CAPITULO 5. DESARROLLO

El desarrollo de MolUDLAP (programa para simulación de acoplamiento molecular proteína-ligando) implicó el trabajo e interacción con diversas APIS de desarrollo en JAVA, por lo que durante este capítulo una sección estará destinada a explicar las experiencias encontradas al momento de haber usado las distintas APIS disponibles, así como los puntos a favor y en contra de su cada una.

El proceso del Acoplamiento Molecular.

Antes de llevar a cabo una simulación de la interacción de un ligando sobre una proteína; es necesario asegurarse de que ambas moléculas cumplan con cierto tipo de características, esto es conocido como preparación de las moléculas, esta preparación tiene como finalidad que al llevar a cabo la simulación se obtengan los mejores resultados posibles.

En la figura 4 se muestran los principales pasos que se siguen en el proceso del AC. Estos son:

- Preparación de la proteína.
- Selección del receptor y el ligando.
- Especificación del sitio activo.
- Selección de los parámetros de acoplamiento.
- Correr la aplicación.
- Obtención de resultados.

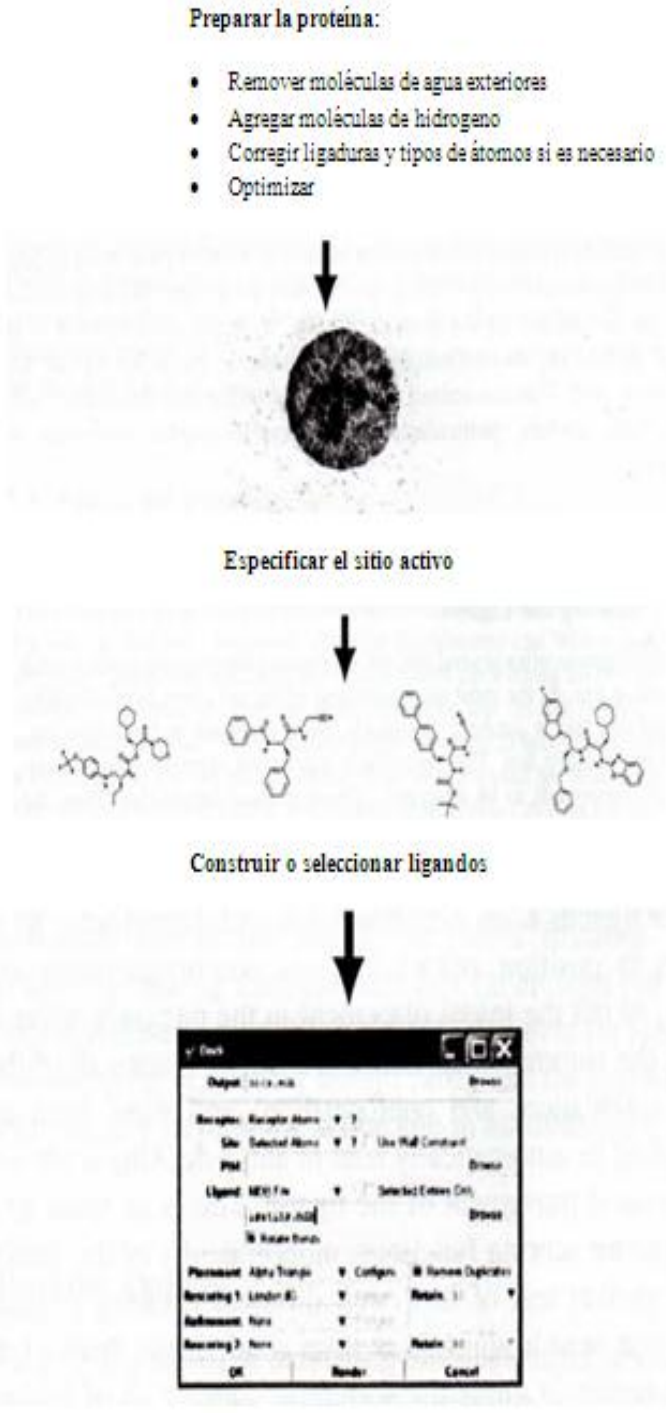


Figura 4. Procedimiento general para una simulación de Acoplamiento Molecular.

Aspectos del Acoplamiento Molecular.

A continuación se describen algunos de los aspectos que se tienen que considerar al momento de desarrollar el proceso de acoplamiento molecular.

Algoritmos de Búsqueda.

El algoritmo de búsqueda posiciona las moléculas en varias ubicaciones, orientaciones y conformaciones dentro del sitio activo, es decir nos ayuda a definir las configuraciones con las que se trabajará.

Grid.

Es una malla de puntos que se utiliza para delimitar el sitio activo (unión) donde se tendrá la interacción entre la Proteína y el ligando, además de que sirve para la ubicación de las coordenadas 3D de las moléculas, permite la asignación de coordenadas en el espacio de configuración para cada uno de los átomos del ligando y el receptor.

Funciones de puntuación.

La evaluación de la función de puntuación es la que nos va a indicar cuál de las configuraciones receptor-ligando obtenidas por la función de búsqueda es la que da como resultado un menor gasto de energía y por lo tanto es la configuración más estable, esta función de puntuación es la que es necesario optimizar.

Force Fields.

