molécula de ácido fórmico.

Si se presta atención, para definir la geometría en coordenadas internas se utiliza para el primer átomo ningún parámetro, para el segundo una variable distancia, para el tercero dos variables, distancia y ángulo de enlace; y para el cuarto y siguientes distancia, ángulo de enlace y ángulo diedro. Estas coordenadas internas definen un número máximo de variables como el número 3n-6 grados de libertad donde n es el número de átomos de la molécula.

2.4 Conclusiones de capítulo

En este capítulo se realizó un análisis y se seleccionaron las herramientas y metodologías para llevar a cabo la implementación de los algoritmos y el software de prueba que lo acompañará. Se identificó un modelo matemático sencillo, que constituye la base para la creación de los algoritmos que permitirán alcanzar los objetivos propuestos.

CAPÍTULO 3: RESULTADOS Y DISCUSIÓN

3.1 Introducción

En el presente capítulo se abordarán temas de especial relevancia para la comprensión del sistema implementado, se explicarán los algoritmos fundamentales que fueron implementados, el diseño de clases que se utilizó, se expondrá una vista global del sistema así como su integración a la plataforma de cálculo distribuido y a la plataforma GRAph TOol, además se realizará la discusión de los resultados fundamentales obtenidos a través de los distintos experimentos realizados con el software.

3.2 Modelo conceptual

Un modelo conceptual es la herramienta más importante del análisis orientado a objetos, es por ello que se ha querido incluir aquí. En él se explican los conceptos más significativos en el dominio del problema, identificando los atributos y las asociaciones. Un modelo conceptual representa cosas del mundo real, no componentes del software, de ahí que sea de especial importancia para clientes y desarrolladores. En UML suele representarse mediante un grupo de diagramas de estructura estática donde no se define ninguna operación. En estos diagramas se muestran conceptos (objetos), asociaciones entre conceptos (relaciones) y atributos de conceptos (atributos). La siguiente figura muestra el modelo conceptual en el marco de una herramienta que busca fragmentos de moléculas en una base de datos de estructuras químicas.

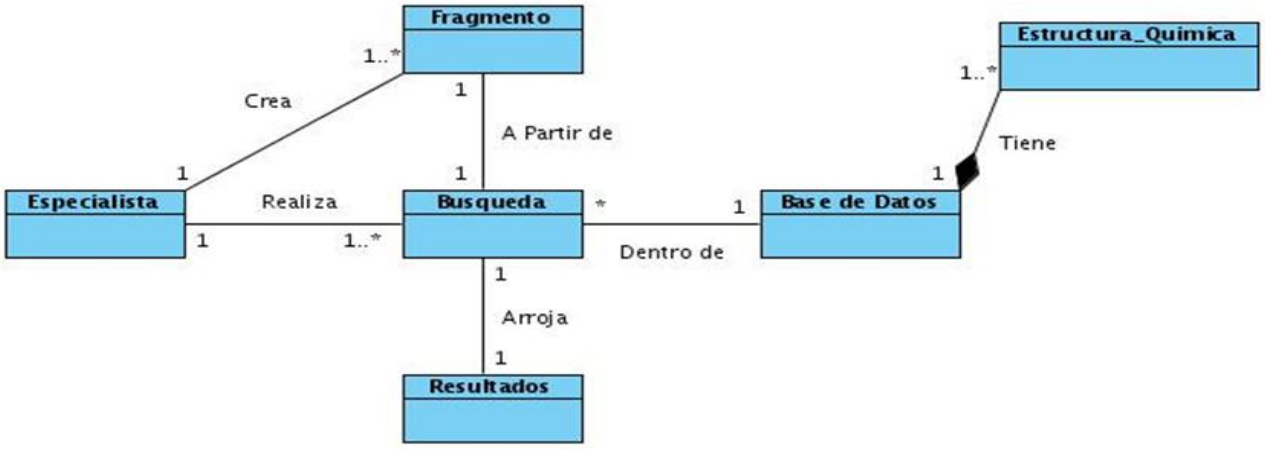


Figura 3. 1: Modelo conceptual

3.3 Algoritmos de búsqueda

3.3.1 Buscando en dos dimensiones

Como se mostró en la Sección 2.3.2 del Capítulo 2, una sencilla forma de representar una molécula es mediante sus coordenadas internas. Utilizando este modelo para la representación de las moléculas se pueden definir y calcular parámetros tales como:

1. Coordenadas de los átomos en el plano y el espacio ([x, y] en el plano y [x,y,z] en el espacio).
2. Tipos de enlaces que existen entre cada par de átomos.
3. Ángulo entre 3 átomos en el plano y ángulo entre 4 átomos en el espacio (ángulo diedro).
4. Distancia entre dos átomos.

Contando con estos datos iniciales de un fragmento de molécula, el problema consiste en analizar su similitud con otras moléculas.

Para realizar una búsqueda en 2D se definen los siguientes pasos:

1. Comprobar que los átomos y enlaces de entrada se encuentren contenidos dentro de la molécula candidata.
2. Comparar las pendientes entre cada par de átomos de entrada y cada par de la molécula candidata. Para ello se utiliza la conocida fórmula para obtener la pendiente entre dos puntos del plano.
3. Comparar los ángulos formados por cada 3 átomos adyacentes de entrada y sus correspondientes en la molécula candidata. Estos ángulos se calculan utilizando la Ley de los Cosenos.

"*En todo triángulo el cuadrado de un lado es igual a la suma de los cuadrados de los otros dos lados menos el doble producto de ellos, por el coseno del ángulo que forman*" [8]

$$c^2 = a^2 + b^2 - 2ab\cos\theta$$

$$b^2 = a^2 + c^2 - 2ac\cos\beta$$

$$a^2 = b^2 + c^2 - 2bc\cos\alpha$$

En resumen, la búsqueda en 2D realiza las siguientes acciones:

1. Se crea una lista con los enlaces y se ordena por el índice del primer átomo del enlace.
2. Se verifica que la longitud de la lista de enlaces del fragmento sea menor o igual que la de la lista de enlaces de la molécula, de lo contrario se descarta dicha molécula.
3. Si es satisfactoria la verificación inicial se procede a buscar el fragmento dentro la lista de enlaces de la molécula candidata.
4. Se selecciona el primer enlace del fragmento, y se busca un semejante entre los enlaces de la molécula candidata (iguales átomos, igual pendiente, igual ángulo entre 3 átomos), obteniéndose una lista con las conexiones semejantes encontradas. Si esta lista es vacía se descarta la molécula candidata y se toma la siguiente.
5. De ser satisfactoria esta búsqueda, se toma el primer enlace de los encontrados como semejantes y se almacena en una lista temporal y se procede a buscar la conexión siguiente del fragmento.
6. A partir de las siguientes búsquedas, verificar la relación entre cada enlace nuevo encontrado con el anterior, ya que estos deben estar conectados para finalmente formar una cadena, así sucesivamente se continúa hasta llegar al final o hasta que no se encuentren coincidencias.

Si la búsqueda concluye satisfactoriamente se obtiene como resultado una lista de enlaces que no es más que el fragmento que inicialmente se estaba buscando y que se encontró dentro de la molécula candidata.

3.3.2 Detrás de la tercera dimensión

Por otra parte, realizar una búsqueda en 3D dimensiones, lejos de constituir un nuevo problema es sólo una condición más que se debe agregar a la búsqueda, ya que los principios se conservan y el modelo matemático es el mismo.

1. Al igual que la búsqueda en 2D el primer paso consiste en comprobar que dentro de la molécula candidata existen los átomos y los enlaces del fragmento de entrada.
2. Comparar la distancia en 3D entre cada par de átomos de la entrada y sus correspondientes en la molécula candidata utilizando la fórmula de distancia en 3D.
3. Comparar el ángulo entre todas las combinaciones de 4 átomos adyacentes (ángulo diedro o ángulo de torsión).

El ángulo diedro es *"la porción de espacio determinada por dos semiplanos con la arista común, a la cual se llama arista del diedro. Cada uno de los dos semiplanos se llama cara del diedro…el rectilíneo de un diedro es el ángulo plano que resulta de cortar el diedro por un plano perpendicular a su arista…la medida de un diedro es igual a la medida de su rectilíneo"* [9]

$$\cos\theta = \frac{|n_1 n_2|}{|n_1||n_2|}$$

La metodología del algoritmo de búsqueda en 3D es en principio la misma que la del algoritmo en 2D, simplemente en este tipo de búsqueda para que dos enlaces sean semejantes deben cumplir además que sus distancias en 3D y los ángulos diedros formados entre cada conjunto de 4 átomos adyacentes sean iguales o por lo menos lo suficientemente semejantes teniendo en cuenta los errores fijados por el que hace uso de la herramienta, que es el responsable de fijar la precisión de la búsqueda.

3.4 Diagrama de clases

En el siguiente diagrama solo se tienen en cuenta aquellas clases de importancia medular en la arquitectura de la aplicación, siendo importante destacar que el sistema se encuentra dividido en dos subsistemas, ya que se implementó de manera independiente una aplicación cliente y una servidora capaces de comunicarse entre sí cuando las acciones del usuario que maneja la aplicación así lo requieren.