En el contexto de mecánicas moleculares, un campo de fuerza (también llamado force field), se refiere a la forma funcional y conjunto de parámetros usados para describir el potencial de energía de un sistema de partículas (típicamente, pero no siempre los átomos). Sirven para calcular las fuerzas de atracción o repulsión entre los átomos de las moléculas.

Los campos de fuerza (force fields) más populares encontrados en la literatura son los siguientes:

- MM2
- MM3
- MM4
- MMFF94 (lo trae implementado el CDK y se trato de usar para los cálculos de la función de puntuación, sin embargo este no funciono).

Aspectos que se deben considerar durante la realización de un programa de Acoplamiento Molecular

Trabajo con la molécula Receptor

Para el modelado de esta parte del problema, se usa a Jmol como visualizador de esta molécula mientras que de CDK se utilizan las funciones que este API nos proporciona para la asignación de cargas de Gasteiger, y para fragmentar la molécula a trabajar se ocupa la función Strands (cadenas).

Es conveniente mencionar que molUDLAP está diseñado de tal manera que el usuario al momento de abrir una molécula y que esta sea visualizada (figura 5), de manera automática se le asignan las cargas de Gasteiger, hay que considerar que las proteínas extraídas en el PDB tienen como formato .pdb.

Figura 5. Visualización de la molécula dock o receptor (proteina) usando Jmol en molUDLAP, el programa agrega de forma automática las cargas de Gasteiger

Trabajo con la molécula ligando

Se ha observado que el formato de una proteína es distinto al de un ligando, por lo que algunas de las herramientas usadas para modelar la proteína no han funcionado para el ligando (ver figura 6).

Se trato de usar la función de total energy del paquete Meshi adaptándola para que dicha función actuara en vez de sobre una proteína , sobre un archivo con el contenido de un ligando y de la cadena de la proteína que el usuario seleccionará; sin embargo debido en primera instancia al distinto formato de un ligando con una proteína y a que se llego a probar esta función sobre una sola proteína en .pdb, se observó que la función total energy de Meshi presento algunas fallas para poder obtener el resultado deseado, por lo que se consideró conveniente no usarla en estos momentos, creemos que esta función no está del todo terminada ya que Meshi es un paquete que está todavía en fase de prueba; cabe señalar que Meshi solo acepta formato .pdb.

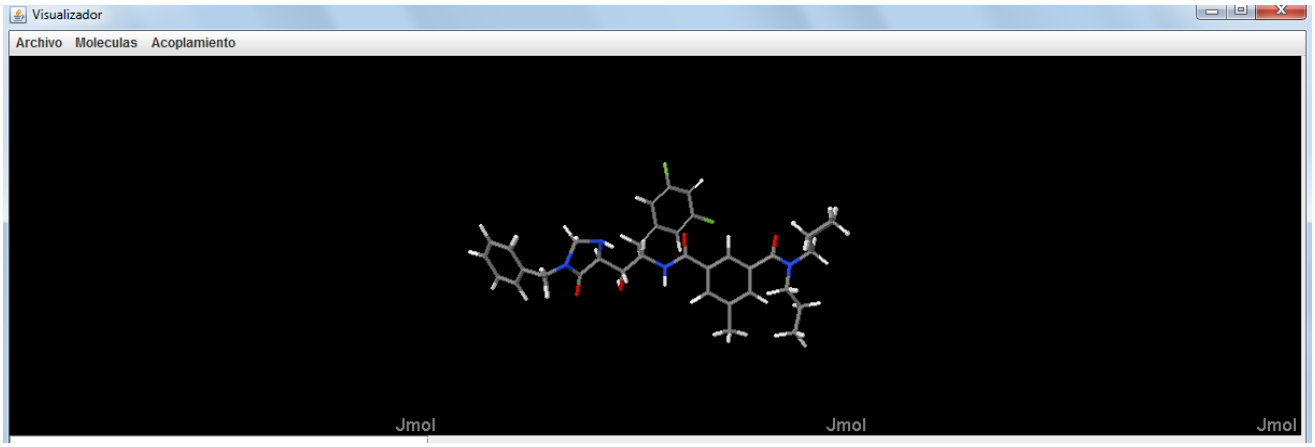


Figura 6. Visualización de la molécula key (ligando) usando Jmol en molUDLAP

Función de Energía

Para esta parte del problema, se trato se usar la función de energía mmff94 (merk molecular force fields 94) de CDK para ser optimizada por el algoritmo de búsqueda, sin embargo no se logró adecuarla al proyecto molUDLAP, debido principalmente a fallas en la forma en que esta función esta implementada por parte de CDK y lamentablemente debido también a la falta de documentación proporcionada por este proyecto, por lo que se consideró como alternativa usar la función de total energy del proyecto Meshi, pero debido a los problemas citados en el párrafo anterior se decidió ya no usarla; debido a los problemas anteriormente expuestos al momento de usar estos dos paquetes (CDK y Meshi), se propuso como alternativa usar una función de energía que considera las interacciones receptor-receptor, ligando-ligando y receptor-ligando para las fuerzas de atracción y repulsión entre los átomos de las moléculas; está función considera un promedio en los parámetros de Lennard Jones para las fuerzas r_{ij}^{12} y r_{ij}^{6}. Se propuso un parámetro para calibrar las distancias que muestra por defecto el grid de CDK con respecto de las distancias reales entre los átomos de las moléculas.

Formula implementada en molUDLAP

Para la función de energía, como se mencionó anteriormente se está empleando el promedio de las tablas de Lennard Jones usadas en la primera versión de Autodock para los términos X^{12} y X^{6}. Para las distancias se propuso un parámetro que relaciona las distancias que por defecto proporciona CDK con respecto de las distancias reales entre los átomos de la proteína y el ligando.

A continuación se muestra la fórmula propuesta, para el proyecto molUDLAP:

$$\mathbf{E=\sum\sum_{prot\text{-}lig}\ (520100.594/(k*r_{ij}^{12})\ -\ 645.146/(k*r_{ij}^{6})\ +\ (m/r_{ij}))\ +\ \sum\sum_{lig\text{-}lig}\ (520100.594/(k*r_{ij}^{12})\ -\ 645.146/(k*r_{ij}^{6})\ +\ (m/r_{ij}))\ +\ \sum\sum_{prot\text{-}prot}\ (520100.594/(k*r_{ij}^{12})\ -\ 645.146/(k*r_{ij}^{6})\ +\ (m/r_{ij}))}$$

Donde se tiene lo siguiente:

El termino $1/r_{ij}^{12}$ representa la fuerza de repulsión de Pauli existente entre dos átomos de una molécula o dos átomos distintas moléculas, siendo r_{ij} la distancia entre dos átomos.

El término $1/r_{ij}^{6}$ representa la fuerza de atracción de Van der Walls existente entre dos átomos de una molécula o entre átomos de diferentes moléculas, siendo r_{ij} como antes la distancia entre dos átomos.

El término $1/r_{ij}$ representa la repulsión electróstatica entre los átomos de las móleculas.

El parámetro k relaciona las distancias que por defecto proporciona el grid de CDK con respecto de las distancias reales entre los átomos de las moléculas, tiene un valor aproximado de $3 <= K <= 6$.

m es el parámetro para calibrar la fuerza electrostática.

Cada r_{ij} depende de los ángulos de rotación alfa (a), beta (b) y gama (g) que representan las rotaciones con respecto de cada uno de los ejes coordenados, por lo que el cambio de cualquiera de estos ángulos nos lleva a una configuración diferente.

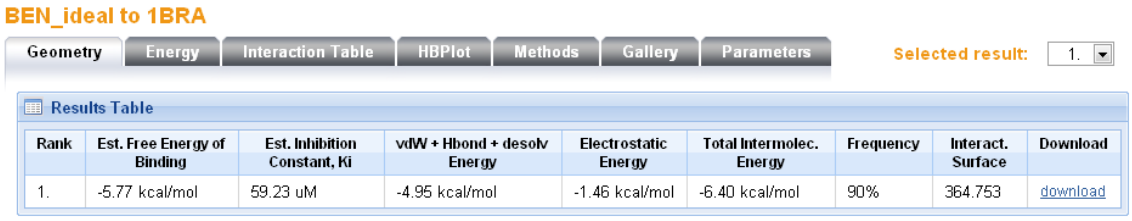
El resultado de esta fórmula es en kcal/mol.

Calibración de la fórmula implementada en molUDLAP

Para mostrar el funcionamiento del molUDLAP con la función de energía propuesta se muestran algunos resultados para la calibración de la fórmula, se toma como referencia los resultados obtenidos en Docking server para 3 ejemplos piloto.

NOTA: En todos los ejemplos se ha seleccionado a la cadena A de la molécula lock (proteína).

Ejemplos:



Rank	Est. Free Energy of Binding	Est. Inhibition Constant, Ki	vdW + Hbond + desolv Energy	Electrostatic Energy	Total Intermolec. Energy	Frequency	Interact. Surface	Download
1.	-5.77 kcal/mol	59.23 uM	-4.95 kcal/mol	-1.46 kcal/mol	-6.40 kcal/mol	90%	364.753	download

Figura 7. Ejemplo piloto #1 tomado de Docking Server.

Figura 8. Resultado del ejemplo anterior con molUDLAP y usando valores de 5.4 y 0.0002 para k y m.

ACP_ideal to 2PTN

Geometry | Energy | Interaction Table | HBPlot | Methods | Gallery | Parameters | Selected result: 1.

Results Table

Rank	Est. Free Energy of Binding	Est. Inhibition Constant, Ki	vdW + Hbond + desolv Energy	Electrostatic Energy	Total Intermolec. Energy	Frequency	Interact. Surface	Download
1.	+734.19 kcal/mol		+729.89 kcal/mol	+0.55 kcal/mol	+730.44 kcal/mol	10%	751.862	download

Figura 9. Ejemplo piloto #2 tomado de Docking Server.

[alpha,beta,gamma](x,y,z):[221.4908,113.4259,325.8872],(-2.8948,-4.901,4.7626) - 799.781 kcal/mol

---

Figura 10. Resultado del ejemplo anterior con molUDLAP y usando valores de 2.0 y 0.0002 para k y m.

9NE_ideal to 1A19

Geometry | Energy | Interaction Table | HBPlot | Methods | Gallery | Parameters | Selected result: 1.

Results Table

Rank	Est. Free Energy of Binding	Est. Inhibition Constant, Ki	vdW + Hbond + desolv Energy	Electrostatic Energy	Total Intermolec. Energy	Frequency	Interact. Surface	Download
1.	+112.31 kcal/mol		+105.47 kcal/mol	+0.64 kcal/mol	+106.11 kcal/mol	10%	642.011	download

Figura 11. Ejemplo piloto #3 tomado de Docking Server.