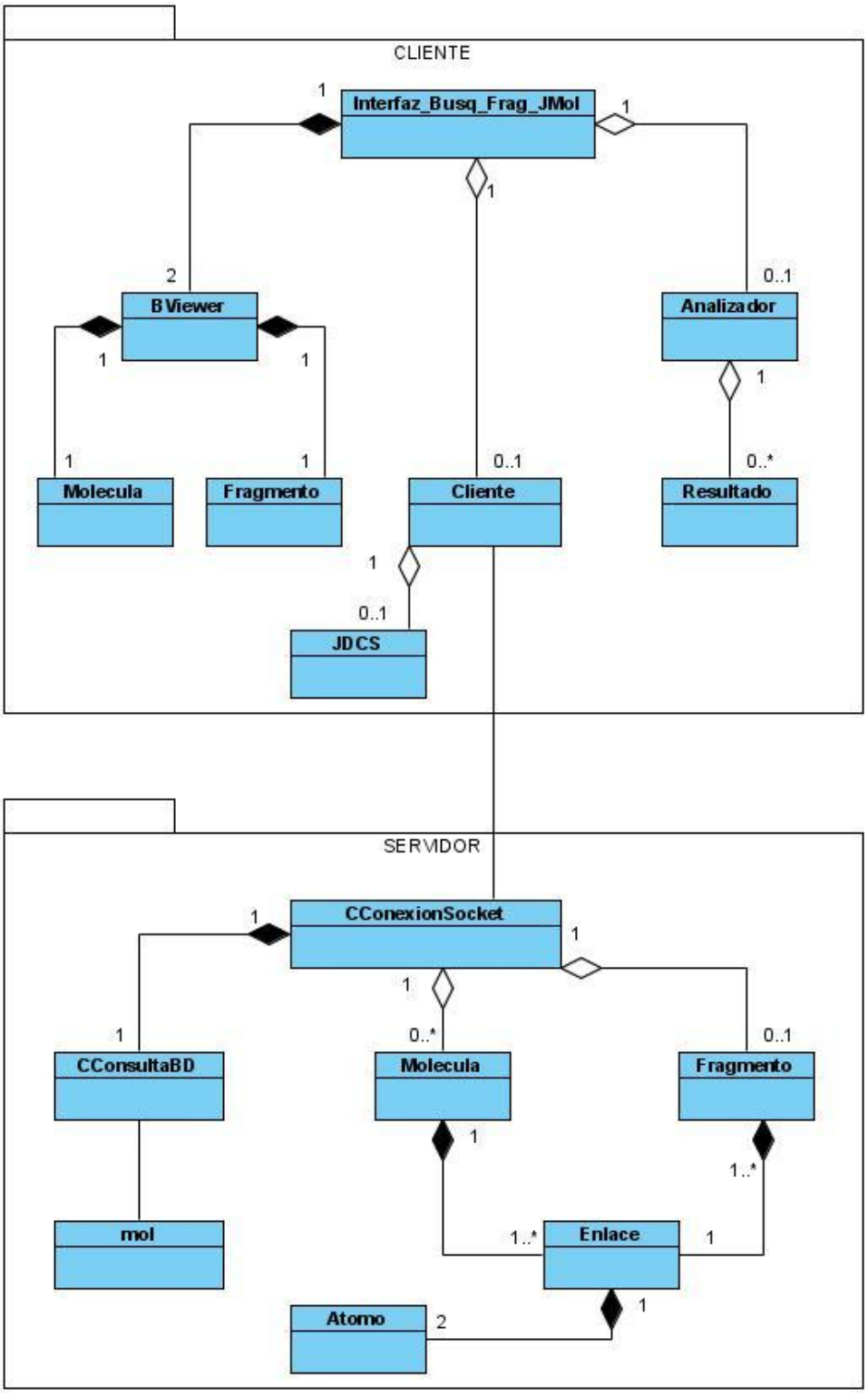


Figura 3. 2: Diagrama de clases.

3.4.1 Clases persistentes

Las clases persistentes representan a los objetos cuya información necesita ser guardada en algún dispositivo o medio de almacenamiento.

El sistema en cuestión utiliza una clase persistente, donde se almacenan los datos de las moléculas en formato .mol en dos columnas llamadas Valores_MDL_2D y Valores_MDL_3D, esta es la tabla que se utiliza para obtener los datos y compararlos con el fragmento proporcionado por el usuario. La estructura de esta tabla está representada en la figura 3.3:

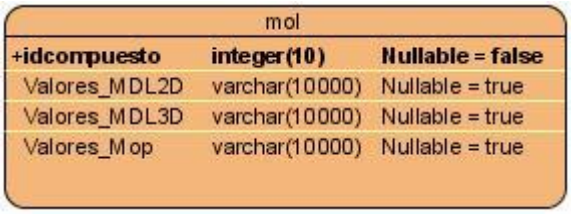


Figura 3. 3: Clases persistentes

3.5 Complejidad de los algoritmos

A menudo se piensa que un algoritmo sencillo no es muy eficiente. Sin embargo, la sencillez es una característica muy interesante a la hora de diseñar un algoritmo, pues facilita su verificación, el estudio de su eficiencia y su mantenimiento.

Respecto al uso eficiente de los recursos, éste suele medirse en función de dos parámetros: el espacio, es decir, memoria que utiliza, y el tiempo, lo que tarda en ejecutarse. Ambos representan el costo que supone encontrar la solución al problema planteado mediante un algoritmo.

El tiempo de ejecución de un algoritmo va a depender de diversos factores como son: los datos de entrada que le suministremos, la calidad del código generado por el compilador para crear el programa objeto, la naturaleza y rapidez de las instrucciones máquina del procesador concreto que ejecute el programa, y la complejidad intrínseca del algoritmo.

Se entiende por tamaño de la entrada el número de componentes sobre los que se va a ejecutar el algoritmo. Por ejemplo, la dimensión de un vector a ordenar, el tamaño de una matriz o simplemente la longitud de una lista.

La unidad de tiempo a la que siempre deben hacer referencia estas medidas de eficiencia no puede ser expresada en segundos o en otra unidad de tiempo concreta, pues no existe un ordenador estándar al que puedan hacer referencia todas las medidas. Es por ello que es muy común denotar por T(n) el tiempo de ejecución de un algoritmo para una entrada de tamaño n.

A la hora de medir el tiempo, siempre se hace en función del número de operaciones elementales que realiza dicho algoritmo, entendiendo por operaciones elementales (OE) aquellas que el ordenador realiza en tiempo acotado por una constante. Así, se consideran OE las operaciones aritméticas básicas, asignaciones a variables de tipo predefinido por el compilador, los saltos, las comparaciones lógicas y el acceso a estructuras indexadas básicas, como son los vectores y matrices. Cada una de ellas contabilizará como 1 OE.

En resumen, el tiempo de ejecución de un algoritmo va a ser una función que mide el número de operaciones elementales que realiza el algoritmo para un tamaño de entrada n.

En el anexo 1 y anexo 2 se muestra el número de operaciones elementales de estos algoritmos y su tiempo de ejecución.

Cota Superior

Dada una función f, se desea 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 denota por $O(f)$. 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.

Para simplificar, por lo general se asume que dado un algoritmo su orden de complejidad es $O(f)$ si su tiempo de ejecución para el peor caso es de orden O de f, es decir, T(n) es de orden $O(f)$.

Teniendo en cuenta las propiedades de la cota superior y los resultados obtenidos al analizar el tiempo de ejecución de los algoritmos que para el caso del algoritmo de 2D fue de $T(n) = 77n^3 + 124n^2 + 109n + 86$ y para el de 3D de $T(n) = 77n^3 + 114n^2 + 290n + 142$ se determinó que la cota superior de la complejidad de ambos algoritmos es del orden $O(n^3)$, aunque teniendo en cuenta que el algoritmo de 3D puede realizar un mayor número de operaciones elementales que su homólogo en 2D.

Se debe recordar que se consideran algoritmos muy eficientes aquellos con complejidad $\log(n)$, eficientes lo de complejidad $O(n^3)$ e ineficientes los de complejidad 2^n . Claramente estos patrones pueden varias según la cantidad de entradas y la velocidad de procesamiento de la computadora que lo ejecute. La bibliografía indica que aquellos algoritmos con complejidad menor que 2^n son teóricamente solucionables. Por lo tanto se puede concluir que aunque no se trate algoritmos óptimos, estos resuelven el problema alcanzando una complejidad razonable.

3.6 Integración con la plataforma de cálculo distribuido

El cálculo distribuido se basa esencialmente en dividir el cálculo o trabajo completo en partes más pequeñas que un ordenador común pueda realizar en un plazo razonable de tiempo. Generalmente un ordenador u ordenadores centrales se encargan de repartir el trabajo entre los distintos ordenadores (unidades de trabajo) que se unen a la plataforma, estos utilizan un programa especial que realiza el cálculo o proceso, una vez terminado, les devuelve el resultado, recibiendo otra pequeña parte del trabajo, así hasta la terminación del mismo.

La plataforma GRAph TOol cuenta entre sus servicios un sistema de cálculo desarrollado por un grupo de estudiantes y profesores. Aunque aún no ha sido integrado a una Grid, se ha logrado que funcione de forma autónoma.