[alpha,beta,gamma](x,y,z):[49.374,298.7079,159.9602],(-2.8518,8.2452,-2.6833) - 154.547 kcal/mol

---

Figura 12. Resultado del ejemplo anterior con molUDLAP y usando valores de 2.05 y 0.0002 para k y m.

Algoritmo de Búsqueda.

Para el desarrollo de esta parte del proyecto se uso como algoritmo para la optimización de la función de energía a un algoritmo genético, el cual solo cuenta con la función de mutación; siendo los parámetros de entrada el número total de configuraciones definidas por el usuario y generadas por el programa, el número total de sobrevivientes que se tengan del total de la población inicial a ser evaluados, y el número de generaciones que son generadas, así como los parámetro k y m; dando como resultado final de la simulación un resultado gráfico donde se muestre al usuario la(s) configuración(es) más estable(s) de la interacción entre el ligando y la proteína.

Es conveniente mencionar que dicha función está encapsulada, de tal manera que esta tiene la posibilidad de ser mejorada para trabajos futuros sobre la implementación de molUDLAP.

Apis de java disponibles para la implementación de programas de química-informática.

Durante el desarrollo de MolUDLAP se usaron las siguientes APIS de desarrollo:

- Jmol
- CDK (Chemistry Development Kit)
- Meshi

Jmol

Jmol es un visor de moléculas gratuito y de código abierto para estudiantes, profesores e investigadores en química y bioquímica. Es multiplataforma, compatible con sistemas Windows, Mac OS X y Linux/Unix.

JmolApplet es una mini aplicación para el navegador web que puede integrarse en páginas web.

La aplicación Jmol es un programa autónomo en Java que funciona localmente en el ordenador, fuera del navegador.

JmolViewer es un conjunto de herramientas de desarrollo que se puede integrar en otros programas Java.

Prestaciones de Jmol

- Software gratuito y de código abierto autorizado bajo la GNU Lesser General Public License
- Miniaplicación (*applet*), aplicación y componente para la integración en sistemas
    - **JmolApplet** es una miniaplicación para el navegador web que puede integrarse en páginas web. Es ideal para el desarrollo de material docente a través de la web y para bases de datos químicas accesibles por internet. La miniaplicación JmolApplet proporciona una vía de actualización para los usuarios del conector Chime.
    - La **aplicación Jmol** es un programa autónomo que se ejecuta localmente en el ordenador.
    - **JmolViewer** puede integrarse como un componente dentro de otros programas Java.
- Multi-idioma
    - Traducido a numerosos idiomas: alemán (de), catalán (ca), checo (cs), chino (tanto zh_CN como zh_TW) coreano (ko), español (es), francés (fr), holandés (nl), húngaro (hu), italiano (it), portugués de Brasil (pt_BR), turco (tr) (además del inglés americano nativo, en-US y del inglés británico, en_GB).

    - Adopta automáticamente el idioma del sistema operativo del usuario, si está entre las traducciones disponibles. Puedes cambiarlo a otro si quieres.
    - Para información actualizada o instrucciones para añadir tu idioma, visita la Wiki.
- Multiplataforma
    - Windows
    - Mac OS X
    - Linux / Unix
- Compatible con los principales navegadores: Internet Explorer, Mozilla o Firefox, Safari, Opera, Konqueror, IceWeasel, ...
- Representación gráfica tridimensional de alto rendimiento sin requisitos de hardware

- Formatos de archivo (véase también la sección file formats de la Wiki de Jmol):

MOL	Estructura de MDL / Elsevier / Symyx (versión clásica V2000)
V3000	Estructura de MDL / Elsevier / Symyx (versión nueva V3000)
SDF	Estructura de MDL / Elsevier / Symyx (múltiples modelos)
CTFile	Tabla química de MDL / Elsevier / Symyx (genérico)
CIF	Crystallographic Information File - formato estándar de la *Unión Internacional de Cristalografía*
mmCIF	Macromolecular Crystallographic Information File - formato estándar de la *Unión Internacional de Cristalografía*
CML	Chemical Markup Language (lenguage químico de marcado)
PDB	Protein Data Bank - Research Collaboratory for Structural Bioinformatics
XYZ	Formato XYZ, archivo de XMol - Minnesota Supercomputer Institute

XYZ+vib	Formato XYZ con información adicional de vectores de vibración
XYZ-FAH	Formato XYZ para Folding@home
MOL2	Sybyl, Tripos
Alchemy	Tripos
CSF	Estructura química de Fujitsu CAChe, ahora Fujitsu Sygress
GAMESS	Formato de salida de General Atomic and Molecular Electronic Structure System (variantes US y UK) - Gordon Research Group, Iowa State University
Gaussian	Formato de salida de Gaussian 94/98/03 - Gaussian, Inc.
Cube	Gaussian, Inc.
Ghemical	Paquete de química computacional Ghemical
MM1GP	Archivo de mecánica molecular de Ghemical
HIN	Archivos HIN o HIV de HyperChem - Hypercube, Inc.
Jaguar	Schrodinger, LLC
MOLPRO	Formato de salida de Molpro
MOPAC	Formato de salida de MOPAC 93/97/2002 (dominio público)
MGF	Formato de salida graphf de MOPAC 2007 (v.7.101) (dominio público)
NWCHEM	Formato de salida de NWChem - Pacific Northwest National Laboratory
odydata	Datos de Odyssey - WaveFunction, Inc.
xodydata	Datos XML de Odyssey - WaveFunction, Inc.
QOUT	Q-Chem, Inc.
SHELX	Structural Chemistry Department, University of Göttingen (Germany)
SMOL	Datos de Spartan - Wavefunction, Inc.
spinput	Datos de Spartan - Wavefunction, Inc.
GRO	Formato Gromos87 de GROMACS
PQR	Formato pdb modificado que incluye carga y radio