En su tesis de maestría, uno de los profesores deja bien definidos cuales son los pasos para ejecutar un cálculo en dicha plataforma:

"Para ejecutar un cómputo distribuido, el programador necesita solamente extender dos clases en Java: DataManager y Algoritm, que vienen programadas y predefinidas en el sistema. También se pueden usar bibliotecas adicionales de Java u otras clases definidas por el usuario". [10]

Como bien se expresa el profesor, las implementaciones de dos interfaces son necesarias para ejecutar un cálculo en la plataforma distribuida:

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…el constructor no debe recibir parámetros. Las inicializaciones deben hacerse leyendo de un fichero de texto" [10]

```
public class ExDataManager extends DataManager {

    private PrintStream ps;
    private int resultsreceived, unitsIssued, totalUnits, size, inicio, cont;
    private Logger problemLog;
    private String jmol;
    private ArrayList listConsulta;
    private PrintStream t;
    private Scanner lee;

    public ExDataManager() throws Throwable {
        //Cuerpo del constructor de la clase
    }

    public void closeResources() throws Throwable {
        //Cuerpo del método
    }

    public Vector generateWorkUnit(ClientInfo arg0) throws Throwable {
        //Cuerpo del método
    }

    public String getStatus() throws Throwable {
        //Cuerpo del método
    }

    public boolean processResults(Long arg0, Vector results) throws Throwable {
        //Cuerpo del método
    }

    public void leerJmol() throws FileNotFoundException {
        //Cuerpo del método
    }
}
```

Figura 3. 4: Declaración de la clase ´ExDataManager´

En la figura 3.4 se pudo observar la declaración de la clase Extendida “ExDataManager”, en ella se pueden observar sus atributos y métodos fundamentales. Todas estas funciones deben ser implementadas por el programador que quiera hacer uso de la plataforma de cálculo distribuido. El cuerpo de toda la clase y todos los métodos se puede encontrar en el Anexo 3.

Clase Algorithm

“La clase Algoritm tiene parte del código que corre en el cliente y procesa las unidades de trabajo generadas por el método generateWorkUnit de la clase DataManager. Solamente un método se debe implementar en Algorithm y tiene por nombre processUnit” [10]

```
public class ExAlgorithm extends Algorithm implements Serializable {

    public Vector processUnit(Vector arg0) throws Throwable {
        //cuerpo del método
    }

}
```

Figura 3.5: Declaración de la clase ´ExAlgorithm´

El método processUnit() es el encargado de procesar los datos generados por una unidad de trabajo. El cuerpo completo de esta clase se puede encontrar en el Anexo 4.

Finalmente es importante señalar el funcionamiento del sistema de cálculo distribuido, para ello nada mejor que un esquema donde se observa el flujo de trabajo que se establece entre un usuario que realiza una petición de cómputo al servidor y la distribución del trabajo de este entre sus nodos clientes.

Importante destacar que como se mencionó anteriormente el usuario es el que provee el Algorithm y el DataManager además de los datos iniciales del problema y el servidor se encargará de distribuir el trabajo en las máquinas clientes y brindarles su parte del problema y el algoritmo necesario para resolverlo.

En cuanto a las unidades de trabajo estas se generan cuando ocurre una petición de trabajo de un cliente, siempre que la pila de unidades de trabajo expiradas este vacía y permanezcan cálculos sin resolver.

El cómputo terminará cuando ya no quede trabajo en el servidor, se empaquete el resultado y se devuelva al usuario inicial.

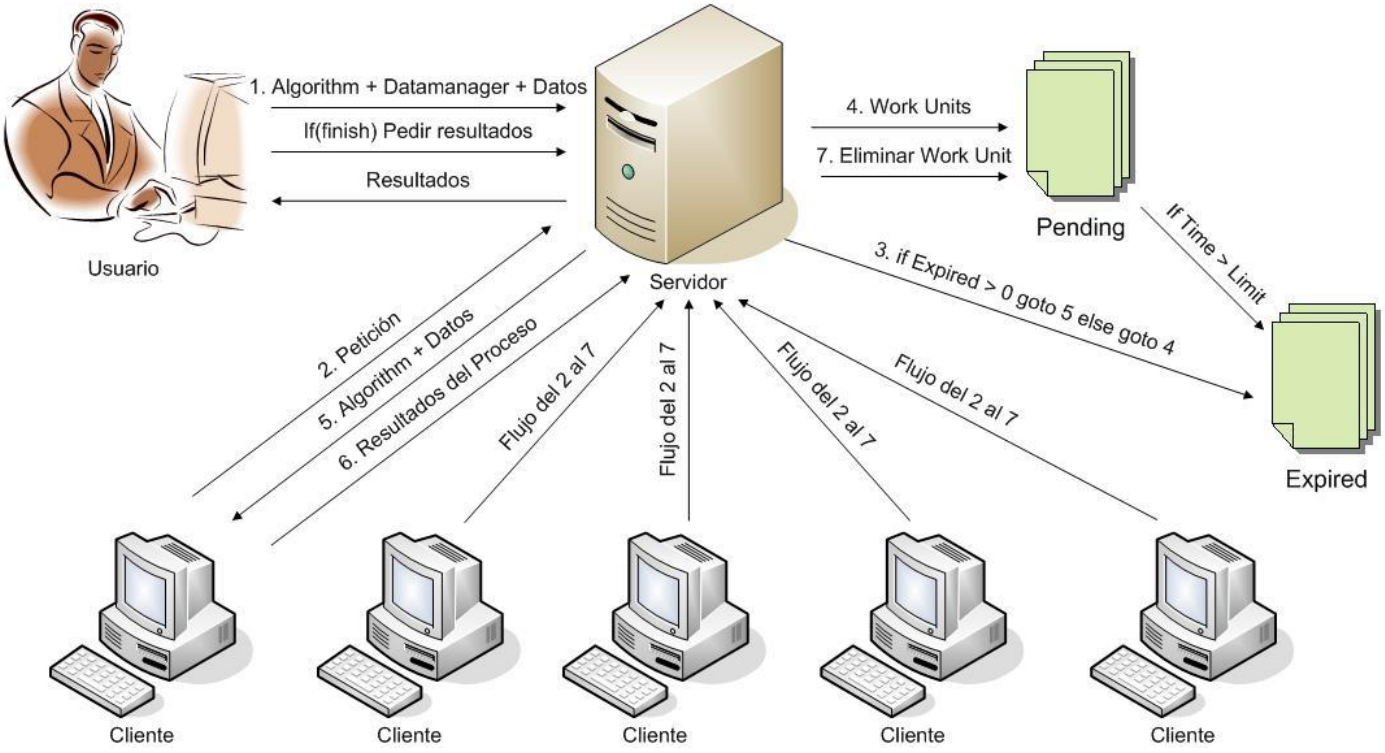


Figura 3. 6: Funcionamiento del sistema de cálculo distribuido.

3.7 Características del sistema de prueba

A pesar de no seguir una metodología de desarrollo, y no estar entre los objetivos de la investigación, se logró obtener un prototipo funcional de un sistema que permite realizar la búsqueda de fragmentos de estructuras químicas utilizando los algoritmos diseñados. Esta aplicación presenta una interfaz sencilla, brindando funcionalidades básicas que permiten a un especialista seleccionar un fragmento y realizar su búsqueda en la base de datos de la Plataforma.

La interfaz gráfica está dividida en 6 secciones que se pueden observar en la figura 3.7, estas son:

1. **Barra de menú:** La barra de menú le da navegabilidad a la aplicación, permitiéndole al usuario realizar las distintas tareas. Entre otras cosas permite importar y exportar moléculas en formato .mol o exportar fragmentos y moléculas como imágenes, seleccionar todo o parte de la molécula, configurar el servidor de base de datos, servidor JDCS y los parámetros de la búsqueda y abrir o almacenar los resultados.

2. **Área de edición:** En esta región se puede seleccionar el fragmento de la molécula que se desea buscar. Posteriormente se le pueden aplicar opciones disponibles en la barra de menú.

3. **Área de visualización:** En esta área se visualizan las moléculas de la base de datos, tanto si fueron seleccionadas en la región 4 como en la región 5.

4. **Explorador de resultados:** En esta región se visualizan los resultados a medida que vayan apareciendo. El usuario tendrá la posibilidad de visualizarlos en la región 3 con solo dar un clic sobre él.

5. **Explorador de la base de datos:** El explorador de la base de datos permite al usuario explorar y visualizar las moléculas que se encuentran almacenadas dentro de la base de datos, esto posibilita realizar la búsqueda a partir de uno de estos elementos en el caso que no se posean elementos almacenados en algún dispositivo local.

6. **Barra de estado:** La barra de estado permite al usuario ver el estado de la búsqueda a partir del porciento del trabajo realizado y el tiempo de búsqueda.