Amber	Paquete de programas de simulación molecular Amber
JME	Java Molecular Editor - Peter Ertl
CASTEP	Paquete de programas CASTEP, utiliza teoría del funcional de la densidad
FHI-aims	Teoría del potencial completo / estructura electrónica con todos los electrones con orbitales locales - Fritz-Haber-Institut der Max-Planck-Gesellschaft
VASP	VASP / VAMP / Vienna ab-initio simulation package
DGrid	Miroslav Kohout, Max-Planck Institute
ADF	Formato de salida de ADF - Amsterdam Density Functional
XSD	Accelrys Materials Studio
AGL	ArgusLab
DFT	Wien2k
AMPAC	Formato de salida de AMPAC - Semichem, Inc.
WebMO	Interfaz WebMO para paquetes de química computacional
Molden	Densidad electrónica / orbitales moleculares
PSI3	Formato de salida del paquete de programas de química cuántica PSI3
CRYSTAL	Formato de salida de CRYSTAL, una herramienta de computación para química del estado sólido y física. Grupo de Química Teórica, Univ. Turín, Italia.

Para el desarrollo de la aplicación se ha usado a Jmol como visualizador, ya que este es muy versátil debido a que permite abrir prácticamente todas las estructuras descargadas del PDB (Protein Data Bank) en cualquier formato.

Además de que se ha logrado que en una misma ventana Jmol pueda visualizar las 2 moléculas necesarias para el desarrollo de la aplicación.

CDK (Chemistry Development Kit)

Chemistry Development Kit (CDK) es una biblioteca Java para la química estructural y bioinformática. Ahora es desarrollado por más de 50 desarrolladores de todo el mundo y se utiliza en más de 10 diferentes academias, así como en proyectos industriales en todo el mundo.

Biblioteca

El propio CDK es una biblioteca en lugar de un programa de usuario. Sin embargo se ha integrado en varios ambientes para que su funcionalidad esté disponible. CDK se utiliza actualmente en diversas aplicaciones, entre las que están CDK-Taverna, Bioclipse y Cinfony.

En 2008 bits de código bajo licencia GPL, fueron retirados de la biblioteca, mientras que los bits de código eran independientes de la biblioteca principal de CDK.

Principales funciones

- Quimioinformática
- Edición y Generación de diagramas 2D
- Generación de geometría 3D
- Búsqueda de subestructuras usando SMARTS
- Cálculos QSAR
- Cálculo de huella digital
- Cálculos de campo de fuerza

De CDK se ha utilizado principalmente la capacidad de su atomcontainer para poder traducir el archivo visual que muestra Jmol a términos de CDK para poder manipular las moléculas, con opciones tales como agregar cargas de Gasteiger, seleccionar cadenas para ubicar el sitio de acción del ligando, etc.

Meshi

Meshi es un paquete de software para el modelado de proteínas. Está escrito en Java exclusivamente en el diseño orientado a objetos (OOD).Hay que tener en cuenta que Meshi se encuentra en una fase bastante preliminar de desarrollo. No todas las características que usted puede esperar ya están disponibles (por ejemplo, Dinámica Molecular), y la documentación no está completa. Inicialmente se intento usar la función de optimización de energía proporcionada por Meshi, sin embargo se encontró que esta no estaba bien desarrollada, por lo que se tuvo que descartar.

Desarrollo de la aplicación.

Para el desarrollo de la aplicación se ha decidido utilizar el model view controller, ya que este modelo da una mejor distribución de los componentes que forman parte del programa.

A continuación se muestra un diagrama de clases (figura 13) de la aplicación:

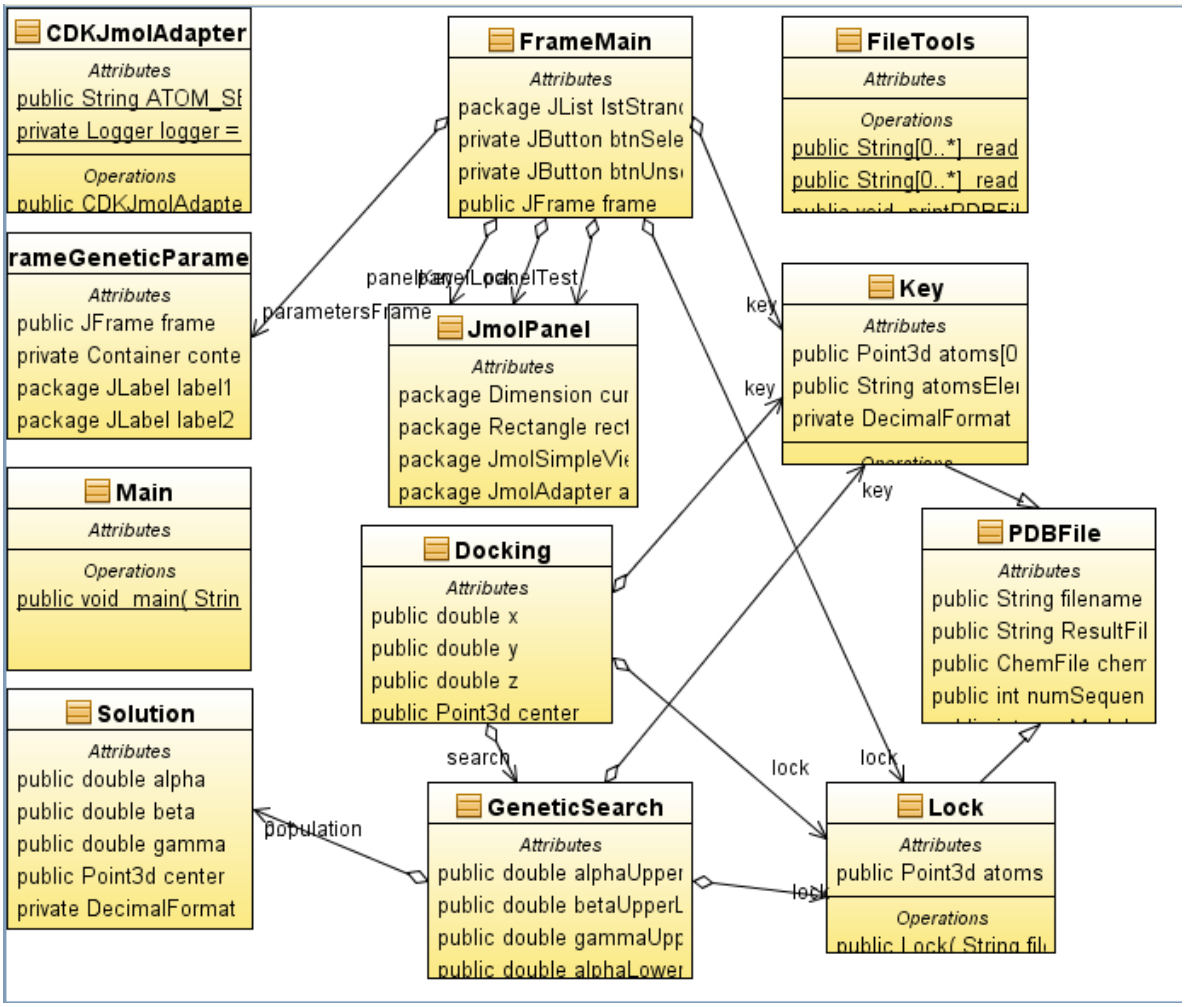


Figura 13. Diagrama de clases del proyecto molUDLAP.

Partes que contiene el proyecto molUDLAP

A continuación se muestran las partes que conforman al programa molUDLAP.

Aspectos importantes en el desarrollo de molUDLAP.

El siguiente diagrama (figura 14) muestra como está organizado el proyecto molUDLAP:

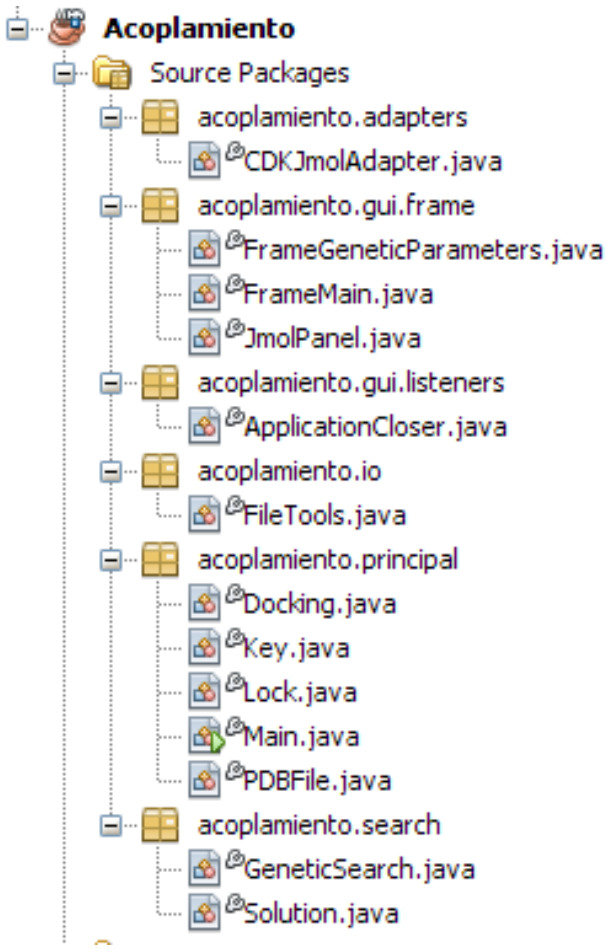


Figura 14. Contenido de paquetes y clases del proyecto molUDLAP.

Donde:

En el paquete adapters se tiene una clase que sirve como adaptador entre Jmol y CDK, de momento no se utiliza, pero se ha decidió dejarla para posibles usos en un futuro.

Se decidió poner en un solo paquete a todas aquellas clases referentes al diseño de la interfaz visual GUI.

El paquete listeners, solo se ocupa para terminar la aplicación (cerrar).

En el paquete IO se tienen clases destinadas al manejo de archivos por ejemplo para leer archivos de una unidad de disco.