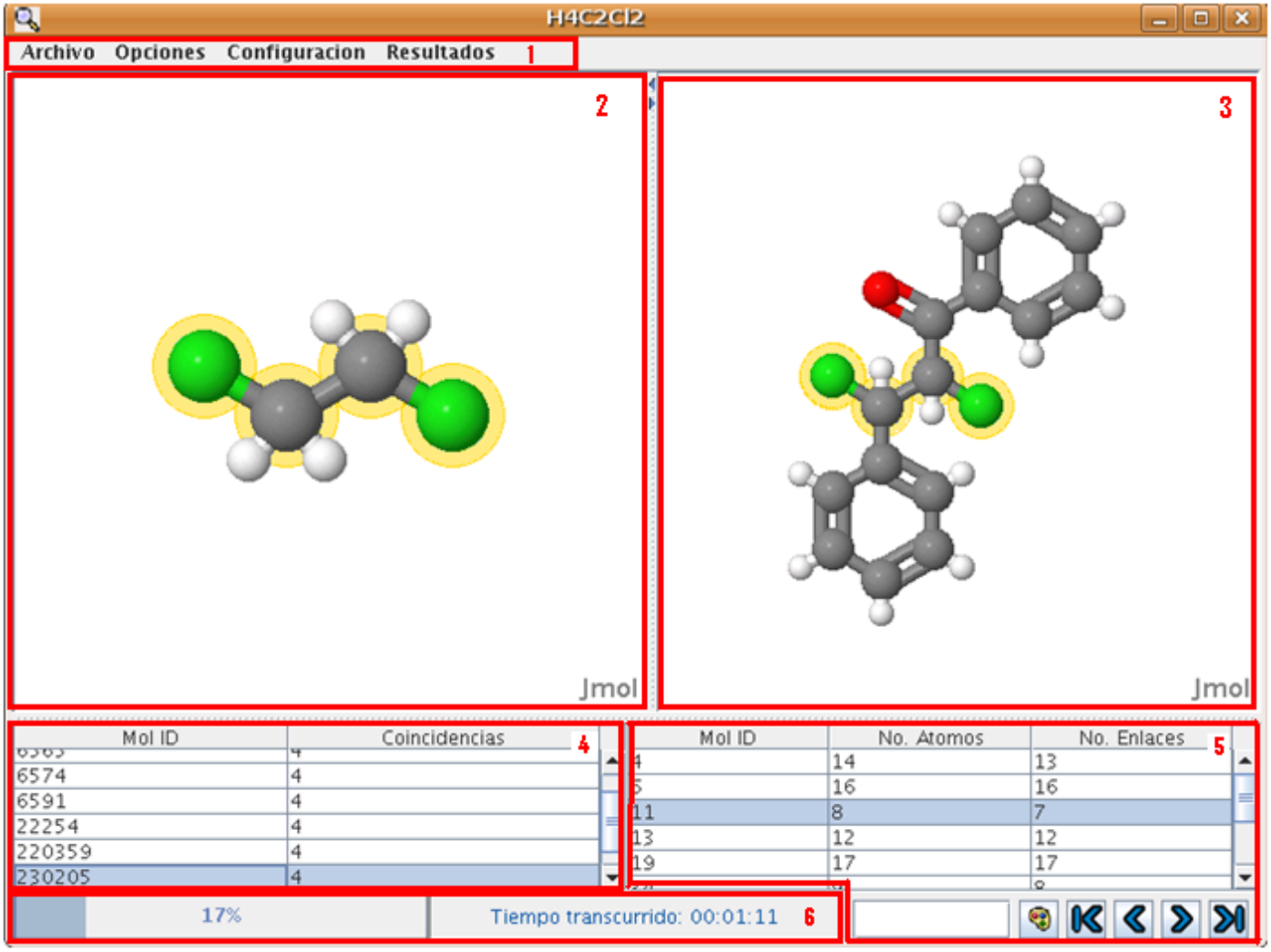


Figura 3. 7: Interfaz gráfica del software de prueba.

3.8 Resultados experimentales

3.8.1 Descripción de los experimentos.

Para comprobar la efectividad de los algoritmos se seleccionó un espacio de búsqueda compuesto por las 100 primeras moléculas de la base de datos y dentro de este se tomó una muestra de 10 moléculas al azar y a cada una de ellas se le hizo corresponder un número del 1 al 13 correspondientes a la cantidad de átomos del fragmento a seleccionar.

Para que la selección de las moléculas fuera realmente al azar se realizó un sencillo programa en la consola del Eclipse que permitió generar 10 números aleatorios entre 1 y 100 correspondientes a la posición de las moléculas dentro de la base de datos (Anexo 5), las que formarán la muestra.

Al ejecutar este programa se obtuvo la colección de números aleatorios 11, 13, 84, 33, 86, 25, 99, 98, 7, 45 respectivamente.

Como se comentó anteriormente, de las 10 moléculas seleccionadas se conformaron fragmentos de 4 a 13 átomos y se realizaron búsquedas de estos fragmentos en 2 y 3 dimensiones.

Para comprobar la exactitud de los algoritmos se realizó una inspección visual de las 100 moléculas para identificar que ningún fragmento fuera pasado por alto.

Todos los errores (ángulo, distancia y ángulo Diedro) fueron fijados al 10 %, lo cual constituye un margen de error bastante bajo teniendo en cuenta que mayormente a los químicos lo que les interesa es la cadena en sí y no la disposición topológica y topográfica que esta posea.

3.8.2 Discusión de los resultados experimentales.

Al realizar las pruebas se obtuvieron los resultados que muestra la siguiente tabla:

Posición Molécula	Id Molécula	Átomos	Tiempo (ms)	Resultados
11	78	4	788	1
13	86	5	925	1
84	326	6	1707	11
33	191	7	987	3
86	335	8	741	1
25	157	9	710	1
99	397	10	701	1
98	389	11	703	2
7	40	12	711	2
45	248	13	846	1
			$\mu = 881{,}9$	$\mu = 4{,}9$

Tabla 3. 1: Resultados de la búsqueda en 2 dimensiones

Como se puede apreciar en la tabla, los tiempos de búsqueda en 100 moléculas de la base de datos oscilan entre 700 y 1800 milisegundos aproximadamente, o sea un poco menos de dos segundos en el peor de los casos. En todos los casos el número de coincidencias es al menos uno, ya que un fragmento siempre se encuentra en la molécula de la cual fue extraído. La media del tiempo fue de 881.9 segundos y se obtuvieron 4.9 coincidencias como promedio. En los siguientes gráficos se puede apreciar la relación entre el número de átomos y los resultados comparados con el tiempo:

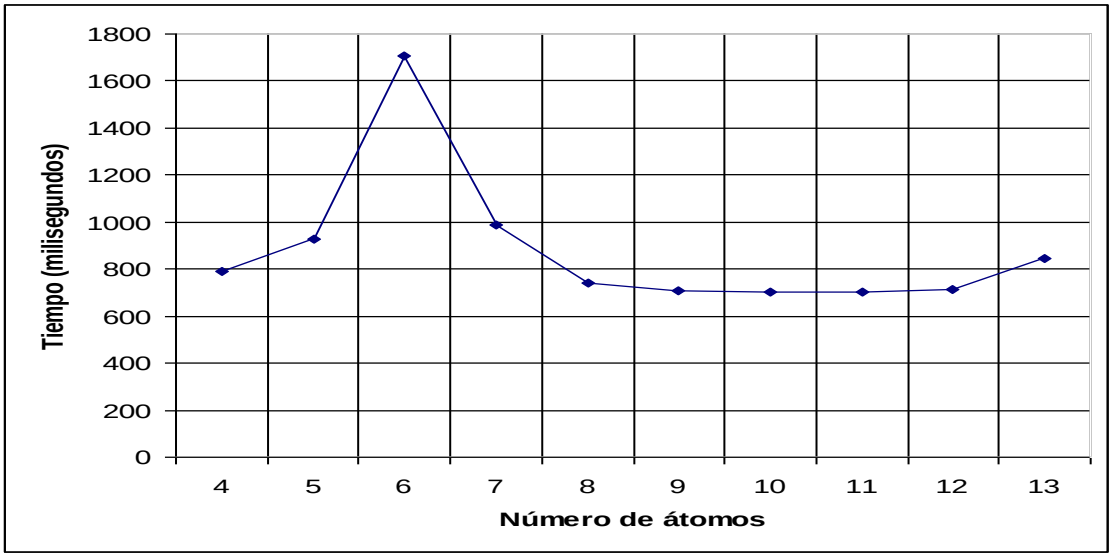


Gráfico 3.1: Relación número de átomos - tiempo en la búsqueda en 2D.

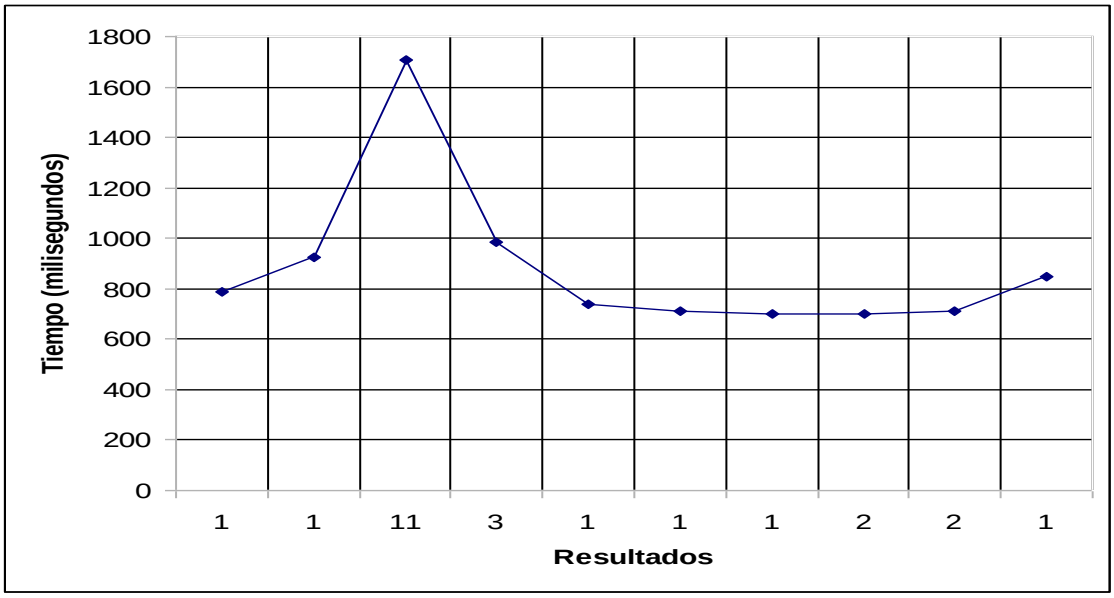


Gráfico 3.2: Relación resultados - tiempo en la búsqueda en 2D.

Llama la atención que en ambos gráficos la tercera búsqueda ha disparado el tiempo fuera de lo que se esperaba como pasa en el resto de los casos. El gráfico 3.1 por si solo no ofrece una respuesta coherente para ello, pues se trata de un fragmento de sólo 6 átomos y tanto sus vecinos de menos como de mayor número de átomos presentan una ligera tendencia hacia un mismo tiempo aproximadamente.

Por otra parte al observar el gráfico 3.2 que muestra información relacionada con el número de resultados obtenidos, se puede notar que el fragmento número 3 ha tenido un número de resultados considerablemente mayor que el resto de los fragmentos, esto se debe a que esta es una cadena muy común que se denomina anillo bencénico y es muy frecuente en los compuestos orgánicos.

La razón por la cual el algoritmo se demora mucho más en fragmentos que son muy comunes se debe sencillamente a que en esos casos se tienen que evaluar todas las condiciones del algoritmo, ya que la mayor parte de las condiciones resultarán satisfactorias.

Si se deseara estimar el tiempo mínimo que demoraría una búsqueda en la base de datos completa, pues se puede comenzar con establecer una sencilla regla de 3 tomando como tiempo de búsqueda la media del tiempo de la tabla 3.1:

100 moléculas ---------- 881.9 milisegundos
180 000 moléculas ---------- x segundos

Por tanto en 180 000 moléculas la búsqueda se demoraría como mínimo:

$$\frac{180\,000 * 881.9}{100} = 1587420\ milisegundos = \frac{1587420}{1000}$$
$$= 1587.42\ segundos = \frac{1587.42}{60} = 26.46\ minutos$$

Claramente este resultado es con un espacio de búsqueda ideal, reducido solo a 100 moléculas de la base de datos, por lo tanto este tiempo hay que considerarlo como una cota mínima, ya que si se tienen en cuenta la carga de transacciones que la base de datos puede alcanzar cuando la búsqueda ha llegado a cierto punto, este tiempo puede tender hacia un aumento que por lo general, según indican las pruebas realizadas no excede los 35 minutos.

Es importante destacar que aunque un fragmento en 2 dimensiones aparezca un número n de veces, este número no necesariamente tiene que ser el mismo cuando se realiza la búsqueda del mismo fragmento en 3 dimensiones, en este caso se agrega la distribución espacial de los átomos lo cual reduce los resultados de la búsqueda considerablemente, así lo demuestran los datos que ofrece la siguiente tabla que se obtuvo al buscar los mismo fragmentos de la tabla 3.1 en tercera dimensión sobre el mismo espacio de búsqueda:

Posición Molécula	Id Molécula	Átomos	Tiempo (ms)	Resultados
11	78	4	1250	1
13	86	5	1120	1
84	326	6	1025	1
33	191	7	1900	2
86	335	8	1012	1
25	157	9	990	1
99	397	10	980	1
98	389	11	1230	1
7	40	12	1150	1
45	248	13	1036	1
			$\mu = 1169.3$	

Tabla 3.3: Resultados de la búsqueda en 3 dimensiones

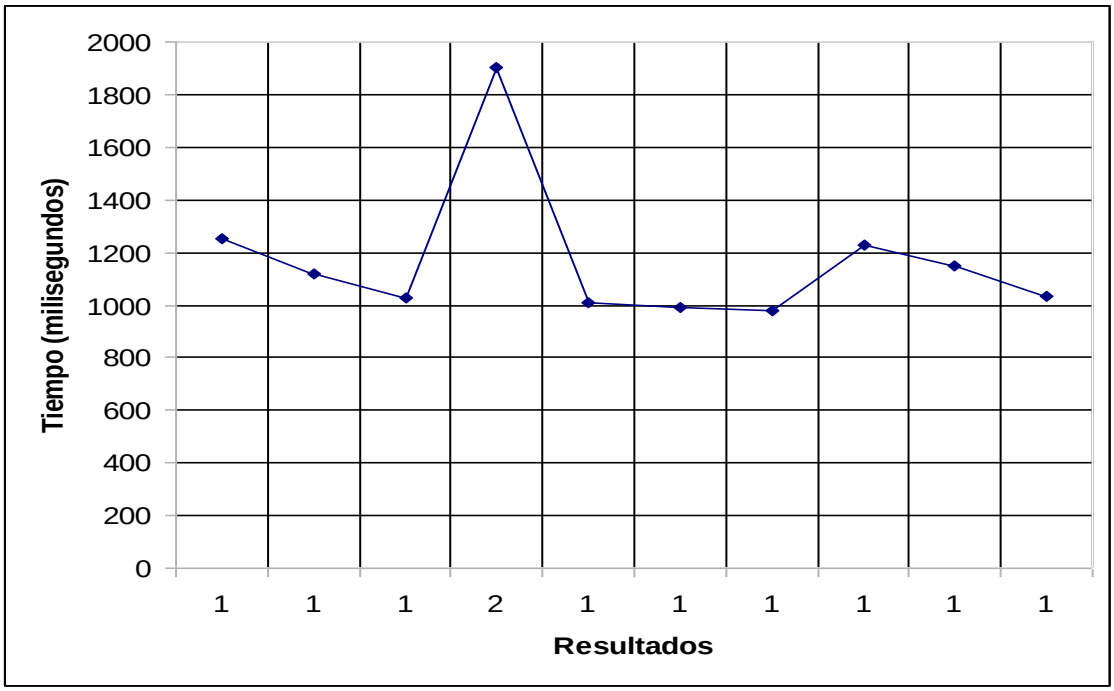


Gráfico 3. 3: Relación resultados - tiempo en la búsqueda en 3D.

La singularidad de las pruebas en 3D radica precisamente en que no es el algoritmo en 3D en sí el que determina en la mayoría de los casos el tiempo de ejecución del programa, sino el mismísimo algoritmo en 2D que se llama implícitamente dentro de este que es el que realiza la primera decantación de las moléculas candidatas para aplicarle los criterios de 3D.

En ambos casos se pudo comprobar mediante inspección visual de las 100 primeras moléculas de la base de datos y con la colaboración de la tutoría, que los resultados obtenidos son correctos teniendo en cuenta los parámetros fijados.

Para ver los fragmentos utilizados en las pruebas se invita al lector a observar las imágenes del Anexo 6 y Anexo 7.

3.9 Aplicaciones

El número de estructuras químicas y la cantidad de compuestos con actividad biológica comprobada se multiplica cada año. Las técnicas de comparación de secuencias, alineación múltiple, análisis filogenético y búsquedas por similitud en bases de datos se están haciendo indispensables en los laboratorios de los químicos farmacéuticos, facilitando el trabajo de estos en tareas tan importantes como la identificación de genes y en la predicción de actividad biológica.

Las búsquedas por similitud, las búsquedas de farmacóforos, la modelación molecular, entre otras son algunas de las tantas aplicaciones en las que se pueden utilizar software como el que se propone en esta investigación.

Adicionalmente la utilización de estos programas constituye hoy día un considerable ahorro de personal, tiempo y recursos en los centros de investigación científica en los que se utiliza, además de la consiguiente disminución de los errores humanos y el aumento de la calidad de los resultados.

En Cuba, donde se promueve la actividad científica, existen numerosos centros científicos-productivos con la capacidad para producir medicamentos, vacunas, compuestos, etc. Queden, por tanto los resultados de esta investiga a disposición de esos investigadores y de los especialistas del Centro de Química Farmacéutica.

3.10 Conclusiones del capítulo

En este capítulo final se presentó un modelo conceptual que permitió entender los procesos que se llevan a cabo en un software con las características señaladas. Se explicaron los algoritmos diseñados y se hizo referencia al aprovechamiento de los recursos de cómputo que se poseen en la Plataforma. Igualmente se analizó la complejidad de los algoritmos, arribándose a la conclusión de que se trata de algoritmos eficientes lo que se comprobó con los tiempos razonables que se alcanzaron durante las pruebas.

Los resultados de los experimentos fueron discutidos y analizados desde el punto de vista de los investigadores, para ello se tabularon los datos y se graficaron estos resultados. Todo esto permitió comprobar la eficacia de los algoritmos implementados que constituyen el objetivo fundamental de esta investigación.

CONCLUSIONES GENERALES

Esta investigación abarcó en total el periodo comprendido entre octubre del 2007 y mayo del 2008. Durante este tiempo se realizó un exhaustivo análisis de la bibliografía disponible sobre el tema. Esta revisión permitió dejar definida la posición de los investigadores respecto al objeto de estudio y trazar una estrategia de trabajo para el resto del desarrollo.