En el paquete principal, se ha decidido crear una clase madre (PDBFile), la cual trae la información necesaria que se le heredará a las clases Key y Lock, estas a su vez cuentas con sus propios métodos, así mismo en la clase Docking se hace la búsqueda de la configuración optima del key para el docking, y se llama a la función de búsqueda genética, mientras que main es donde se ejecuta todo el programa.

Finalmente en el paquete search, se ha decidido poner el algoritmo de búsqueda, de forma tal que este pueda ser removido o bien mejorado sin que esto afecte el funcionamiento de lo demás.

Rotación y traslación de la molécula key

Dado que solo se tiene como molécula flexible al key, lo siguiente solo se aplica para esta molécula.

Una vez que el usuario ha proporcionado los parámetros de entrada que van a ser evaluados por molUDLAP, se lleva a cabo la simulación, en una parte de esta se tiene que rotar y trasladar la molécula key, para hacer los cálculos necesarios durante el proceso de AC.

Para esto se tienen los siguientes pasos:

1.- Se parte de la matriz identidad

$$I_1 = (1),\ I_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},\ I_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix},\ \cdots,\ I_n = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

2.- Posteriormente se procede a realizar la multiplicación por la matriz de rotación en el eje X.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \circ \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Rotación en x

3.- Posteriormente se procede a realizar la multiplicación por la matriz de rotación en el eje Y.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \circ \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Rotación en y

4.- Posteriormente se procede a realizar la multiplicación por la matriz de rotación en el eje Z.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$ **Rotación en z**

De esta manera se obtiene la rotación de la molécula.

5.- Después el resultado obtenido se multiplica por la matriz de traslación.

Con lo cual se tiene el movimiento del key, primero se rota y luego se hace el desplazamiento, para cada átomo de esta molécula y así hasta que se hayan concluido todas las configuraciones introducidas por el usuario.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Estas rotaciones y traslaciones se lograron hacer con ayuda de las funciones implementadas en Java3D.

Carga de las moléculas

Como se ha mencionado anteriormente se ha decidido utilizar a Jmol como visualizador, en lo que respecta a esta herramienta se ha usado de la siguiente manera

Se llama al método loadToJmol en la clase PDBFile, la cual ocupa la clase JmolSimpleViewer del API Jmol ,a continuación se muestra la función que hace este trabajo:

```
//  Carga un PDB al viewer
   public JmolSimpleViewer loadToJmol(PDBFile file, JmolSimpleViewer viewer)
   {
      //  Abriendo el archivo para mostrar JMOL
      String res = viewer.openFile(filename);
      if(res != null)
      {
         this.output = "";
         this.error = "Hubo un problema al cargar la molecula en Jmol.\n" + res;
      }
      else
      {
         this.output = "Molecula cargada en Jmol: OK";
         this.error = "";
      }
      return viewer;
   }
```

Mientras que CDK se usa para manipular las moléculas, con funciones como fragmentar por cadenas o bien agregar cargas de Gasteiger.

Primero se carga el archivo .PDB, con lo cual se crea un archivo de tipo ChemFile, para utilizar entonces Sequence que a su vez es utilizado por MoleculeSet, para representar a las moléculas, de aquí se parte para dividir la molécula en Strands (cadenas), que a su vez contienen AtomContainer, con lo cual se tienen objetos de tipo Atom, de esta manera obtenemos los componentes de la molécula, con lo cual la podemos manipular; cabe señalar que todo lo anterior se aplica únicamente al Lock.

A continuación se muestra parte del proceso anteriormente explicado:

```
//  Abre un archivo PDB por medio del CDK
public void openFileCDK(String filename)
{
   try
   {
      //  Abriendo el flujo de lectura del archivo PDB
      FileInputStream input = new FileInputStream(filename);
      //  Procesando el flujo con el reader adecuado
      PDBReader reader = new PDBReader(input);
      //  Creando el modelo base de la molecula
      chemFile = (ChemFile)reader.read((ChemObject)new ChemFile());

      //  Cleanning...
      reader.close();
      input.close();
      reader = null;
```

```
            input = null;
            this.output = "Datos CDK molecula: OK";
            this.error = "";
        }
        catch(Exception exc)
        {
            this.output = "";
            this.error = "Hubo un problema al extraer la información de la molecula con CDK.\n" + exc.toString();
        }
    }

    //  Extrae las cadenas de la proteina Lock
    public void getComponents()
    {
        //  El numero de secuencias en la molecula, idealmente debe ser 1
        //  Si hay dos quiere decir que ya empezamos a manipular la molecula
        numSequences = chemFile.getChemSequenceCount();

        for(int i = 0; i < numSequences; i++)
        {
            //  Debe haber una sola secuencia por archivo
            ChemSequence sequence = (ChemSequence)chemFile.getChemSequence(i);
            //  Una secuencia solo puede traer un chemModel
            numModels = sequence.getChemModelCount();
```

```
for(int ii = 0; ii < numModels; ii++)   {
   ChemModel model = (ChemModel)sequence.getChemModel(ii);
   //  Un chemModel quimico solo puede traer un MoleculeSet
   moleculeSet = (MoleculeSet)model.getMoleculeSet();
   //  Caluculando el numero de moleculas en la proteina
   numMolecules = moleculeSet.getMoleculeCount();
   //  Recuperando las moleculas
   for(int iii = 0; iii < numMolecules; iii++)
      molecules.add((Molecule)moleculeSet.getMolecule(iii));

   //  Calculando las cadenas (Chain = Strand)
   //  Creando un Biopolimero temporal
   for(int iii = 0; iii < numMolecules; iii++)
   {
      BioPolymer bp = (BioPolymer)molecules.get(iii);
      strandNames.add(new ArrayList<String>(bp.getStrandNames()));
      this.strands = bp.getStrands();
   }

   //  Cleanning...
   model = null;
}

//  Cleanning...
```

```
        sequence = null;
    }
}

//  Aplica cargas Gasteiger para la cadena seleccionada
public void applyGasteigerToSelectedStrand()
{
    try
    {
        GasteigerMarsiliPartialCharges gasteiger = new GasteigerMarsiliPartialCharges();
        gasteiger.calculateCharges((AtomContainer)this.selectedStrand);

        //  Cleanning...
        gasteiger = null;
        this.error = "";
        this.output = "Gasteiger OK";
    }
    catch(Exception exc)
    {
        this.output = "";
        this.error = "Hubo un error al aplicar Gasteiger.\n" + exc.toString();
    }
}
```