Las principales fuentes bibliográficas se encontraron en Internet, constituyendo estas casi el 70 por ciento de la bibliografía consultada, el resto forman parte de libros de textos y libros de la colección de la biblioteca de la Universidad y la Biblioteca Nacional de Cuba.

Al final de esta investigación se logró alcanzar los objetivos trazados al inicio, así lo demuestran los aspectos que se señalan a continuación:

- Se diseñaron e implementaron algoritmos de búsqueda en 2 y 3 dimensiones que no se encontraron referenciados en la bibliografía consultada.
- Se implementó una aplicación informática funcional, sencilla, con una interfaz gráfica agradable para comprobar los algoritmos.
- Se comprobó la eficacia de los algoritmos a través de la aplicación implementada y su eficiencia a través del análisis de su complejidad.

RECOMENDACIONES

- Incorporar a los algoritmos de búsqueda la capacidad de identificar en una molécula más de una aparición del fragmento buscado.
- Incorporar la posibilidad de conectarse a otras bases de datos con otros formatos de moléculas.
- Realizar el levantamiento formal de requerimientos de software para una distribución formal del sistema que permita desarrollarlo a través del ciclo de desarrollo y con las pautas que propone la metodología Open Basic.

REFERENCIAS BIBLIOGRÁFICAS

1. Secretaria del Evento, Sexta conferencia de examen de los estados partes en la convención sobre la prohibición del desarrollo, la producción y el almacenamiento de las armas bacteriológicas y tóxicas y sobre su destrucción, 2006. Disponible en: http://www.bradford.ac.uk/acad/sbtwc/btwc/rev_cons/6rc/docs/inf/BWC_CONF.VI_INF.4_SP.pdf.

2. Ing. Oscar Zepeda García, ¿Qué es Bioinformática?. 2008. Disponible en: http://www.solociencia.com/biologia/bioinformatica-concepto.htm

3. Adrian Coutin, La información de estructuras químicas y su implicación en el desarrollo de la bioinformática, 2006. Disponible en: http://www.congreso-info.cu/UserFiles/File/Info/Info97/Ponencias/110.pdf.

4. Anónimo, Modelado molecular para química médica, 2006. Disponible en: http://www.forumtecnologico.es /articulos_html/ft_9_html/09-02/index.asp.

5. Daniel Expósito López , Abrahán García Soto, José Martin Gómez, Tablas Hash. Universidad de Granada. Disponible en: http://decsai.ugr.es/~jfv/ed1/tedi/cdrom/docs/tablash.html.

6. Introducción a la Ingeniería de Software. Universidad de las Ciencias Informáticas (UCI), Departamento ISW. 2008. pág. 18.

7. Miguel Ángel Álvarez, Programación Orientada a Objetos, 2008. Disponible en: http://www.desarrolloweb.com/articulos/1540.php

8. Monografias.com, Ley de Cosenos. 2008. Disponible en: http://www.monografias.com/trabajos13/trigo/trigo.shtml#cose.

9. MSN Encarta, Diedro 2008. Disponible en:

http://mx.encarta.msn.com/encyclopedia_961546312/Diedro.html

10. Aguilera Mendoza, Longendri. Sistema de cómputo distribuido aplicado a la Bioinformática. Universidad de las Ciencias Informáticas (UCI), Ciudad de la Habana 2008. pág. 39.

BIBLIOGRAFÍA

1. **Perez Valdez, Yunier Rene y Bravo R, Yanet del Carmen.** Interfaz para la visualización y edición de estructuras químicas. 2007. Universidad de las Ciencias Informáticas (UCI), Ciudad de la Habana, pág 99.

2. **Escalona Arranz, Julio César y Carrasco Velar, Ramón.** Introducción al diseño de fármacos. 2008. Ciudad de La Habana. Editorial Universitaria. ISBN 978-959-16-0647-1.

3. **Carrasco R Padrón, J. A.** Definition of a novel atomic index for QSAR the refractotopological state Revista Cubana de Farmacia. 2000. Suplemento especial, p.329-31, vol. 34.

4. **Autores., Colectivo de.** La Biblia de Java. 2005. http://www.todoprogramas.com/programa/descargar/librolabibliadejava2. ISBN: 8441518653.

5. **Medina-Franco, J. L, López Vallejo, F y Castillo.** Diseño de fármacos asistido por computadora, Edu. Quím. 2007, 17, 452-457.

6. **Guerequeta, Rosa y Vallecillo, Antonio.** Técnicas de Diseño de Algoritmos. 1998. Servicio de Publicaciones de la Universidad de Málaga. http://www.lcc.uma.es/~av/Libro/indice.html. ISBN: 84-7496-666-3.

7. **Gallego Fernández, Isabel y Medina Llinas, Manuel.** Algoritmia y Programación para Ingenieros. 2006.http://www.taringa.net/posts/downloads/1129387/Libro-de-Algoritmica-y-Programacion-para-Ingenieros-(-inform.html. ISBN: 88-5696-625-3 .

8. Modelado Molecular para Química Médica. 2002. http://www.forumtecnologico.es/descargas/REVDescarga/ft_09/art_02_09.pdf.

9. **Jacobson, Ivan, Booch, Grady y Rumbaugh, James.** El proceso Unificado de desarrollo de Software. Editorial Felix Varela, La Habana 2004..

10. **Humprey, Watts S.** Introducción al Proceso de Software Personal. Editorial Felix Varela, La

Habana 2005.

11. **Pressman, Roger.** Ingeniería de Software. Un enfoque práctico. Editorial Felix Varela. La Habana 2005.

12. **Larman, Craig.** UML y Patrones. Editorial Felix Varela, 2004..

13. JMOL. 2007. http://jmol.sourceforge.net/.

14. ERL, P. JME Molecular Editor, Novartis Institutes for BioMedical Research. 2004. http://www.molinspiration.com/jme/.

15. Swing y JFC. SunMicrosystem. 2007. http://www.programacion.com/java/tutorial/swing/.

16. Base de datos Visual de Moléculas. http://www.educaplus.org/moleculas3d/.

17. Diseño Molecular. http://personal5.iddeo.es/pefeco/dibmol.html.

18. Programacion Distribuida y Paralela. 2006. http://lsi.ugr.es/~jmantas/pdp/pdp.html.

19. **Larranz, Teresa.** España presenta el mayor supercomputador de Europa. [En línea] [Citado el: 6 de Noviembre de 2004.] http://www.reuters.es/locales/c_newsArticle.jsp?type=topNews&localeKey=es_ES&storyID=6731260.

20. **Pearson, Kirk.** Internet-based Distributed Computing Projects. 2004. http://www.aspenleaf.com/distributed/ap-science.html#setiathome.

ANEXOS

```
int BusquedaFrag2D(CFragmento frag, double errorAngulo)
{
   int encontrado = 0, cantEncontrados = 0;                    (- 2 OE -)
   if(frag.GetNumAtomosF()> this.numAtomos) return -1;         (- 4 OE -)

   CLista<CConexion> listConexF = frag.GetListCaminoFragF();   (- 3 OE -)
   CLista<CConexion> listPrimConex = EncontrarPrimerasConexiones
   (listConexF.Obtener(0), errorAngulo);                (- 67 + n69 OE -)

   if(listPrimConex.Vacia())  return -1;                       (- 4 OE -)
   for(int p = 0; p < listPrimConex.Longitud(); p++)
   {
      listConexResult = new Clista<CConexion>();               (- 5 OE -)
      listConexResult.Adicionar(listPrimConex.Obtener(p));     (- 11 OE -)
      cantEncontrados = 1;                                     (- 1 OE -)
      listConex.Obtener(listPrimConex.Obtener(p).GetID()).SetCamino(1);
                                                               (- 12 OE -)
     for(int f = 1; f < listConexF.Longitud(); f++)
     {
       encontrado = 0;                                         (- 1 OE -)
       CLista<CConexion> listTempResult = this.BuscarConexCoincidencia
       (listConexF.Obtener(f),errorAngulo);            (- 9 + n50 OE -)
       if(!listTempResult.Vacia())
       {
        for(int t = 0; t < listTempResult.Longitud(); t++)
        {
          if(this.HayConexion(listTempResult.Obtener(t)
          ,listConexResult.Obtener(listConexF .Obtener(f)
          .GetIDConex())) == 1)
          {
            listConexResult.Adicionar(listTempResult.Obtener(t));
            listConex.Obtener(listTempResult.Obtener(t).GetID())
            .SetCamino(1);
            t = listTempResult.Longitud();
            encontrado = 1;
            cantEncontrados++;
          }                                                (- 68 OE -)
        }                                       for t (- 2 + n70 OE -)

        if(encontrado == 0)
        {
```

```
            cantEncontrados = 0;                                    (- 1 OE -)
            this.RestaurarVisitados(listConex);                (- 2 + n7 OE -)
            f = listConexF.Longitud();                              (- 1 OE -)
          }                                                    (- 4 + n7 OE -)
        }
        else
          {
            cantEncontrados = 0;
            this.RestaurarVisitados(listConex);
            f = listConexF.Longitud();
          }                                                (- 6 + n77 OE -)
      }                          for f (- 11 + n50 + n(- 74 + n77 OE -))

    if(cantEncontrados == (listConexF.Longitud()))
      return 1;                                                     (- 4 OE -)
  }
                          for p (- 2 + n( 40 + n50 + n( 74 + n77 OE )))

  return -1;                                                        (- 4 OE -)
}
```