Función de optimización de energía

En un principio se trataron de usar las funcionalidades destinadas para este tipo de tarea implementadas en CDK (MMFF94EnergyFunction) y en Meshi (TotalEnergy), sin embargo como se explico estas no funcionaron, por lo que se propuso en este proyecto una función para la energía total. Esta función considera la fuerza de atracción y repulsión entre los átomos de las moléculas y la fuerza electrostática y relaciona las distancias que por defecto brinda el grid de CDK con las distancias reales a través de un parámetro de escala. También considera la ponderación de las fuerzas x^{12} y x^{6} por medio de un promedio de los coeficientes de Lennard Jones para este tipo de fuerzas. Las distancias dependen de los ángulos de rotación con respecto de cada uno de los ejes coordenados, con lo cual se obtiene diferentes configuraciones del sistema proteína-ligando.

El valor de la función de la energía es el que va a permitir decidir cuáles son las mejores configuraciones. Esta es la función que se debe optimizar. A continuación se muestra la implementación realizada en molUDLAP de esta fórmula:

```
//  Evalua la energia entre el Lock y el Key con la solucion proporcionada
   public double eval(ArrayList<Point3d> lock, ArrayList<Point3d> key)
   {
      return getSum(lock, key, 1) + getSum(lock, lock, 1) + getSum(key, key, 1);
   }

   //  Calcula la sumatoria de los arreglos y aplica la funcion deseada dependiendo del tipo de arreglo

   //  Key vs Key, Lock vs Key, Lock vs Lock
   public double getSum(ArrayList<Point3d> a1, ArrayList<Point3d> a2, int stage)
```

```
{
    double ret = 0.0;

    Point3d p1 = new Point3d();
    Point3d p2 = new Point3d();

    for(int i=0; i<a1.size(); i++)
    {
        p1 = (Point3d)a1.get(i);

        for(int j=0; j<a2.size(); j++)
        {
            p2 = (Point3d)a2.get(j);

            switch (stage)
            {
                case 1:
                    ret += getProteinProtein(getEuclidianDistance(p1,  p2));
                    break;
                case 2:
                    break;
                case 3:
                    break;
            }
        }
```

```
        }


        p1 = null;
        p2 = null;


        return ret;
    }


    //  Formula general base para el calculo de energia basada en distancias
    public double getProteinProtein(double r)
    {
        if(Math.pow(r, 12) != 0 && Math.pow(r, 6) != 0 && r != 0)
            return (560740/Math.pow(4.5*r, 12)) - (645.146/Math.pow(4.5*r, 6)) + (0.0002/(4.5*r));
        else
            return 0;
    }
```

Algoritmo de búsqueda

Para esta parte se ha decidido utilizar un algoritmo genético, ya que es una opción muy recomendada en la literatura referente al tema, se decidió ponerlo en un paquete y en clases separadas por cuestiones de implementación y para que el trabajo a futuro se facilite si se requiere mejorarlo o bien removerlo e insertar otro tipo de solución.

Cabe mencionar que por cuestiones de tiempo se decidió implementar únicamente la operación que realiza la mutación, las otras operaciones como

el cruzamiento será conveniente considerarlas para un refinamiento del algoritmo. Las mutaciones se consideran para modificar los dos tripletes (X, Y, Z) , uno con el centro de la molécula Key y otro con la orientación de esta misma, se agregan grados de más y se hace la evaluación, eliminando las peores, y así hasta que se obtenga la mejor configuración del key contra el lock, siendo esta última configuración la que se muestra al usuario.

Hay que señalar que si bien aunque durante el proceso de simulación se puedan obtener configuraciones que resulten mejores que la última configuración obtenida, estas se descartan ya que se tomará únicamente en cuenta a la última configuración resultado de la mutación y sobreviviente a través de todas las generaciones como buena, con lo que las anteriores configuraciones se podrían pensar como falsos positivos.

A continuación se muestra el algoritmo implementado.

```
//  Ejecuta la busqueda genetica
   public void Run(int toEliminate, int startNumber)
   {
      boolean next = false;
      //  Iteracion 0
      initPopulation(startNumber);
      evaluatePopulation();
      next = getSurvivors(toEliminate);
      //  Mientras existan sobrevivientes
      while(next)
```

```
    {
        mutate();
        evaluatePopulation();
        next = getSurvivors(toEliminate);
    }
    //this.key.WriteToPDB_CDK(population.get(0));
}
```