Anexo 1: Tiempo de ejecución del algoritmo de búsqueda en 2D.

```
int BusquedaFrag3D(CFragmento frag, double error, double errorDistancia, double
errorDiedro, int tipo)
{
    CLista<CConexion> listFrag = frag.GetListCaminoFragF();          (- 3 OE -)
    int contador = 0;                                                (- 1 OE -)
       if(tipo == 3)
         {
           if(this.BusquedaFrag2D(frag, error) == 1)
                                                    (77n3 + 124n2 + 109n + 88 OE)
               {
                   CLista<CAtomo> atomosMolecula = this.AtomosEncontradosMol();
                                                              (20n2 + 44n + 22)
                   CLista<CAtomo> atomosFragmento = frag.AtomosEncontradosMolF();
                                                              (20n2 + 44n + 22)
                   for (int i = 0; i < listConexResult.Longitud(); i++)
                   {
                       double porcientoErrorDistancia = (double)((listConexResult.
                        Obtener(i) .GetDistancia3D())*errorDistancia)/100;
                                                                   (- 10 OE -)
                       if((listConexResult.Obtener(i).GetDistancia3D() >=
                       (listFrag.Obtener(i).GetDistancia3D()
                         -porcientoErrorDistancia)) && (listConexResult.Obtener(i)
                             contador++;
                             else return -1;                       (- 30 OE -)
    }                                                        (- 2 + 42n OE -)

       if(frag.GetNumAtomosF()>= 4)
           if (contador == listConexResult.Longitud())
           {
               int signoM = 0;
               double porcientoErrorDiedro,ame1,afe1,restaPM = 0;
               double restaPF = 0;                                  (- 6 OE -)
       for (int i = 0; i < atomosFragmento.Longitud()-3; i++)
       {
           porcientoErrorDiedro = (double)((frag.CalcularAngDiedro(atomosFragmento
            .Obtener(i), atomosFragmento.Obtener(i+1), atomosFragmento.Obtener(i+2)
           atomosFragmento.Obtener(i+3)))*errorDiedro)/100;          (- 10 OE -)
           porcientoErrorDiedro = (double)Math.round(porcientoErrorDiedro*100000)/
            100000;                                                  (- 4 OE -)
           ame1 = (double)Math.round(frag.CalcularAngDiedro(atomosMolecula
           .Obtener(i), atomosMolecula.Obtener(i+1), atomosMolecula
           .Obtener(i+2), atomosMolecula.Obtener(i+3))*100000)/100000;
                                                                    (- 11 OE -)
        porcientoErrorDiedro = (double)Math.round(porcientoErrorDiedro*100000)/
         100000;                                                 (- 4 OE -)
        ame1 = (double)Math.round(frag.CalcularAngDiedro(atomosMolecula
        .Obtener(i), atomosMolecula.Obtener(i+1), atomosMolecula
        .Obtener(i+2), atomosMolecula.Obtener(i+3))*100000)/100000;
                                                                (- 11 OE -)
         afe1 = (double)Math.round(frag.CalcularAngDiedro(atomosFragmento
         .Obtener(i), atomosFragmento.Obtener(i+1), atomosFragmento
         .Obtener(i+2), atomosFragmento.Obtener(i+3))*100000)/100000;
```

```
                                                                    (- 11 OE -)
            signoM = (int)Math.signum(ame1);                         (- 2 OE -)
            if(signoM == -1)
             {
                restaPM = ame1 + porcientoErrorDiedro;
                restaPF = ame1 - porcientoErrorDiedro;
             }
            else
            {
                restaPM = ame1 - porcientoErrorDiedro;
                restaPF = ame1 + porcientoErrorDiedro;
            }                                                        (- 7 OE -)
        if((afe1 >= (restaPM)) && (afe1 <= (restaPF)))
            contador++;
        else return -1;                                              (- 4 OE -)
        }                                                  (- 2 + 51n OE -)
        return 1;
    }
   }
  else  return -1;
  }                                                        (- 8 + 51n OE -)
  else  return -1;
}
```

Anexo 2: Tiempo de ejecución del algoritmo de búsqueda en 3D.

```
public class ExDataManager extends DataManager {

    private PrintStream ps;
    private int resultsreceived;
    private int unitsIssued;
    private int totalUnits;
    private Logger problemLog;
    private int size;
    private int pos;
    private int cantA;
    private Hashtable salida;
    private String place;

    public ExDataManager() throws Throwable {
        // crear un fichero para guardar el progreso del problema
        problemLog = Logger.getAnonymousLogger();
        FileHandler phandler = null;
        File logsDir = new File(PROBLEMDIRECTORY, "probLog");
        logsDir.mkdir();
        phandler = new FileHandler(logsDir.getAbsolutePath()
                + "/problem %g.log", 10000000, 5, false);

        // formateador para el fichero
        SimpleFormatter simple = new SimpleFormatter();
        phandler.setFormatter(simple);
        problemLog.setUseParentHandlers(false);
        problemLog.addHandler(phandler);
        problemLog.info("Entering DataManager");

        size = 2000;
        pos = 0;
        totalUnits = 90; // Cantidad de paquetes
        unitsIssued = 0;
```

```
        // crear un writer para archivo de los resultados
        File r = new File(PROBLEMDIRECTORY, "results.txt");
        FileOutputStream fos = new FileOutputStream(r);
        ps = new PrintStream(fos);

        problemLog.info("DataManager constructor complete");
        prepHasht();
        place = PROBLEMDIRECTORY.getAbsolutePath();
        ThreadConsulta objcon = new ThreadConsulta(place + "/");
        objcon.setPriority( Thread.MIN_PRIORITY );
        objcon.start();
    }

    public void adjustGranularity(double arg0) throws Throwable {
        // TODO Auto-generated method stub
    }

    public void closeResources() throws Throwable {
        try {
            ps.close();
        } catch (Exception e) {
        }
        Handler[] handlers = problemLog.getHandlers();
        for (int i = 0; i < handlers.length; i++) {
            problemLog.removeHandler(handlers[i]);
            handlers[i].flush();
            handlers[i].close();
        }
        problemLog = null;
    }
    public Vector generateWorkUnit(ClientInfo arg0) throws Throwable {
        if (unitsIssued == totalUnits) {
            problemLog.info(" Finished Issuing units: " + unitsIssued);
            return null;
```

```
        }
        problemLog.info("Preprocesamiento creado correctamente");

        Vector unit = new Vector();

        File a = new File(place + "/"  + "tempconsulta"
                + (unitsIssued + 1) + ".tem");
        problemLog.info("profich");
        if (a.exists() == false) {
                return null;
        }

        unit.add(salida);
        unit.add(proFic(unitsIssued));
        unit.add(cantA);

        unitsIssued++;
        problemLog.info("Issuing unit:" + unitsIssued);
        return unit;
    }

    public String getStatus() throws Throwable {
        // TODO Auto-generated method stub
        return "Number units processed" + resultsreceived + "out of"
                + totalUnits;

    }

    public boolean processResults(Long arg0, Vector results) throws Throwable {
        String respuesta = (String) results.get(0);
        problemLog.info("Processing results set" + resultsreceived);
        ps.println(respuesta);
        ps.println("\n");
        resultsreceived++;
```

```
        resultsreceived++;

        if (resultsreceived == totalUnits) {
            problemLog.info("Total resultados recibidos - exiting ");
            ps.close();
            return true;
        } else {
            return false;
        }
    }
    public void prepHasht() throws Throwable {
        problemLog.info("Entering prepHash");
        ArrayList prueba = new ArrayList();

        File arch = new File(PROBLEMDIRECTORY, "Frag7.mol");
        problemLog.info("paso el fichero");

        Ficheros frag = new Ficheros(arch);
        problemLog.info("Entering leer fich");

        prueba = frag.leerJmol();
        cantA = frag.getCantidad();
        P prep = new P();
        prep.Preproc(prueba);
        problemLog.info("fin prep hash");
        salida = prep.getHash();
    }

    public ArrayList proFic(int pos1) {
        File b = new File(place + "/" + "tempconsulta" + (pos1) + ".tem");
        problemLog.info("despierto");
        Scanner lee = null;
        ArrayList res = new ArrayList();
        try {
```

```
            lee = new Scanner(b);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        lee.useDelimiter("END,");
        while (lee.hasNext()) {
                res.add(lee.next());
        }
        b.delete();
        return res;
    }
}
```

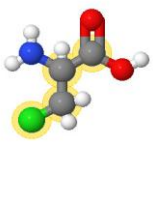
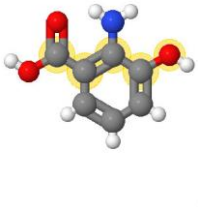
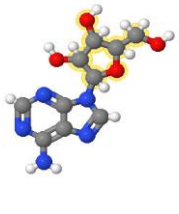
Anexo 3: Clase ExDataManager

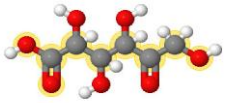
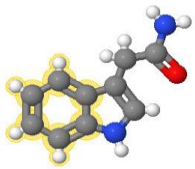
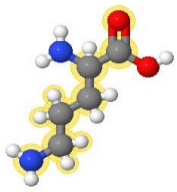
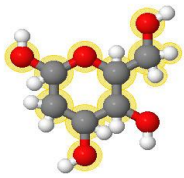
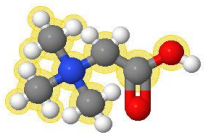
```
public class ExAlgorithm extends Algorithm implements Serializable {

    public Vector processUnit(Vector arg0) throws Throwable {
        String fragMol = (String) arg0.get(0);
        int inicio = (Integer) arg0.get(1);
        int tope = (Integer) arg0.get(2);
        ArrayList listConsulta = (ArrayList) arg0.get(3);

        CFragmento frag = new CFragmento(fragMol);
        int contError = 0;
        ArrayList resultadofinal = new ArrayList();

        int verificar = 0;
        String resultado = "";
        for (int i = 0; i < listConsulta.size(); i++) {
            try {
                CMolecula molecula = new CMolecula((String) listConsulta.get(i));
                int valor = molecula.BusquedaFrag2D(frag, 10);
                if (valor != -1) {
                    verificar = 1;
                    resultado = Integer.toString(molecula.GetID());
                    resultado = resultado + " "+ molecula.IndiceAtomosEncontradosMol();
                    resultadofinal.add(resultado);
                }
                } catch (Exception e) {
                    contError++;
                    }


        }

        Vector fin = new Vector();
        fin.add(resultadofinal);
        return fin;
    }
}
```

Anexo 4: Clase ExAlgorithm

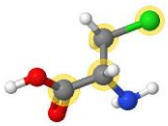
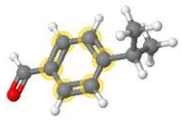
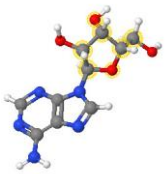
```
public class Consola {
    /**
     * @param args
     */

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        System.out.println("Generando numeros aleatorios para las pruebas del Buscador");

        System.out.println("Las 10 Moleculas Seleccionadas son:");

        Random R = new Random();
        for (int i = 0; i < 10; i++) {
            System.out.println(R.nextInt(100) + "---" + (i+4) + " atomos");
        }

    }

}
```

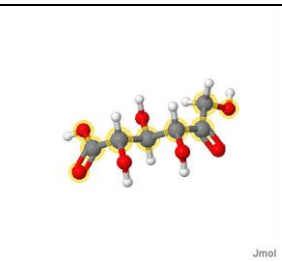
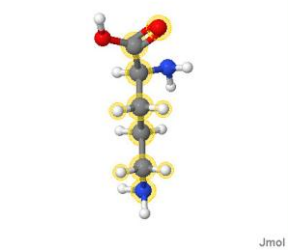
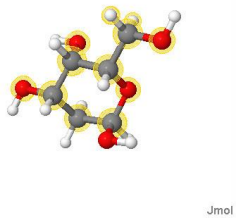
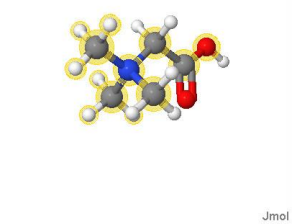
Anexo 5: Detalle de la clase ´Consola´ creada para generar números aleatorios

 **Fragmento 1 – 4 Átomos seleccionados**	 **Fragmento 2 – 5 Átomos seleccionados**
 **Fragmento 3 – 6 Átomos seleccionados**	 **Fragmento 4 – 7 Átomos seleccionados**

 **Fragmento 5 – 8 Átomos seleccionados**	 **Fragmento 6 – 9 Átomos seleccionados**
 **Fragmento 7 – 10 Átomos seleccionados**	 **Fragmento 8 – 11 Átomos seleccionados**
 **Fragmento 9 – 12 Átomos seleccionados**	 **Fragmento 10 – 13 Átomos seleccionados**

Anexo 6: Muestras seleccionadas para las pruebas de búsqueda en 2D

 **Fragmento 1 – 4 Átomos seleccionados**	 **Fragmento 2 – 5 Átomos seleccionados**
 **Fragmento 3 – 6 Átomos seleccionados**	 **Fragmento 4 – 7 Átomos seleccionados**

 **Fragmento 5 – 8 Átomos seleccionados**	 **Fragmento 6 – 9 Átomos seleccionados**
 **Fragmento 7 – 10 Átomos seleccionados**	 **Fragmento 8 – 11 Átomos seleccionados**
 **Fragmento 9 – 12 Átomos seleccionados**	 **Fragmento 10 – 13 Átomos seleccionados**

Anexo 7: Muestras seleccionadas para las pruebas de búsqueda en 3D

GLOSARIO DE TÉRMINOS

Algoritmo: Lista bien definida, ordenada y finita de operaciones que permite hallar la solución a un problema.

Análisis conformacional: Exploración de todos los confórmeros que se pueden obtener de una molécula dada al realizar torsiones alrededor de enlaces sencillos, observando los cambios en la energía molecular asociados a esas torsiones.

Base de Datos Relacional: Se refiere a aquellas Bases de Datos que están de acuerdo con el modelo relacional. Este a su vez se basa en la lógica de predicados y la teoría de conjuntos. Es el sistema más utilizado en la actualidad.

Código Abierto: Término con el que se conoce al software distribuido y desarrollado libremente.

Compilar: Acción que realiza un compilador, este último es un programa que lee un programa escrito en un lenguaje (lenguaje fuente) y lo traduce a un programa equivalente en otro lenguaje (lenguaje objeto).

Confórmeros: Estructuras de un mismo compuesto que se obtienen al girar, alrededor de enlaces simples, la parte de la molécula situada a un lado del enlace con respecto a la localizada al otro lado del mismo.

Copyright: En español derecho de autor. Se refiere al conjunto de normas y principios que regulan los derechos morales y patrimoniales que la ley concede a los autores.

Depurar: En inglés se le conoce como debugging, ya que se asemeja a la eliminación de bichos (bugs), manera en que se conoce informalmente a los errores de programación.

Disponibilidad: Es un sistema y su implementación asociada que asegura un cierto grado absoluto de continuidad operacional.

Entorno de desarrollo: Es un programa compuesto por un conjunto de herramientas para un programador. Puede dedicarse en exclusiva a un sólo lenguaje de programación o para varios.

Escalabilidad: Propiedad deseable de un sistema, una red o un proceso, que indica su habilidad para, o bien manejar el crecimiento continuo de trabajo de manera fluida, o bien para estar preparado para hacerse más grande sin perder calidad en los servicios ofrecidos.

Estructuras cristalinas: Para determinar completamente la estructura cristalina elemental de un sólido, además de definir la forma geométrica de la red, es necesario establecer las posiciones en la celda de los átomos o moléculas que forman el sólido cristalino.

Fármacos: Término farmacológico para cualquier compuesto biológicamente activo, capaz de modificar el metabolismo de las células sobre las que hace efecto.

Frecuencia vibracional: Frecuencia en que vibra la molécula.

Geometría de Equilibrio: Es la conformación en que una estructura logra más estabilidad.

Grafo: Conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos, que permiten representar relaciones entre elementos de un conjunto.

Herencia: En la programación orientada a objeto, es un mecanismo que permite derivar una clase de otra, de manera que extienda su funcionalidad.

Ligandos: Los iones o moléculas que rodean a un metal en un complejo.

Macromoléculas: Moléculas que tienen una masa molecular elevada, formadas por un gran número de átomos.

Matriz: Una ordenación rectangular de elementos algebraicos que pueden sumarse y multiplicarse.

Minería de datos: Consiste en la extracción no trivial de información que reside de manera implícita en los datos. Dicha información era previamente desconocida y podrá resultar útil para algún proceso.

Modularidad: Capacidad que tiene un sistema de ser estudiado, visto o entendido como la unión de varias partes que interactúan entre sí y que trabajan para alcanzar un objetivo común.

Multihilo: Se refiere a la programación multihilos o concurrente, en la cual un programa puede ejecutar dos o más acciones concurrentemente con un solo CPU.

Multiplataforma: Término usado para referirse a los programas, sistemas operativos, lenguajes de programación, u otra clase de software, que puedan funcionar en diversas plataformas.

Multiusuario: Se refiere a sistemas o programas que permiten el acceso de dos o más usuarios a los mismos recursos concurrentemente.

Polimorfismo: Capacidad que tienen los objetos de una clase de responder al mismo mensaje o evento en función de los parámetros utilizados durante su invocación. Un objeto polimórfico es una entidad que puede contener valores de diferentes tipos durante la ejecución del programa.

Portabilidad: Un software depende de la plataforma en la que corre. La portabilidad es mayor cuanto menor es la dependencia del software de plataforma.

Propiedades QSAR: Propiedades de relación estructura-actividad.

Síntesis Química: Es el proceso de obtener compuestos químicos a partir de substancias más simples.

Sistema Gestor de Base de Datos (SGBD): Conjunto de programas que se encargan de crear y mantener una o varias bases de datos, asegurando su integridad, confidencialidad y seguridad.

Tamizaje: Depuración, elección cuidadosa y minuciosa.

Virtual: Representación de las cosas a través de medios electrónicos o representaciones de la realidad.